aws

개발자 가이드

# Amazon DynamoDB



API 버전 2012-08-10

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

# Amazon DynamoDB: 개발자 가이드

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

# Table of Contents

What is Amazon DynamoDB? ..... 1
특성 ..... 2
Serverless ..... 2
NoSQL ..... 2
완전관리형 ..... 2
규모를 따지지 않는 한 자릿수 밀리초의 성능 ..... 2
사용 사례 ..... 3
기능 ..... 4
전역 테이블을 사용한 다중 활성 복제 ..... 4
ACID 트랜잭션 ..... 4
변경 데이터 캡처 ..... 4
보조 인덱스 ..... 4
서비스 통합 ..... 5
서버리스 통합 ..... 5
Amazon S3로 데이터 가져오기 및 내보내기 ..... 5
제로 ETL 통합 ..... 5
캐싱 ..... 6
보안 ..... 6
복원성 ..... 7
글로벌 테이블 ..... 7
연속 백업과 특정 시점으로 복구 ..... 7
온디맨드 백업 및 복원 ..... 7
DynamoDB 액세스 ..... 8
요금 ..... 8
시작 ..... 8
DynamoDB 시작하기 ..... 10
신규 사용자 리소스 ..... 11
신규 사용자를 위한 Amazon DynamoDB 추가 모범 사례 ..... 12
AWS CLI 리소스 ..... 12
프로그래밍 리소스 ..... 12
DynamoDB 액세스 ..... 13
콘솔 사용 ..... 13
AWS CLI 사용 ..... 14
API 사용 ..... 16

NoSQL 워크벤치 사용 ..... 16
IP 주소 범위 ..... 17
사전 조건 ..... 18
DynamoDB 설정 ..... 18
DynamoDB 설정(웹 서비스) ..... 19
DynamoDB local 설정(다운로드 가능 버전) ..... 22
1단계: 테이블 생성 ..... 43
2단계: 데이터 쓰기 ..... 96
3단계: 데이터 읽기 ..... 128
4단계: 데이터 업데이트 ..... 154
5단계: 데이터 쿼리 ..... 187
6단계: (선택 사항) 정리 ..... 224
다음 단계 ..... 242
작동 방법 ..... 244
치트 시트 ..... 244
초기 설정 ..... 244
SDK 또는 CLI ..... 244
기본 작업 ..... 245
이름 지정 규칙 ..... 246
서비스 할당량 기본 사항 ..... 247
추가 정보 ..... 248
핵심 구성 요소 ..... 249
테이블, 항목 및 속성 ..... 249
프라이머리 키 ..... 253
보조 인덱스 ..... 254
DynamoDB Streams ..... 257
DynamoDB API ..... 258
컨트롤 플레인 ..... 259
데이터 영역 ..... 259
DynamoDB Streams ..... 261
트랜잭션 ..... 261
지원되는 데이터 형식 및 이름 지정 규칙 ..... 262
이름 지정 규칙 ..... 262
데이터 타입 ..... 263
데이터 형식 서술자 ..... 268
DynamoDB 테이블 클래스 ..... 268

DynamoDB의 파티션 및 데이터 배포 ... 269
데이터 배포: 파티션 키 ... 269
데이터 배포: 파티션 키 및 정렬 키 ... 271
SQL에서 NoSQL로 전환하는 방법 알아보기 ... 273
관계형 또는 NoSQL? ... 274
액세스 및 인증 ... 276
테이블 생성 ... 280
테이블에 대한 정보 가져오기 ... 282
테이블에 데이터 쓰기 ... 284
테이블에서 데이터 읽기 ... 288
인덱스 관리 ... 296
테이블의 데이터 수정 ... 301
테이블에서 데이터 삭제 ... 305
테이블 제거 ... 307
Amazon DynamoDB 학습 리소스 및 도구 ... 308
코딩 및 시각화를 위한 도구 ... 308
권장 가이드 ... 309
지식 센터 ... 310
블로그 게시물, 리포지토리 및 가이드 ... 311
데이터 모델링 및 디자인 패턴 ... 311
교육 과정 ... 312
읽기 및 쓰기 ... 313
DynamoDB 읽기 일관성 ... 313
최종적으로 일관된 읽기 ... 313
강력하게 일관된 읽기 ... 314
글로벌 테이블 읽기 일관성 ... 314
읽기 및 쓰기 작업 ... 314
읽기 작업 소비량 ... 314
쓰기 작업 소비량 ... 316
DynamoDB 처리량 용량 ... 318
온디맨드 모드 ... 318
프로비저닝된 모드 ... 319
DynamoDB 온디맨드 용량 모드 ... 319
읽기 요청 단위 및 쓰기 요청 단위 ... 320
초기 처리량 및 규모 조정 속성 ... 321
온디맨드 테이블의 최대 처리량 ... 321

프로비저닝된 용량 모드 ... 323
읽기 및 쓰기 용량 단위 ... 325
초기 처리량 설정 선택 ... 325
DynamoDB Auto Scaling ... 326
Auto Scaling으로 처리량 용량 관리 ... 327
예약 용량 ... 357
웜 처리량 ... 357
테이블의 웜 처리량 확인 ... 358
테이블의 웜 처리량 증가 ... 361
웜 처리량이 더 높은 테이블 만들기 ... 368
웜 처리량 시나리오 ... 379
버스트 및 조정 용량 ... 381
버스트 용량 ... 381
조정 용량 ... 381
용량 모드 전환 ... 382
프로비저닝된 모드에서 온디맨드 모드로 ... 383
온디맨드 모드에서 프로비저닝된 모드로 ... 384
DynamoDB를 사용한 프로그래밍 ... 386
DynamoDB에 대한 AWS SDK 지원 개요 ... 386
SDK에서 AWS 계정 기반 엔드포인트 지원 ... 388
DynamoDB에서 작동하는 프로그래밍 인터페이스 ... 389
높은 수준의 프로그래밍 인터페이스 ... 396
코드 예시 실행 ... 455
하위 수준 API ... 462
Python을 사용한 프로그래밍 ... 467
Boto 소개 ... 468
Boto 설명서 ... 468
클라이언트 및 리소스 계층 ... 469
batch_writer 사용 ... 472
추가 코드 예시 ... 472
세션 및 스레드 안전 ... 473
구성 ... 473
오류 처리 ... 478
로깅 ... 480
이벤트 후크 ... 481
페이지 매김 및 페이지네이터 ... 482

Waiters ... 484
JavaScript를 사용한 프로그래밍 ... 485
AWS SDK for JavaScript 정보 ... 485
AWS SDK for JavaScript V3 ... 486
JavaScript 설명서 ... 486
추상화 계층 ... 486
마샬 유틸리티 함수 ... 489
항목 읽기 ... 489
조건부 쓰기 ... 491
페이지 매김 ... 492
구성 ... 494
Waiters ... 497
오류 처리 ... 497
로깅 ... 499
고려 사항 ... 500
AWS SDK for Java 2.x를 사용한 프로그래밍 ... 501
AWS SDK for Java 2.x 소개 ... 501
시작 ... 502
SDK for Java 2.x 설명서 ... 511
지원되는 인터페이스 ... 512
추가 코드 예시 ... 526
동기식 및 비동기식 프로그래밍 ... 526
HTTP 클라이언트 ... 527
구성 ... 529
오류 처리 ... 535
AWS 요청 ID ... 536
로깅 ... 536
페이지 매김 ... 539
데이터 클래스 주석 ... 540
오류 처리 ... 540
오류 구성 요소 ... 541
트랜잭션 오류 ... 541
오류 메시지 및 코드 ... 542
애플리케이션에서의 오류 처리 ... 546
오류 재시도 횟수 및 지수 백오프 ... 546
일괄 작업 및 오류 처리 ... 547

AWS SDK 작업 ... 548
DynamoDB 작업 ... 550
테이블 작업 ... 550
테이블에 대한 기본 작업 ... 550
DynamoDB에서 테이블 클래스 선택 시 고려 사항 ... 559
태그 및 레이블 ... 560
글로벌 테이블 작업 ... 566
작동 방법 ... 567
자습서: 글로벌 테이블 만들기 ... 574
글로벌 테이블 보안 ... 611
글로벌 테이블 요금 청구 ... 616
글로벌 테이블 버전 ... 618
글로벌 테이블 모범 사례 ... 629
항목 작업 ... 630
항목 크기 및 형식 ... 631
항목 읽기 ... 632
항목 쓰기 ... 633
반환 값 ... 636
배치 작업 ... 637
원자성 카운터 ... 639
조건부 쓰기 ... 640
표현식 사용 ... 645
TTL(Time To Live) ... 683
테이블 쿼리 ... 712
테이블 스캔 ... 721
PartiQL 쿼리 언어 ... 729
항목 작업: Java ... 775
항목 작업: .NET ... 806
인덱스 작업 ... 839
글로벌 보조 인덱스 ... 843
DynamoDB의 로컬 보조 인덱스 ... 900
트랜잭션 작업 ... 952
작동 방법 ... 953
트랜잭션에서 IAM 사용 ... 962
예제 코드 ... 965
스트림 작업 ... 969

옵션 ..... 970
Kinesis Data Streams 작업 ..... 971
DynamoDB Streams 작업 ..... 988
DAX를 통한 인 메모리 가속화 ..... 1069
DAX 사용 사례 ..... 1070
DAX 사용 참고 사항 ..... 1071
작동 방식 ..... 1072
DAX에서 요청을 처리하는 방식 ..... 1073
항목 캐시 ..... 1075
쿼리 캐시 ..... 1076
DAX 클러스터 구성 요소 ..... 1076
노드 ..... 1077
클러스터 ..... 1077
리전 및 가용성 영역 ..... 1079
파라미터 그룹 ..... 1079
보안 그룹 ..... 1079
클러스터 ARN ..... 1080
클러스터 엔드포인트 ..... 1080
노드 엔드포인트 ..... 1080
서브넷 그룹 ..... 1080
이벤트 ..... 1081
유지보수 윈도우 ..... 1081
DAX 클러스터 생성 ..... 1082
DAX에 대한 IAM 서비스 역할을 생성하여 DynamoDB 액세스 ..... 1083
AWS CLI 사용 ..... 1085
콘솔 사용 ..... 1091
정합성 모델 ..... 1096
DAX 클러스터 노드 간 정합성 ..... 1096
DAX 항목 캐시 동작 ..... 1097
DAX 쿼리 캐시 동작 ..... 1100
강력히 정합하는 트랜잭션 읽기 ..... 1101
음성 캐싱 ..... 1101
쓰기 전략 ..... 1101
DAX 클라이언트로 개발 ..... 1105
자습서: 샘플 애플리케이션 실행 ..... 1105
DAX를 사용하도록 기존 애플리케이션 수정 ..... 1168

DAX 클러스터 관리 ..... 1169
DAX 클러스터 관리를 위한 IAM 권한 ..... 1170
DAX 클러스터 크기 조정 ..... 1172
DAX 클러스터 설정 사용자 지정 ..... 1174
TTL 설정 구성 ..... 1175
DAX에 대한 태그 지정 지원 ..... 1176
AWS CloudTrail 통합 ..... 1177
DAX 클러스터 삭제 ..... 1177
Monitoring DynamoDB Accelerator ..... 1178
DAX 모니터링 도구 ..... 1178
CloudWatch를 사용하여 모니터링 ..... 1180
AWS CloudTrail을 사용하여 DAX 작업 로깅 ..... 1203
DAX T3/T2 버스트 가능 인스턴스 ..... 1204
DAX T2 인스턴스 패밀리 ..... 1204
DAX T3 인스턴스 패밀리 ..... 1204
DAX 액세스 제어 ..... 1205
DAX의 IAM 서비스 역할 ..... 1206
DAX 클러스터 액세스를 허용하는 IAM 정책 ..... 1208
사례 연구: DynamoDB 및 DAX 액세스 ..... 1209
DynamoDB에 액세스할 수는 있지만 DAX로는 액세스할 수 없음 ..... 1210
DynamoDB 및 DAX에 대한 액세스 ..... 1212
DAX를 통해 DynamoDB에 액세스할 수 있지만 DynamoDB에 직접 액세스할 수는 없음 ..... 1218
DAX 저장 데이터 암호화 ..... 1221
AWS Management Console을 사용하여 저장 데이터 암호화 활성화 ..... 1223
DAX 전송 중 데이터 암호화 ..... 1224
DAX에 대한 서비스 연결 역할 사용 ..... 1225
DAX에 대한 서비스 연결 역할 권한 ..... 1225
DAX에 대한 서비스 연결 역할 생성 ..... 1227
DAX에 대한 서비스 연결 역할 편집 ..... 1227
DAX에 대한 서비스 연결 역할 삭제 ..... 1227
AWS 계정 간 DAX 액세스 ..... 1229
IAM 설정 ..... 1229
VPC 설정 ..... 1232
크로스 계정 액세스를 허용하도록 DAX 클라이언트 수정 ..... 1235
DAX 클러스터 크기 조정 안내서 ..... 1239
개요 ..... 1240

트래픽 추정 ..... 1240
로드 테스트. ..... 1241
데이터 모델링 ..... 1243
항목 컬렉션 작업 ..... 1244
항목 컬렉션으로 데이터를 구성하여 쿼리 속도 향상 ..... 1245
데이터 모델링 기초 ..... 1246
단일 테이블 설계 ..... 1246
다중 테이블 설계 ..... 1248
데이터 모델링 빌딩 블록 ..... 1250
복합 정렬 키 ..... 1251
멀티테넌시 ..... 1252
희소 인덱스 ..... 1253
TTL(Time To Live) ..... 1254
TTL(Time To Live) 아카이브 ..... 1255
수직 파티셔닝 ..... 1256
쓰기 샤딩 ..... 1259
데이터 모델링 스키마 설계 패키지 ..... 1260
사전 조건 ..... 1261
소셜 네트워크 ..... 1262
게임 프로필 ..... 1271
불만 관리 시스템 ..... 1280
반복 결제 ..... 1297
디바이스 상태 업데이트 ..... 1302
온라인 상점 ..... 1315
관계형 모델링 ..... 1338
기존 관계형 데이터베이스 모델 ..... 1339
DynamoDB를 통해 JOIN 작업의 필요성을 없애는 방법 ..... 1341
DynamoDB 트랜잭션이 쓰기 프로세스의 오버헤드를 제거하는 방법 ..... 1342
첫 번째 단계 ..... 1343
예 ..... 1345
DynamoDB로 마이그레이션 ..... 1349
마이그레이션하는 이유 ..... 1349
마이그레이션 시 고려 사항 ..... 1350
작동 방식 ..... 1352
마이그레이션 도구 ..... 1353
마이그레이션 전략 선택 ..... 1353

오프라인 마이그레이션 ..... 1355
하이브리드 마이그레이션 ..... 1357
온라인 - 각 테이블을 일대일로 마이그레이션 ..... 1359
온라인 - 사용자 지정 스테이징 테이블을 사용한 마이그레이션 ..... 1360
NoSQL Workbench ..... 1363
다운로드 ..... 1364
Install ..... 1365
데이터 모델 제작자 ..... 1372
새 모델 생성 ..... 1372
기존 모델 가져오기 ..... 1379
모델 내보내기 ..... 1382
기존 모델 편집 ..... 1384
데이터 시각화 도우미 ..... 1387
샘플 데이터 추가하기 ..... 1388
CSV에서 가져오기 ..... 1391
패싯 ..... 1392
집계 보기 ..... 1395
데이터 모델 커밋 ..... 1396
작업 빌더 ..... 1399
데이터 세트에 연결 ..... 1399
작업 빌드 ..... 1401
테이블 복제 ..... 1412
CSV로 내보내기 ..... 1413
샘플 데이터 모델 ..... 1414
직원 데이터 모델 ..... 1414
토론 포럼 데이터 모델 ..... 1414
음악 라이브러리 데이터 모델 ..... 1415
스키장 데이터 모델 ..... 1415
신용카드 제안 데이터 모델 ..... 1416
북마크 데이터 모델 ..... 1416
릴리스 이력 ..... 1417
백업 및 복원 ..... 1422
시점 백업 ..... 1423
시작하기 전 준비 사항 ..... 1423
시점 복구 활성화 ..... 1424
온디맨드 백업 ..... 1428

작동 방식 ... 1429
테이블 백업 ... 1432
테이블 복원 ... 1434
테이블 백업 삭제 ... 1440
IAM 사용 ... 1441
백업에 대한 요금 청구 ... 1448
작동 방법 ... 1449
DynamoDB 백업 요금 청구 예시 ... 1449
복원 ... 1453
특정 시점으로 복구를 사용하여 테이블 복원 ... 1453
DynamoDB 테이블을 특정 시점으로 복원 ... 1455
AWS 백업 사용 ... 1461
작동 방식 ... 1462
백업 생성 ... 1465
백업 복사 ... 1467
테이블 복원 ... 1468
백업 삭제 ... 1469
AWS Backup에서 관리되는 온디맨드 백업과 DynamoDB 비교 ... 1469
코드 예제 ... 1471
기본 사항 ... 1484
Hello DynamoDB ... 1485
기본 사항 알아보기 ... 1495
작업 ... 1651
시나리오 ... 2053
DAX로 읽기 가속화 ... 2056
고급 글로벌 보조 인덱스 시나리오 ... 2064
DynamoDB 테이블에 데이터를 제출하기 위한 앱 구축 ... 2067
단일 속성과 여러 값 비교 ... 2068
항목의 TTL을 조건부로 업데이트 ... 2090
로컬 인스턴스에 연결 ... 2097
표현식 연산자 수 계산 ... 2099
COVID-19 데이터를 추적하는 REST API 생성 ... 2131
메신저 애플리케이션 생성 ... 2132
사진을 관리하기 위한 서버리스 애플리케이션 만들기 ... 2133
글로벌 보조 인덱스가 있는 테이블 만들기 ... 2137
웜 처리량이 활성화된 테이블 만들기 ... 2141

DynamoDB 데이터를 추적하는 웹 애플리케이션 만들기 ..... 2151
WebSocket 채팅 애플리케이션 생성 ..... 2152
TTL을 사용하여 항목 생성 ..... 2153
MRSC 글로벌 테이블 만들기 및 관리 ..... 2160
MREC를 보여주는 글로벌 테이블 만들기 및 관리 ..... 2180
PartiQL DELETE를 사용하여 데이터 삭제 ..... 2189
이미지에서 PPE 감지 ..... 2196
PartiQL INSERT를 사용하여 데이터 삽입 ..... 2196
브라우저에서 Lambda 함수 간접 호출 ..... 2200
글로벌 보조 인덱스 관리 ..... 2201
리소스 기반 정책 관리 ..... 2204
DynamoDB 성능 모니터링 ..... 2206
고급 쿼리 작업 수행 ..... 2207
목록 작업 수행 ..... 2242
맵 작업 수행 ..... 2274
집합 작업 수행 ..... 2298
PartiQL 문 배치를 사용하여 테이블 쿼리 ..... 2329
PartiQL을 사용하여 테이블 쿼리 ..... 2391
글로벌 보조 인덱스를 사용하여 테이블 쿼리 ..... 2445
begins_with 조건을 사용하여 테이블 쿼리 ..... 2454
날짜 범위를 사용하여 테이블 쿼리 ..... 2459
복잡한 필터 표현식을 사용하여 테이블 쿼리 ..... 2468
동적 필터 표현식을 사용하여 테이블 쿼리 ..... 2476
필터 표현식 및 제한을 사용하여 테이블 쿼리 ..... 2485
중첩 속성을 사용하여 테이블 쿼리 ..... 2489
페이지 매김을 사용하여 테이블 쿼리 ..... 2498
강력하게 일관된 읽기를 사용하여 테이블 쿼리 ..... 2509
PartiQL SELECT를 사용하여 데이터 쿼리 ..... 2513
TTL 항목에 대한 쿼리 ..... 2519
날짜 및 시간 패턴을 사용하여 테이블 쿼리 ..... 2523
EXIF 및 기타 이미지 정보 저장 ..... 2537
속성 기반 액세스 제어 설정 ..... 2538
업데이트 표현식 순서 이해 ..... 2541
테이블의 웜 처리량 설정 업데이트 ..... 2572
항목의 TTL 업데이트 ..... 2578
PartiQL UPDATE를 사용하여 데이터 업데이트 ..... 2582

API Gateway를 사용하여 Lambda 함수 간접 호출 ..... 2589
Step Functions를 사용하여 Lambda 함수 간접 호출 ..... 2590
문서 모델 사용 ..... 2591
상위 수준 객체 지속성 모델 사용 ..... 2607
원자성 카운터 작업 사용 ..... 2616
조건부 작업 사용 ..... 2639
표현식 속성 이름 사용 ..... 2669
예약된 이벤트를 사용하여 Lambda 함수 간접 호출 ..... 2697
로컬 보조 인덱스로 작업 ..... 2699
Streams 및 Time-to-Live 작업 ..... 2701
글로벌 테이블 및 다중 리전 복제 최종 일관성(MREC) 작업 ..... 2705
리소스 태그 지정 작업 ..... 2708
테이블 암호화 작업 ..... 2711
서버리스 예제 ..... 2713
DynamoDB 트리거에서 간접적으로 Lambda 함수 간접 호출 ..... 2713
DynamoDB 트리거로 Lambda 함수에 대한 배치 항목 실패 보고 ..... 2722
AWS 커뮤니티 기여 ..... 2734
서버리스 애플리케이션 빌드 및 테스트 ..... 2734
보안 ..... 2736
AWS 관리형 정책 ..... 2737
AWS 관리형 정책 ..... 2737
AWS 관리형 정책: AmazonDynamoDBFullAccess_v2 ..... 2737
AmazonDynamoDBReadOnlyAccess ..... 2738
AWS 관리형 정책으로 DynamoDB 업데이트 ..... 2739
리소스 기반 정책 ..... 2741
테이블 생성 ..... 2742
리소스 기반 정책 연결 ..... 2748
스트림에 정책 연결 ..... 2752
리소스 기반 정책 제거 ..... 2755
크로스 계정 액세스 ..... 2756
퍼블릭 액세스 차단 ..... 2757
API 작업 ..... 2760
IAM 권한 부여 ..... 2768
예시 ..... 2768
고려 사항 ..... 2776
모범 사례 ..... 2777

속성 기반 액세스 제어 ..... 2778
ABAC를 사용해야 하는 이유는 무엇입니까? ..... 2780
조건 키 ..... 2780
고려 사항 ..... 2780
DynamoDB에서 ABAC 활성화 ..... 2781
ABAC 사용 ..... 2783
사용 사례 예제 ..... 2785
문제 해결 ..... 2798
데이터 보호 ..... 2799
저장 시 암호화 ..... 2800
DynamoDB 연결 보안 ..... 2825
IAM ..... 2826
ID 및 액세스 관리 ..... 2827
조건 사용 ..... 2860
규정 준수 확인 ..... 2883
복원성 ..... 2884
인프라 보안 ..... 2885
VPC 엔드포인트 사용 ..... 2885
AWS PrivateLink for DynamoDB ..... 2894
Amazon VPC 엔드포인트의 유형 ..... 2895
AWS PrivateLink for Amazon DynamoDB 사용 시 고려 사항 ..... 2896
Amazon VPC 엔드포인트 생성 ..... 2897
Amazon DynamoDB 인터페이스 엔드포인트에 액세스 ..... 2897
DynamoDB 인터페이스 엔드포인트에서 DynamoDB 테이블에 및 제어 API 작업에 액세스 . 2897
온프레미스 DNS 구성 업데이트 ..... 2900
Amazon VPC 엔드포인트 정책 생성 ..... 2902
AWS PrivateLink for DynamoDB Streams ..... 2903
구성 및 취약성 분석 ..... 2908
보안 모범 사례 ..... 2909
예방적 보안 모범 사례 ..... 2909
탐지 보안 모범 사례 ..... 2911
모니터링 및 로깅 ..... 2915
모니터링 계획 ..... 2915
성능 기준 ..... 2915
통합 서비스 ..... 2916
자동 모니터링 도구 ..... 2916

지표 모니터링 ..... 2917
DynamoDB 지표 사용 방법 ..... 2917
CloudWatch 콘솔에서 지표 보기 ..... 2918
AWS CLI에서 지표 보기 ..... 2919
지표 및 차원 ..... 2920
DynamoDB에 CloudWatch 경보 생성 ..... 2945
로깅 작업 ..... 2949
CloudTrail의 DynamoDB 정보 ..... 2949
DynamoDB 로그 파일 항목 이해 ..... 2953
Contributor Insights ..... 2972
작동 방법 ..... 2973
시작하기 ..... 2979
IAM 사용 ..... 2985
모범 사례 ..... 2991
NoSQL 설계 ..... 2991
NoSQL 및 RDBMS ..... 2992
두 가지 주요 개념 ..... 2992
일반적인 접근법 ..... 2993
NoSQL Workbench ..... 2994
DynamoDB Well-Architected 렌즈 ..... 2994
비용 최적화 ..... 2994
Amazon DynamoDB Well-Architected 렌즈 검토 실시 ..... 3041
Amazon DynamoDB Well-Architected 렌즈의 요소 ..... 3041
파티션 키 설계 ..... 3043
워크로드 배포 ..... 3044
쓰기 샤딩 ..... 3045
효율적으로 데이터 업로드 ..... 3046
정렬 키 설계 ..... 3048
버전 제어 ..... 3048
보조 인덱스 ..... 3049
일반 지침 ..... 3050
희소 인덱스 ..... 3053
집계 ..... 3054
GSI 오버로딩 ..... 3055
GSI 샤딩 ..... 3057
복제본 생성 ..... 3061

큰 항목 ..... 3062
압축 ..... 3063
수직 파티셔닝 ..... 3063
Amazon S3 사용 ..... 3063
시계열 데이터 ..... 3064
시계열 데이터의 설계 패턴 ..... 3064
시계열 테이블 예 ..... 3065
다대다 관계 ..... 3066
인접 목록 ..... 3066
구체화된 그래프 ..... 3067
쿼리 및 스캔 ..... 3072
스캔 성능 ..... 3072
급증 방지 ..... 3072
병렬 스캔 ..... 3075
테이블 설계 ..... 3076
글로벌 테이블 설계 ..... 3077
글로벌 테이블 설계 ..... 3077
주요 사실 ..... 3078
사용 사례 ..... 3079
쓰기 모드 ..... 3080
라우팅 요청 ..... 3087
리전 대피 ..... 3096
글로벌 테이블을 사용한 처리량 용량 ..... 3098
글로벌 테이블 체크리스트 및 FAQ ..... 3100
컨트롤 플레인 ..... 3106
대량 데이터 작업 ..... 3106
조건부 배치 업데이트 ..... 3107
효율적인 대량 작업 ..... 3112
버전 관리 구현 ..... 3113
이 패턴을 사용해야 하는 경우 ..... 3113
패턴 설계 ..... 3114
패턴 사용 ..... 3115
결제 및 사용량 보고서 ..... 3117
처리량 용량 ..... 3120
스트림 ..... 3124
스토리지 ..... 3125

백업 및 복원 ..... 3126
데이터 전송 ..... 3129
CloudWatch ..... 3129
DAX ..... 3130
DynamoDB 테이블을 한 계정에서 다른 계정으로 마이그레이션 ..... 3132
교차 계정 백업 및 복원 시 AWS Backup을 사용하여 테이블 마이그레이션 ..... 3132
S3로 내보내기를 사용하여 테이블을 마이그레이션하고 S3에서 가져오기 ..... 3134
DAX 권장 가이드 ..... 3136
DAX의 적합성 평가 ..... 3137
DAX 클라이언트 구성 ..... 3139
DAX 클러스터 구성 ..... 3141
DAX 클러스터 크기 조정 ..... 3146
클러스터 배포 ..... 3152
클러스터 작업 ..... 3154
DAX 모니터링 ..... 3156
다른 AWS 서비스와 함께 DynamoDB 사용 ..... 3159
Amazon Cognito와 통합 ..... 3159
Amazon Redshift와 통합 ..... 3162
CMK와 교차 계정 통합 시 고려 사항 ..... 3162
Amazon Redshift 제로 ETL 통합 ..... 3166
COPY를 사용하여 DynamoDB에서 Amazon Redshift로 데이터 로드 ..... 3174
Amazon EMR과 통합 ..... 3175
개요 ..... 3176
자습서: Amazon DynamoDB 및 Apache Hive 작업 ..... 3176
Hive에서 외부 테이블 생성 ..... 3185
HiveQL 문 처리 ..... 3188
DynamoDB의 데이터 쿼리 ..... 3190
Amazon DynamoDB 간 데이터 복사 ..... 3192
성능 튜닝 ..... 3206
S3와 통합 ..... 3211
Amazon S3에서 가져오기 ..... 3211
Amazon S3로 내보내기 ..... 3233
Amazon SageMaker Lakehouse와 통합 ..... 3255
Amazon SageMaker Lakehouse와의 제로 ETL 통합 ..... 3255
Amazon OpenSearch Service와 통합 ..... 3257
작동 방식 ..... 3257

통합 생성 ..... 3258
다음 단계 ..... 3258
영향을 미치는 변경 사항 처리 ..... 3259
OpenSearch Service와 제로 ETL 통합 ..... 3262
Amazon EventBridge 통합 ..... 3264
작동 방식 ..... 3265
콘솔을 통한 통합 생성 ..... 3266
다음 단계 ..... 3268
Amazon MSK와 통합 ..... 3268
작동 방법 ..... 3269
통합 예제 ..... 3270
다음 단계 ..... 3276
통합 모범 사례 ..... 3276
스냅샷 생성 ..... 3276
변경 데이터 캡처 ..... 3277
DynamoDB에서 생성형 AI 사용 ..... 3278
DynamoDB용 생성형 AI 사용 사례 ..... 3278
DynamoDB용 생성형 AI 블로그 ..... 3279
OpenSearch Service와 DynamoDB의 제로 ETL 통합 활용 ..... 3279
할당량 및 제약 조건 ..... 3281
할당량 관리 작업 수행 ..... 3281
DynamoDB 할당량 액세스 ..... 3281
콘솔에서 현재 할당량 보기 ..... 3282
AWS CLI를 사용하여 현재 할당량 보기 ..... 3282
할당량 증가 요청 ..... 3284
할당량 ..... 3285
읽기/쓰기 처리량 ..... 3286
예약 용량 ..... 357
표 ..... 3288
글로벌 테이블 ..... 3289
보조 인덱스 ..... 3290
프로젝션된 보조 인덱스 속성 ..... 3290
DynamoDB Streams ..... 3290
Amazon S3에서 가져오기 ..... 3291
Amazon S3로 테이블 내보내기 ..... 3291
백업 및 복원 ..... 3291

Contributor Insights ..... 3291
제약 조건 ..... 3291
읽기/쓰기 용량 모드 ..... 3292
보조 인덱스 ..... 3290
파티션 키와 정렬 키 ..... 3293
이름 지정 규칙 ..... 3294
데이터 타입 ..... 3295
Items ..... 3296
속성 ..... 3296
표현식 파라미터 ..... 3297
DynamoDB Transactions ..... 3297
DynamoDB Streams ..... 3298
DynamoDB Accelerator(DAX) ..... 3298
API별 제약 조건 ..... 3299
DynamoDB 저장 데이터 암호화 ..... 3301
API 참조 ..... 3303
문제 해결 ..... 3304
내부 서버 오류 ..... 3304
내부 서버 오류 조사 ..... 3304
내부 서버 오류로 인한 영향 최소화 ..... 3305
운영 상황 파악 개선 ..... 3306
지연 시간 ..... 3309
제한(Throttling) 문제 ..... 3312
프로비저닝된 모드 ..... 3313
온디맨드 모드 ..... 3314
CloudWatch 지표 사용 ..... 3316
부록 ..... 3318
DynamoDB와의 SSL/TLS 연결 설정 문제 해결 ..... 3318
애플리케이션 또는 서비스 테스트 ..... 3318
클라이언트 브라우저 테스트 ..... 3319
소프트웨어 애플리케이션 클라이언트 업데이트 ..... 3319
클라이언트 브라우저 업데이트 ..... 3320
수동으로 인증서 번들 업데이트 ..... 3320
DynamoDB에서 사용할 테이블 및 데이터 예제 ..... 3320
샘플 데이터 파일 ..... 3321
예시 테이블 생성 및 데이터 업로드 ..... 3334

예시 테이블 생성 및 데이터 업로드 - Java ..... 3335
예시 테이블 생성 및 데이터 업로드 - .NET ..... 3345
AWS SDK for Python (Boto3)을 사용하는 예시 애플리케이션 ..... 3357
1단계: 로컬 배포 및 테스트 ..... 3358
2단계: 데이터 모델 및 구현 세부 정보 검사 ..... 3363
3단계: 프로덕션 내 배포 ..... 3372
4단계: 리소스 정리 ..... 3381
DynamoDB의 예약어 ..... 3381
AWS SDK for Java 1.x 예제 ..... 3394
DAX 및 Java SDK v1 ..... 3395
DAX를 사용하도록 기존 SDK for Java 1.x 애플리케이션 수정 ..... 3407
SDK for Java 1.x를 사용하여 글로벌 보조 인덱스 쿼리 ..... 3412
AWS SDK for Go 1.x 예제 ..... 3415
Go 및 DAX ..... 3416
AWS SDK for Node.js 2.x 예제 ..... 3418
Node.js 및 DAX ..... 3418
문서 기록 ..... 3429
이전 업데이트 ..... 3458
레거시 기능 ..... 3488
글로벌 테이블 버전 2017.11.29(레거시) ..... 3488
작동 방법 ..... 3489
모범 사례 및 요구 사항 ..... 3494
전역 테이블 생성 ..... 3497
전역 테이블 모니터링 ..... 3502
전역 테이블에 IAM 사용 ..... 3503
이전 하위 수준 API 버전(2011-12-05) ..... 3506
BatchGetItem ..... 3507
BatchWriteItem ..... 3514
CreateTable ..... 3521
DeleteItem ..... 3527
DeleteTable ..... 3533
DescribeTables ..... 3537
GetItem ..... 3541
ListTables ..... 3545
PutItem ..... 3548
Query ..... 3555

스캔 ..... 3568
UpdateItem ..... 3584
UpdateTable ..... 3592
레거시 DynamoDB 조건부 파라미터 ..... 3597
AttributesToGet ..... 3599
AttributeUpdates ..... 3600
ConditionalOperator ..... 3602
예상 ..... 3603
KeyConditions ..... 3608
QueryFilter ..... 3611
ScanFilter ..... 3613
기존 파라미터를 이용한 조건 작성 ..... 3615

# What is Amazon DynamoDB?

Amazon DynamoDB는 모든 규모에서 10밀리초 미만의 성능을 제공하는 서버리스, NoSQL, 완전관리형 데이터베이스입니다.

DynamoDB는 관계형 데이터베이스의 규모 조정 및 운영 복잡성을 극복하기 위한 요구 사항을 해결합니다. DynamoDB는 규모와 관계없이 일관된 성능이 필요한 운영 워크로드에 맞게 특별히 구축되고 최적화되었습니다. 예를 들어, DynamoDB는 사용자가 10명이든 1억 명이든 상관없이 장바구니 사용 사례에 대해 일관된 한 자릿수 밀리초 성능을 제공합니다. 2012년에 출시된 DynamoDB는 비용을 절감하고 대규모로 성능을 개선하면서 관계형 데이터베이스에서 벗어날 수 있도록 지속적으로 지원합니다.

모든 규모, 산업, 지역의 고객이 DynamoDB를 통해 소규모로 시작하여 전 세계로 규모를 조정할 수 있는 최신 서버리스 애플리케이션을 빌드합니다. DynamoDB는 거의 모든 크기의 테이블을 지원하도록 규모를 조정할 수 있는 동시에 일관된 10밀리초 미만의 성능과 고가용성을 제공합니다.

Amazon Prime Day와 같은 이벤트가 있을 때 DynamoDB는 Alexa, Amazon.com 사이트, 모든 Amazon 주문 처리 센터를 포함하여 트래픽이 많은 여러 Amazon 자산 및 시스템을 지원합니다. 이러한 이벤트에서 DynamoDB API는 Amazon 자산 및 시스템의 직접 호출 수조 건을 처리했습니다. DynamoDB는 초당 50만 개 이상의 최대 트래픽이 요청되는 테이블을 통해 수백 명의 고객에게 지속적으로 서비스를 제공합니다. 또한, 테이블 크기가 200TB를 초과하는 수백 명의 고객에게 서비스를 제공하고 시간당 10억 개 이상의 요청을 처리합니다.

주제


- DynamoDB의 특징
- DynamoDB 사용 사례
- DynamoDB의 기능
- 서비스 통합
- 보안
- 복원성
- DynamoDB 액세스
- DynamoDB 요금
- DynamoDB 시작하기

# DynamoDB의 특징

## Serverless

DynamoDB를 사용하면 서버를 프로비저닝하거나 소프트웨어를 패치, 관리, 설치, 유지 보수 또는 운영할 필요가 없습니다. DynamoDB는 가동 중지가 발생하지 않는 유지 관리를 제공합니다. 버전(메이저, 마이너 또는 패치)이 없으며 유지 관리 기간도 없습니다.

DynamoDB의 온디맨드 용량 모드는 읽기 및 쓰기 요청에 사용량에 따른 요금이 적용되므로, 사용하는 만큼에 대해서만 비용을 지불하면 됩니다. 온디맨드를 사용하면 DynamoDB는 즉시 테이블을 스케일 업하거나 스케일 다운하여 용량을 조정하고 관리 없이도 성능을 유지합니다. 또한, 0으로 스케일 다운하므로 테이블에 트래픽이 없고 콜드 스타트가 없을 때는 처리량에 대한 비용을 지불하지 않아도 됩니다.

## NoSQL

NoSQL 데이터베이스인 DynamoDB는 기존의 관계형 데이터베이스보다 향상된 성능, 확장성, 관리성 및 유연성을 제공하도록 특별히 구축되었습니다. 다양한 사용 사례를 지원하기 위해 DynamoDB는 키-값 및 문서 데이터 모델을 모두 지원합니다.

관계형 데이터베이스와 달리 DynamoDB는 JOIN 연산자를 지원하지 않습니다. 쿼리에 응답하는 데 필요한 처리 능력과 데이터베이스 왕복 횟수를 줄이려면 데이터 모델을 비정규화하는 것이 좋습니다. NoSQL 데이터베이스인 DynamoDB는 강력한 읽기 일관성과 ACID 트랜잭션을 제공하여 엔터프라이즈급 애플리케이션을 빌드합니다.

## 완전관리형

완전관리형 데이터베이스 서비스인 DynamoDB는 차별화되지 않은 과중한 데이터베이스 관리 작업을 처리하므로, 사용자는 고객을 위한 가치 창출에 집중할 수 있습니다. 설정, 구성, 유지 관리, 고가용성, 하드웨어 프로비저닝, 보안, 백업, 모니터링 등을 처리합니다. 이렇게 하면 DynamoDB 테이블을 생성할 때 프로덕션 워크로드에 즉시 사용할 수 있습니다. DynamoDB는 업그레이드가 필요하지 않고 가동 중지가 발생하지 않으면서 가용성, 신뢰성, 성능, 보안 및 기능을 지속적으로 개선합니다.

## 규모를 따지지 않는 한 자릿수 밀리초의 성능

DynamoDB는 모든 규모에서 10밀리초 미만의 성능을 제공하여 관계형 데이터베이스의 성능과 확장성을 개선하기 위해 특별히 개발되었습니다. 이러한 규모와 성능을 달성하기 위해 DynamoDB는 고성능 워크로드에 최적화되어 있으며 효율적인 데이터베이스 사용을 장려하는 API를 제공합니다. JOIN 작

업과 같이 규모에 비해 비효율적이고 성능이 떨어지는 기능은 생략됩니다. DynamoDB는 사용자 수가 100명이든 1억 명이든 상관없이 일관된 10밀리초 미만의 애플리케이션 성능을 제공합니다.

# DynamoDB 사용 사례

모든 규모, 산업, 지역의 고객이 DynamoDB를 통해 소규모로 시작하여 전 세계로 규모를 조정할 수 있는 최신 서버리스 애플리케이션을 빌드합니다. DynamoDB는 운영 오버헤드가 거의 또는 전혀 없이 모든 규모에서 일관된 성능이 필요한 사용 사례에 적합합니다. 다음 목록은 DynamoDB를 사용할 수 있는 몇 가지 사용 사례를 보여줍니다.

- 금융 서비스 애플리케이션 - 실시간 거래 및 라우팅, 대출 관리, 토큰 생성, 거래 원장과 같은 애플리케이션을 빌드하는 금융 서비스 회사를 예로 들어 보겠습니다. DynamoDB 전역 테이블을 사용하면 애플리케이션이 이벤트에 응답하고 선택한 AWS 리전의 트래픽을 빠른 읽기 및 쓰기 성능을 통해 로컬로 처리할 수 있습니다.

  DynamoDB는 가용성 요구 사항이 가장 엄격한 애플리케이션에 적합합니다. 스토리지 또는 처리량 증가, 버전 관리 및 라이선싱을 위해 인스턴스 규모를 수동으로 조정해야 하는 운영 부담을 없애줍니다.

  DynamoDB 트랜잭션을 사용하여 단일 요청으로 하나 이상의 테이블에서 원자성, 격리, 내구성(ACID)을 달성할 수 있습니다. (ACID) 트랜잭션은 금융 거래 처리 또는 주문 이행을 포함하는 워크로드에 적합합니다. DynamoDB는 늘어나거나 줄어드는 워크로드를 즉시 수용하기 때문에 거래 시간과 같은 시장 상황에 맞게 데이터베이스 규모를 효율적으로 조정할 수 있습니다.
- 게임 애플리케이션 - 게임 회사는 게임 상태, 플레이어 데이터, 세션 기록, 순위표와 같은 게임 플랫폼의 모든 부분에 DynamoDB를 사용할 수 있습니다. DynamoDB가 선택받은 이유는 서버리스 아키텍처가 제공하는 확장성, 일관된 성능, 손쉬운 운영 때문입니다. DynamoDB는 성공적인 게임을 지원하는 데 필요한 스케일 아웃 아키텍처에 매우 적합합니다. 게임의 처리량 규모를 인/아웃으로 빠르게 조정합니다(콜드 스타트 없이 0으로 규모 조정). 이 확장성은 트래픽이 최고치에 맞춰 스케일 아웃하든, 게임플레이 사용량이 적을 때 스케일 백하든 관계없이 아키텍처의 효율성을 최적화합니다.
- 스트리밍 애플리케이션 - 미디어 및 엔터테인먼트 회사는 DynamoDB를 콘텐츠, 콘텐츠 관리 서비스에 대한 메타데이터 인덱스로 사용하거나 실시간에 가까운 스포츠 통계를 제공하는 데 사용합니다. 또한, DynamoDB를 통해 사용자 감시 목록 및 북마크 서비스를 실행하고 매일 수십억 건의 고객 이벤트를 처리하여 추천을 생성합니다. 이러한 고객은 DynamoDB의 확장성, 성능 및 복원력을 활용할 수 있습니다. DynamoDB는 증가하거나 줄어드는 워크로드 변화에 따라 규모를 조정하여 모든 수준의 수요를 뒷받침할 수 있는 스트리밍 미디어 사용 사례를 지원합니다.

다양한 업계의 고객이 DynamoDB를 사용하는 방법에 대해 자세히 알아보려면 Amazon DynamoDB Customers 및 This is My Architecture를 참조하세요.

# DynamoDB의 기능

## 전역 테이블을 사용한 다중 활성 복제

전역 테이블은 99.999% 가용성으로 선택한 AWS 리전에 걸쳐 데이터의 다중 활성 복제 기능을 제공합니다. 전역 테이블은 자체 복제 솔루션을 구축하고 유지 관리하지 않고도 다중 리전의 다중 활성 데이터베이스를 배포할 수 있는 완전관리형 솔루션을 제공합니다. 전역 테이블에서는 테이블을 사용할 수 있는 AWS 리전을 지정할 수 있습니다. DynamoDB는 진행 중인 데이터 변경 사항을 이러한 모든 테이블에 복제합니다.

전 세계에 분산된 애플리케이션은 선택한 리전의 로컬 데이터에 액세스하여 10밀리초 미만의 읽기 및 쓰기 성능을 달성할 수 있습니다. 전역 테이블은 다중 활성의 성격을 띠므로, 기본 테이블이 필요하지 않습니다. 즉, 리전 간에 애플리케이션을 장애 조치할 때 복잡하거나 지연된 장애 조치나 데이터베이스 가동 중지 시간이 발생하지 않습니다.

## ACID 트랜잭션

DynamoDB는 업무상 중요한 워크로드를 위해 구축되었습니다. 여기에는 복잡한 비즈니스 로직이 필요한 애플리케이션을 위한 (ACID) 트랜잭션 지원이 포함됩니다. DynamoDB는 트랜잭션에 대한 기본적인 서버측 지원을 제공하여 테이블 내 및 테이블 간에 여러 항목을 조정된 양자택일 방식으로 변경하는 개발자 경험을 간소화합니다.

## 이벤트 기반 아키텍처를 위한 변경 데이터 캡처

DynamoDB는 항목 수준 변경 데이터 캡처(CDC) 레코드의 스트리밍을 거의 실시간으로 지원합니다. CDC용 DynamoDB Streams 및 DynamoDB용 Amazon Kinesis Data Streams라는 2가지 스트리밍 모델을 제공합니다. 애플리케이션이 테이블의 항목을 생성, 업데이트 또는 삭제할 때마다 스트림은 모든 항목 수준 변경의 시간 순서 시퀀스를 거의 실시간으로 기록합니다. 따라서 DynamoDB Streams는 이벤트 기반 아키텍처를 사용하는 애플리케이션이 변경 사항을 적용하고 이에 따라 조치를 취하는 데 이상적입니다.

## 보조 인덱스

DynamoDB는 글로벌 및 로컬 보조 인덱스를 모두 생성할 수 있는 옵션을 제공하므로, 대체 키를 사용하여 테이블 데이터를 쿼리할 수 있습니다. 이러한 보조 인덱스를 사용하면 프라이머리 키 이외의 속성을 사용하여 데이터에 액세스할 수 있어 데이터에 최대한 유연하게 액세스할 수 있습니다.

# 서비스 통합

DynamoDB는 여러 AWS 서비스와 광범위하게 통합되어 데이터에서 더 많은 가치를 얻고, 차별화되지 않은 부담스러운 작업을 제거하고, 워크로드를 대규모로 운영할 수 있도록 지원합니다. 몇 가지 예로는 AWS CloudFormation, Amazon CloudWatch, Amazon S3, AWS Identity and Access Management(IAM), AWS Auto Scaling 등이 있습니다. 다음 섹션에서는 DynamoDB를 사용하여 수행할 수 있는 몇 가지 서비스 통합을 설명합니다.

## 서버리스 통합

DynamoDB는 엔드 투 엔드 서버리스 애플리케이션을 빌드하기 위해 여러 서버리스 AWS 서비스와 기본적으로 통합됩니다. 예를 들어, DynamoDB와 AWS Lambda를 통합하여 DynamoDB Streams의 이벤트에 자동으로 응답하는 코드 조각인 트리거를 만들 수 있습니다. 트리거를 사용하면 DynamoDB 테이블의 데이터 수정에 응답하는 이벤트 기반 애플리케이션을 빌드할 수 있습니다. 비용 최적화를 위해 Lambda가 DynamoDB 스트림에서 처리하는 이벤트를 필터링할 수 있습니다.

다음 목록은 DynamoDB를 사용한 서버리스 통합의 몇 가지 예제를 보여줍니다.

- GraphQL API를 생성하기 위한 AWS AppSync
- REST API를 생성하기 위한 Amazon API Gateway
- 서버리스 컴퓨팅용 Lambda
- 변경 데이터 캡처(CDC)용 Amazon Kinesis Data Streams

## Amazon S3로 데이터 가져오기 및 내보내기

DynamoDB를 Amazon S3와 통합하면 분석 및 기계 학습을 위해 Amazon S3 버킷으로 데이터를 쉽게 내보낼 수 있습니다. DynamoDB는 전체 테이블 내보내기 및 증분 내보내기를 지원하여 지정된 기간 사이에 변경, 업데이트 또는 삭제된 데이터를 내보낼 수 있습니다. Amazon S3에서 새 DynamoDB 테이블로 데이터를 가져올 수도 있습니다.

## 제로 ETL 통합

DynamoDB는 Amazon Redshift와의 제로 ETL 통합 및 Amazon DynamoDB와 함께 OpenSearch Ingestion 파이프라인 사용을 지원합니다. 이러한 통합을 통해 DynamoDB 테이블 데이터에 대해 복잡한 분석을 실행하고 고급 검색 기능을 사용할 수 있습니다. 예를 들어, DynamoDB 데이터에 대해 전체 텍스트 및 벡터 검색과 시맨틱 검색을 수행할 수 있습니다. 제로 ETL 통합은 DynamoDB에서 실행되는 프로덕션 워크로드에 영향을 주지 않습니다.

## 캐싱

DynamoDB Accelerator(DAX)는 DynamoDB용으로 구축된 완전관리형 고가용성 캐싱 서비스입니다. DAX는 초당 요청 수가 몇백만 개인 경우에도 몇 밀리초에서 몇 마이크로초까지 최대 10배의 개선된 성능을 제공합니다. DAX는 사용자가 캐시 무효화, 데이터 채우기 또는 클러스터 관리를 신경 쓸 필요 없이 DynamoDB 테이블에 인메모리 가속을 추가하는 데 필요한 모든 번거로운 작업을 수행합니다.

## 보안

DynamoDB는 IAM을 사용하여 DynamoDB 리소스에 대한 액세스를 안전하게 제어할 수 있습니다. IAM을 사용하면 리소스에 액세스할 수 있는 DynamoDB 사용자를 제어하는 권한을 중앙에서 관리할 수 있습니다. IAM을 사용하여 리소스를 사용하도록 인증(로그인) 및 권한 부여(권한 있음)된 대상을 제어합니다. DynamoDB는 IAM을 사용하기 때문에 사용자 이름이나 암호 없이 DynamoDB에 액세스할 수 있습니다. 복잡한 암호 교체 정책을 관리할 필요가 없으므로, 보안 태세가 단순해집니다. 또한, IAM을 사용하면 세분화된 액세스 제어를 통해 속성 수준에서 권한을 부여할 수 있습니다. IAM Access Analyzer 및 퍼블릭 액세스 차단(BPA)을 지원하여 리소스 기반 정책을 정의함으로써 정책 관리를 간소화할 수 있습니다.

기본적으로 DynamoDB는 저장된 모든 고객 데이터를 암호화합니다. 저장 중 암호화는 AWS Key Management Service(AWS KMS)에 저장된 암호화 키를 사용하여 데이터의 보안을 강화합니다. 저장 시 암호화를 사용하면 엄격한 암호화 규정 준수 및 규제 요구 사항이 필요한, 보안에 민감한 애플리케이션을 구축할 수 있습니다. 암호화된 테이블에 액세스하면 DynamoDB가 테이블 데이터를 투명하게 해독합니다. 암호화된 테이블을 사용 또는 관리하기 위해 코드나 애플리케이션을 변경할 필요가 없습니다. DynamoDB는 사용자가 기대하는 한 자릿수 밀리초 지연 시간을 지속적으로 제공하며, 모든 DynamoDB 쿼리는 암호화된 데이터에서 원활하게 작동합니다.

DynamoDB가 AWS 소유 키(기본 암호화 유형)와 AWS 관리형 키 또는 고객 관리형 키 중에서 어떤 것으로 사용자 데이터를 암호화하도록 할지 지정할 수 있습니다. AWS 소유 KMS 키를 사용한 기본 암호화는 추가 비용 없이 사용할 수 있습니다. 클라이언트측 암호화의 경우 AWS 데이터베이스 암호화 SDK를 사용할 수 있습니다.

DynamoDB는 또한 HIPAA, PCI DSS, GDPR을 포함한 여러 규정 준수 표준을 준수하므로, 규제 요구 사항을 충족할 수 있습니다.

# 복원성

기본적으로 DynamoDB는 3개의 가용 영역에 데이터를 자동으로 복제하여 높은 내구성과 99.99% 가용성 SLA를 제공합니다. 나아가 DynamoDB는 비즈니스 연속성 및 재해 복구 목표를 달성하는 데 도움이 되는 추가 기능을 제공합니다.

DynamoDB에는 데이터 복원력과 백업 요구 사항을 지원하는 다음 기능이 포함되어 있습니다.

Features

- 글로벌 테이블
- 연속 백업과 특정 시점으로 복구
- 온디맨드 백업 및 복원

## 글로벌 테이블

DynamoDB 전역 테이블은 99.999%의 가용성 SLA와 다중 리전 복원력을 제공합니다. 이를 통해 복원력이 뛰어난 애플리케이션을 빌드하고 최저 Recovery Time Objective(RTO) 및 Recovery Point Objective(RPO)에 맞게 애플리케이션을 최적화할 수 있습니다. 또한, 전역 테이블은 AWS Fault Injection Service(AWS FIS)와 통합되어 전역 테이블 워크로드에 대한 결함 주입 실험을 수행합니다. 예를 들어, 모든 복제 테이블에 대한 전역 테이블 복제를 일시 중지할 수 있습니다.

## 연속 백업과 특정 시점으로 복구

연속 백업은 초당 세분화를 제공하고 특정 시점으로 복구를 개시할 수 있습니다. 특정 시점으로 복구를 사용하면 최근 35일 중 원하는 최신 시점으로 테이블을 복원할 수 있습니다. 복구 기간을 1~35일 사이의 값으로 설정할 수 있습니다.

연속 백업과 특정 시점으로 복구를 시작하는 경우에는 프로비저닝된 용량을 사용하지 않습니다. 또한, 애플리케이션의 성능이나 가용성에도 영향을 미치지 않습니다.

## 온디맨드 백업 및 복원

온디맨드 백업 및 복원 기능을 사용하면 장기 보존을 위한 테이블의 전체 백업과 규정 준수 요구를 위한 아카이브를 생성할 수 있습니다. 백업은 테이블 성능에 영향을 주지 않으므로, 모든 크기의 테이블을 백업할 수 있습니다. AWS Backup 통합과 함께 AWS Backup를 사용하여 DynamoDB 온디맨드 백업을 자동으로 예약, 복사, 태그 지정하고 수명 주기를 관리할 수 있습니다. AWS Backup를 사용하면 계정 및 리전 전체에 온디맨드 백업을 복사하고 비용 최적화를 위해 오래된 백업을 콜드 스토리지로 전환할 수 있습니다.

# DynamoDB 액세스

AWS Management Console, AWS Command Line Interface, DynamoDB용 NoSQL Workbench 또는 DynamoDB API를 사용하여 DynamoDB로 작업할 수 있습니다.

자세한 내용은 DynamoDB 액세스 섹션을 참조하세요.

# DynamoDB 요금

DynamoDB는 활성화하기로 선택한 모든 선택적 기능과 함께 테이블의 데이터 읽기, 쓰기 및 저장에 대한 요금을 청구합니다. DynamoDB에는 테이블에 읽기 및 쓰기를 처리하기 위한 각각의 청구 옵션이 있는 온디맨드 및 프로비저닝 용량 모드가 있습니다.

DynamoDB는 상시 프리 티어에도 포함되어 25GB의 스토리지를 제공합니다. 또한 상시 프리 티어에는 25개의 프로비저닝된 쓰기 및 25개의 프로비저닝된 읽기 용량 유닛(WCU, RCU)이 포함되어 있어 매월 2억 개의 요청을 처리하기에 충분합니다.

자세한 내용은 Amazon DynamoDB 요금을 참조하세요.

# DynamoDB 시작하기

DynamoDB를 처음 사용할 경우 먼저 다음의 주제를 읽는 것이 좋습니다.

- DynamoDB 시작하기 - DynamoDB 설정, 샘플 테이블 생성, 데이터 업로드 프로세스를 안내합니다. 이 주제에서는 AWS Management Console, AWS CLI, NoSQL Workbench, DynamoDB API를 사용하여 몇 가지 기본 데이터베이스 작업을 수행하는 방법에 대한 정보도 제공합니다.
- DynamoDB 핵심 구성 요소 - DynamoDB의 기본 개념을 설명합니다.
- DynamoDB를 사용한 설계 및 아키텍처 설계 모범 사례 - NoSQL 설계, DynamoDB Well-Architected 렌즈, 테이블 설계 및 기타 여러 DynamoDB 기능에 대한 권장 사항을 제공합니다. 이러한 모범 사례는 DynamoDB를 사용할 때 성능을 극대화하고 처리 비용을 최소화하는 데 도움이 됩니다.

또한, DynamoDB에 익숙해지려면 처음부터 끝까지 완벽한 절차를 제시하는 다음 자습서를 검토하는 것이 좋습니다. 상시 프리 티어 기능을 사용하여 이러한 자습서를 완료할 수 있습니다.

- Amazon DynamoDB를 사용하여 NoSQL 테이블 생성 및 쿼리
- NoSQL 키-값 데이터 스토어를 사용하여 애플리케이션 구축

DynamoDB로 마이그레이션하기 위한 리소스, 도구 및 전략에 대한 자세한 내용은 DynamoDB로 마이그레이션을 참조하세요. 최신 블로그와 백서를 읽으려면 Amazon DynamoDB 리소스를 참조하세요.

# DynamoDB 시작하기

다음 섹션에서 DynamoDB 테이블을 생성 및 관리하고 이에 연결하는 방법을 알아봅니다.

시작하기 전에 Amazon DynamoDB의 기본 개념을 숙지해야 합니다. What is Amazon DynamoDB?에서 간략한 개요를 확인하고 Amazon DynamoDB의 핵심 구성 요소에서 더 자세히 살펴볼 수 있습니다. 그런 다음 사전 조건으로 넘어가세요.

> **Note**
>
> AWS 가입 시 DynamoDB를 시작할 수 있는 AWS 프리 티어를 제공합니다. Amazon DynamoDB에 대한 프리 티어 혜택을 아직 초과하지 않은 경우 이 섹션의 예제를 완료하는 데 비용이 들지 않습니다. 초과한 경우 테이블을 생성한 시점부터 테이블을 삭제할 때까지 표준 DynamoDB 사용 요금이 발생합니다.
> 프리 티어 계정을 신청하고 싶지 않을 경우에는 컴퓨터에 DynamoDB Local(다운로드 가능 버전)을 설정할 수 있습니다. 다운로드 가능 버전을 통해 AWS 계정에 가입하거나 DynamoDB 웹 서비스에 액세스하지 않고도 로컬에서 애플리케이션을 개발하고 테스트할 수 있습니다.

주제


- 신규 사용자를 위한 Amazon DynamoDB 리소스
- DynamoDB 액세스
- 사전 조건
- DynamoDB 설정
- 1단계: DynamoDB에 테이블 생성
- 2단계: DynamoDB 테이블에 데이터 쓰기
- 3단계: DynamoDB 테이블의 데이터 읽기
- 4단계: DynamoDB 테이블의 데이터 업데이트
- 5단계: DynamoDB 테이블의 데이터 쿼리
- 6단계: (선택 사항) DynamoDB 테이블을 삭제하여 리소스 정리
- DynamoDB에 관해 더 알아보기

# 신규 사용자를 위한 Amazon DynamoDB 리소스

처음 사용할 경우 먼저 다음 섹션을 읽고 필요에 따라 참조하는 것이 좋습니다.

- 서비스 하이라이트 및 요금 - 제품 세부 정보 페이지에서는 전반적인 DynamoDB 제품 개요, 일반적인 사용 사례, 서비스 하이라이트 및 요금 정보를 제공합니다.
- DynamoDB 리소스 - DynamoDB 리소스 섹션에는 서비스, 데이터 모델링 개념, 핵심 특성 및 기능을 소개하는 비디오, 튜토리얼, 권장 가이드가 있습니다.
- 시작하기 - DynamoDB 시작하기 섹션에는 DynamoDB 설정, 샘플 테이블 생성, 데이터 업로드에 대한 정보가 포함되어 있습니다. 또한 AWS Management Console, AWS CLI, NoSQL Workbench, DynamoDB API를 사용하여 기본적인 데이터베이스 작업을 수행하는 방법에 대한 정보도 제공합니다.
- DynamoDB 기초 교육 과정 - 테이블 설계, 데이터 유형, 기본적인 작업을 포함하여 DynamoDB의 기본 사항을 가르치는 무료 디지털 교육 과정입니다. 이 교육 과정은 DynamoDB 및 NoSQL 데이터베이스를 처음 접하는 개발자에게 적합합니다.
- DynamoDB Nuggets - 주요 DynamoDB 개념과 기능을 설명하는 짧고 핵심적인 비디오 튜토리얼 모음입니다. 이 짧은 비디오에서는 데이터 모델링, 파티셔닝 전략, 액세스 패턴, 모범 사례와 같은 주제를 이해하기 쉬운 형식으로 다룹니다.
- DynamoDB 코드 샘플 리포지토리 - 다양한 프로그래밍 언어로 실용적인 DynamoDB 코드 예제를 제공합니다.
- 무료 DynamoDB 교육 - AWS에서는 DynamoDB 개념, 기능 및 모범 사례를 다루는 무료 디지털 교육 과정을 제공합니다. 이 DynamoDB 심층 교육 과정 시리즈는 효율적인 데이터 모델을 설계하고, 효과적인 파티션 키를 구현하고, 보조 인덱스를 효과적으로 사용하는 방법을 학습하는 데 도움이 됩니다.
- NoSQL Workbench for DynamoDB – NoSQL Workbench는 데이터 모델링, 데이터 시각화 및 쿼리 개발 기능을 제공하는 통합 시각적 도구입니다. DynamoDB 테이블을 설계, 생성, 쿼리, 관리하는 데 도움이 되므로 DynamoDB의 데이터 모델링 개념을 학습하고 액세스 패턴을 테스트하는 데 특히 유용합니다.
- DynamoDB 설계 패턴 - 다양한 사용 사례의 모범 사례를 보여주는 모범 사례 및 데이터 모델링 예제입니다. 효과적인 솔루션을 구현하기 위한 실용적인 예제와 코드 샘플이 포함되어 있습니다.
- 실습 튜토리얼 - 테이블 생성부터 복잡한 쿼리 구현에 이르기까지 일반적인 DynamoDB 작업을 안내하는 AWS Management Console 내 단계별 튜토리얼입니다.
- DynamoDB로 마이그레이션 - 이 DynamoDB로 마이그레이션에서는 데이터베이스를 DynamoDB로 마이그레이션하기 위한 프로세스, 도구 및 전략에 대한 개요를 제공합니다.
- AWS Well-Architected Lens for DynamoDB - the section called “DynamoDB Well-Architected 렌즈”에서는 DynamoDB를 사용하여 신뢰성 있고 안전하며 효율적이고 비용 효과적인 애플리케이션을

설계하고 운영하기 위한 아키텍처 모범 사례를 제공합니다. 여기에는 Well-Architected Framework의 6가지 원칙에 부합하는 설계 원칙, 아키텍처 패턴 및 운영 지침이 포함됩니다.

## 신규 사용자를 위한 Amazon DynamoDB 추가 모범 사례

이전 섹션을 완료한 후에는 다음 섹션을 읽어 보십시오.

- DynamoDB 처리량 용량

  DynamoDB에 사용할 수 있는 두 가지 처리량 모드를 간략히 살펴보고 애플리케이션에 적절한 용량 모드를 선택하기 위해 고려할 사항을 알아봅니다. 온디맨드 모드는 대부분의 DynamoDB 워크로드에 권장되는 기본 처리량 옵션입니다.
- 모범 사례

  DynamoDB로 작업할 때 성능을 극대화하고 비용을 최소화하도록 문제를 식별하고 해결합니다.

## AWS CLI 리소스

AWS 명령줄 인터페이스(AWS CLI)를 사용할 경우 시작하는 데 도움이 되도록 다음 문서를 사용할 수 있습니다.

- AWS CLI 설명서

  이 섹션에서는 AWS CLI 다운로드, 시스템에서 작동 중인 AWS CLI 가져오기 및 AWS 자격 증명 제공에 대한 정보를 제공합니다.
- AWS CLI DynamoDB 설명서

  이 별도의 문서는 구문 및 예제를 포함한 모든 DynamoDB용 AWS CLI 명령을 다룹니다.

## 프로그래밍 리소스

널리 사용되는 다양한 프로그래밍 언어로 DynamoDB API를 사용하도록 애플리케이션 프로그램을 작성할 수 있습니다. 다음과 같은 몇 가지 리소스가 있습니다.

- Amazon Web Services용 도구

  AWS에서는 DynamoDB에 대한 지원과 더불어 다양한 소프트웨어 개발 키트(SDK)를 제공합니다. Java, .NET, PHP, Ruby 및 기타 언어를 사용하여 DynamoDB에 대해 코딩할 수 있습니다. 이러한

SDK는 DynamoDB에 대한 요청의 형식을 지정하고 응답을 분석하며 재시도 논리 및 오류 처리를 제공함으로써 애플리케이션 개발을 상당히 간소화할 수 있습니다.

- DynamoDB API 참조

  AWS SDK를 사용하지 않으려는 경우 DynamoDB API를 직접 사용하면서 DynamoDB와 상호 작용할 수 있습니다. 이 문서는 구문 및 예제를 포함한 모든 DynamoDB API 작업을 다룹니다. 이 섹션에서 요청 생성과 인증 및 응답 처리에 대한 정보와 문제 해결 팁을 찾을 수 있습니다.

# DynamoDB 액세스

AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 DynamoDB API를 사용하여 Amazon DynamoDB에 액세스할 수 있습니다.

주제

- 콘솔 사용
- AWS CLI 사용
- API 사용
- DynamoDB에 대한 NoSQL 워크벤치 사용
- IP 주소 범위

## 콘솔 사용

Amazon DynamoDB용 AWS Management Console은 https://console.aws.amazon.com/dynamodb/home에서 액세스할 수 있습니다.

DynamoDB 콘솔에서 수행할 수 있는 몇 가지 작업은 다음과 같습니다.

- 테이블 관리: 테이블을 생성, 업데이트 및 삭제합니다. 용량 계산기는 용량 요구 사항을 추정하는 데 도움이 될 수 있습니다.
- 데이터와의 상호 작용: 테이블의 항목을 나열, 추가, 업데이트 및 삭제합니다. TTL(Time To Live) 설정
- 모니터링 및 분석: 대시보드를 보고, 경보를 모니터링 및 설정하고, DynamoDB 테이블에 대한 지표와 알림을 분석합니다.
- 최적화 및 확장: 보조 인덱스, 스트림, 트리거, 예약 용량 및 기타 고급 기능을 관리하여 DynamoDB 사용을 개선합니다.

DynamoDB 콘솔은 DynamoDB 리소스를 관리하기 위한 포괄적인 인터페이스를 제공합니다. 콘솔에 액세스하여 상호 작용하며 자세히 알아보는 것이 좋습니다.

## AWS CLI 사용

AWS Command Line Interface(AWS CLI)를 사용하면 명령줄에서 여러 AWS 서비스를 관리하고 스크립트를 통해 자동화할 수 있습니다. 테이블 생성과 같이 특별 작업을 수행할 때 AWS CLI를 사용할 수 있습니다. 또한 이를 사용하여 Amazon DynamoDB 작업을 유틸리티 스크립트 내에 포함할 수 있습니다.

DynamoDB에서 AWS CLI를 사용하려면 액세스 키 ID와 보안 액세스 키를 얻어야 합니다. 자세한 내용은 프로그래밍 방식 액세스 권한 부여 단원을 참조하십시오.

AWS CLI의 DynamoDB에 사용할 수 있는 모든 명령의 전체 목록을 보려면 AWS CLI 명령 참조를 참조하세요.

### AWS CLI 다운로드 및 구성

AWS CLI는 http://aws.amazon.com/cli에서 사용할 수 있습니다. Windows, macOS 또는 Linux에서 실행됩니다. AWS CLI 다운로드 후 다음 단계에 따라 설치 및 구성합니다.

1. AWS Command Line Interface 사용 설명서로 이동합니다.
2. AWS CLI 설치 및 AWS CLI 구성 지침을 따릅니다.

### DynamoDB와 함께 AWS CLI 사용

명령줄 형식은 DynamoDB 작업 이름과 뒤에 나오는 해당 작업 파라미터로 구성됩니다. AWS CLI는 파라미터 값의 간편 구문과 JSON을 지원합니다.

예를 들어, 다음 명령을 사용하면 이름이 Music인 테이블이 생성됩니다. 파티션 키는 Artist이고 정렬 키는 SongTitle입니다. (읽기 쉽도록 이 섹션에서는 긴 명령이 여러 줄로 나누어져 있습니다.)

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

다음 명령을 통해 테이블에 새 항목이 추가됩니다. 본 예제에서 간편 구문과 JSON이 함께 사용됩니다.

```
aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}}' \
    --return-consumed-capacity TOTAL

aws dynamodb put-item \
    --table-name Music \
    --item '{
        "Artist": {"S": "Acme Band"},
        "SongTitle": {"S": "Happy Day"},
        "AlbumTitle": {"S": "Songs About Life"} }' \
    --return-consumed-capacity TOTAL
```

명령줄에서는 유효한 JSON을 작성하기가 어려울 수 있지만, AWS CLI는 JSON 파일을 읽을 수 있습니다. 아래에서 이름이 key-conditions.json인 파일에 저장되어 있는 JSON 코드 조각을 예로 들어보겠습니다.

```
{
    "Artist": {
        "AttributeValueList": [
            {
                "S": "No One You Know"
            }
        ],
        "ComparisonOperator": "EQ"
    },
    "SongTitle": {
        "AttributeValueList": [
            {
                "S": "Call Me Today"
            }
        ],
        "ComparisonOperator": "EQ"
    }
}
```

이제 AWS CLI를 사용하여 `Query` 요청을 발행할 수 있습니다. 이 예제에서는 `--key-conditions` 파라미터에 key-conditions.json 파일의 콘텐츠가 사용됩니다.

```
aws dynamodb query --table-name Music --key-conditions file://key-conditions.json
```

DynamoDB 로컬과 함께 AWS CLI 사용

AWS CLI는 컴퓨터에서 실행되는 DynamoDB 로컬(다운로드 가능 버전)와 상호 작용할 수도 있습니다. 이렇게 하려면 다음 파라미터를 각 명령에 추가합니다.

`--endpoint-url http://localhost:8000`

다음은 AWS CLI를 사용하여 로컬 데이터베이스에 테이블을 나열하는 예제입니다.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

DynamoDB에서 기본값(8000)이 아닌 다른 포트 번호를 사용하는 경우 그에 따라 `--endpoint-url` 값을 수정합니다.

> **Note**
>
> AWS CLI는 기본 엔드포인트로 DynamoDB 로컬(다운로드 가능 버전)을 사용할 수 없습니다. 따라서 각각의 명령어를 사용하여 `--endpoint-url`을 지정해야 합니다.

## API 사용

AWS Management Console 및 AWS Command Line Interface를 사용하여 대화식 Amazon DynamoDB 작업을 수행할 수 있습니다. 하지만 DynamoDB를 최대한 활용하기 위해 AWS SDK를 사용하여 애플리케이션 코드를 작성할 수 있습니다.

AWS SDK는 Java, 브라우저의 JavaScript, .NET, Node.js, PHP, Python, Ruby, C++, Go, Android, iOS에서 DynamoDB 에 대한 광범위한 지원을 제공합니다.

DynamoDB에서 AWS SDK를 사용하려면 AWS 액세스 키 ID와 보안 액세스 키를 얻어야 합니다. 자세한 내용은 DynamoDB 설정(웹 서비스) 단원을 참조하십시오.

AWS SDK를 이용한 DynamoDB 애플리케이션 프로그래밍에 대한 수준 높은 개요를 알아보려면 DynamoDB 및 AWS SDK를 사용한 프로그래밍 단원을 참조하십시오.

## DynamoDB에 대한 NoSQL 워크벤치 사용

DynamoDB용 NoSQL Workbench를 다운로드하고 사용하여 DynamoDB에 액세스할 수도 있습니다.

Amazon DynamoDB용 NoSQL Workbench는 최신 데이터베이스 개발 및 작업에 사용할 수 있는 플랫폼 간, 클라이언트 측 GUI 애플리케이션입니다. Windows, macOS 및 Linux에서 사용할 수 있습니다. NoSQL Workbench는 DynamoDB 테이블 설계, 생성, 쿼리 및 관리에 도움이 되는 데이터 모델링, 데이터 시각화 및 쿼리 개발 기능을 제공하는 시각적 개발 도구입니다. NoSQL Workbench는 이제 설치 프로세스의 선택적 부분으로 DynamoDB Local을 포함하므로 DynamoDB Local에서 데이터를 더 쉽게 모델링할 수 있습니다. DynamoDB Local 및 그 요구 사항에 대한 자세한 내용은 DynamoDB local 설정(다운로드 가능 버전) 섹션을 참조하세요.

> **Note**
>
> DynamoDB용 NoSQL Workbench는 현재 2단계 인증(2FA)으로 구성된 AWS 로그인을 지원하지 않습니다.

데이터 모델링

DynamoDB용 NoSQL Workbench를 사용하여 새로운 데이터 모델을 구축하거나 애플리케이션 데이터 액세스 패턴을 충족하는 기존 데이터 모델을 기반으로 모델을 설계할 수 있습니다. 그리고 프로세스를 종료할 때 설계된 데이터 모델을 가져오고 내보낼 수도 있습니다. 자세한 내용은 NoSQL Workbench로 데이터 모델 빌드 섹션을 참조하세요.

데이터 시각화

데이터 모델 시각화 도우미에는 애플리케이션 코드를 쓰지 않고 쿼리를 매핑하고 액세스 패턴(패싯)을 시각화할 수 있는 캔버스가 있습니다. 각 패싯은 DynamoDB의 서로 다른 각 액세스 패턴에 해당합니다. 데이터 모델에서 사용할 샘플 데이터를 자동 생성할 수 있습니다. 자세한 내용은 데이터 액세스 패턴 시각화 섹션을 참조하세요.

작업 빌드

NoSQL Workbench는 쿼리 개발 및 테스트용 그래픽 사용자 인터페이스를 제공합니다. 작업 빌더를 사용하여 실시간 데이터 세트를 보고, 탐색하고, 쿼리할 수 있습니다. 그리고 체계적인 작업 빌더를 사용하여 데이터 영역 작업을 빌드하고 수행할 수 있습니다. 프로젝션 식과 조건식을 지원을 지원하며, 샘플 코드를 여러 언어로 생성할 수 있습니다. 자세한 내용은 NoSQL Workbench로 데이터 세트 탐색 및 작업 빌드 단원을 참조하십시오.

## IP 주소 범위

Amazon Web Services(AWS)는 현재 IP 주소 범위를 JSON 형식으로 게시합니다. 현재 범위를 보려면 ip-ranges.json을 다운로드합니다. 자세한 내용은 AWS 일반 참조의 AWS IP 주소 범위를 참조하세요.

DynamoDB 테이블 및 인덱스에 액세스하는 데 사용할 수 있는 IP 주소 범위를 찾으려면 ip-ranges.json 파일에서 `"service": "DYNAMODB"` 문자열을 검색하십시오.

> **Note**
>
> DynamoDB Streams 또는 DAX(DynamoDB Accelerator)에는 IP 주소 범위가 적용되지 않습니다.

## 사전 조건

Amazon DynamoDB 자습서를 시작하기 전에 DynamoDB 액세스에서 DynamoDB에 액세스하는 방법을 알아보세요. 그런 다음 DynamoDB 설정 에서 웹 서비스 또는 로컬에서 다운로드한 버전을 통해 DynamoDB를 설정합니다. 그런 다음 1단계: DynamoDB에 테이블 생성로 진행합니다.

> **Note**
>
> - AWS Management Console을 통해서만 DynamoDB와 상호 작용할 계획이면 AWS 액세스 키가 필요 없습니다. AWS에 가입 단계를 완료한 다음 1단계: DynamoDB에 테이블 생성로 이동합니다.
> - 프리 티어 계정을 신청하고 싶지 않을 경우에는 DynamoDB Local(다운로드 가능 버전)을 설정할 수 있습니다. 그런 다음 1단계: DynamoDB에 테이블 생성으로 이동합니다.
> - Linux와 Windows의 터미널에서 CLI 명령으로 작업할 때 차이점이 있습니다. 다음 가이드에서는 Linux 터미널(macOS 포함)용으로 형식이 지정된 명령과 Windows CMD용으로 형식이 지정된 명령을 제공합니다. 사용 중인 터미널 애플리케이션에 가장 적합한 명령을 선택합니다.

## DynamoDB 설정

Amazon DynamoDB 웹 서비스 외에도 AWS는 컴퓨터에서 실행할 수 있는 다운로드 가능 DynamoDB 버전을 제공합니다. 다운로드 가능 버전은 코드를 개발하고 테스트하는 데 유용합니다. 이를 통해 DynamoDB 웹 서비스에 액세스하지 않고도 로컬에서 애플리케이션을 작성하고 테스트할 수 있습니다.

이 섹션의 항목에서는 DynamoDB(다운로드 가능 버전) 및 DynamoDB 웹 서비스를 설정하는 방법을 설명합니다.

주제


- DynamoDB 설정(웹 서비스)
- DynamoDB local 설정(다운로드 가능 버전)


# DynamoDB 설정(웹 서비스)

Amazon DynamoDB 웹 서비스를 사용하려면

1. AWS에 가입합니다.
2. AWS 액세스 키를 가져옵니다(프로그래밍 방식으로 DynamoDB에 액세스하는 데 사용).

> **Note**
>
> AWS Management Console만을 통해 DynamoDB와 상호 작용하려면 AWS 액세스 키가 필요하지 않으므로 이 단계를 건너 뛰어 콘솔 사용으로 이동할 수 있습니다.

3. 자격 증명을 구성합니다(프로그래밍 방식으로 DynamoDB에 액세스하는 데 사용).

## AWS에 가입

DynamoDB 서비스를 사용하려면 AWS 계정이 있어야 합니다. 계정이 아직 없는 경우에 로그인하려고 하면 계정을 만들도록 요청하는 메시지가 표시됩니다. AWS 서비스에 가입하더라도 사용한 서비스에 대해서만 청구됩니다.

AWS에 가입하려면

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   AWS 계정 루트 사용자에 가입하면 AWS 계정 루트 사용자가 만들어집니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

## 프로그래밍 방식 액세스 권한 부여

프로그래밍 방식이나 AWS Command Line Interface(AWS CLI)를 통해 DynamoDB에 액세스하려면 먼저 프로그래밍 방식 액세스 권한이 있어야 합니다. DynamoDB 콘솔만 사용하려는 경우에는 프로그래밍 방식 액세스 권한이 필요하지 않습니다.

사용자가 AWS Management Console 외부에서 AWS와 상호 작용하려면 프로그래밍 방식의 액세스가 필요합니다. 프로그래밍 방식으로 액세스를 부여하는 방법은 AWS에 액세스하는 사용자 유형에 따라 다릅니다.

사용자에게 프로그래밍 방식 액세스 권한을 부여하려면 다음 옵션 중 하나를 선택합니다.

| 프로그래밍 방식 액세스가 필요한 사용자는 누구인가요? | To | 액세스 권한을 부여하는 사용자 |
| --- | --- | --- |
| 작업 인력 ID<br><br>(IAM Identity Center가 관리하는 사용자) | 임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 AWS API에 대한 프로그래밍 요청에 서명합니다. | 사용하고자 하는 인터페이스에 대한 지침을 따릅니다.<br><br>• AWS CLI에 대해서는 AWS Command Line Interface 사용 설명서에서 AWS IAM Identity Center을 사용하도록 AWS CLI 구성을 참조하세요.<br>• AWS SDK, 도구,AWS API에 대해서는 AWS SDK 및 도구 참조 가이드에서 IAM Identity Center 인증을 참조하세요. |
| IAM | 임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 AWS API에 대한 프로그래밍 요청에 서명합니다. | IAM 사용자 설명서의 AWS 리소스와 함께 임시 자격 증명 사용에 나와 있는 지침을 따르세요. |
| IAM | (권장되지 않음)<br>장기 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 | 사용하고자 하는 인터페이스에 대한 지침을 따릅니다. |

| 프로그래밍 방식 액세스가 필요한 사용자는 누구인가요? | To | 액세스 권한을 부여하는 사용자 |
| --- | --- | --- |
| | AWS API에 대한 프로그래밍 요청에 서명합니다. | • AWS CLI에 대해서는 AWS Command Line Interface 사용 설명서에서 IAM 사용자 자격 증명을 사용한 인증을 참조하세요.<br>• AWS SDK와 도구는 AWS SDK 및 도구 참조 가이드에서 장기 자격 증명을 사용한 인증을 참조하세요.<br>• AWS API는 IAM 사용자 설명서에서 IAM 사용자의 액세스 키 관리를 참조하세요. |

## 보안 인증 정보 구성

프로그래밍 방식이나 AWS CLI를 통해 DynamoDB에 액세스하기 전에 애플리케이션에 권한 부여를 활성화하기 위한 자격 증명을 구성해야 합니다.

이를 달성하는 데는 몇 가지 방법이 있습니다. 예를 들어, 액세스 키 ID 및 보안 액세스 키를 저장하는 자격 증명 파일을 수동으로 생성할 수 있습니다. `aws configure`의 AWS CLI 명령을 사용하여 파일을 자동으로 생성할 수도 있습니다. 또는 환경 변수를 사용할 수 있습니다. 자격 증명 구성에 대한 자세한 내용은 프로그래밍 관련 AWS SDK 개발자 안내서를 참조하세요.

AWS CLI를 설치 및 구성하는 방법은 AWS CLI 사용 섹션을 참조하세요.

## 다른 DynamoDB 서비스와 통합

DynamoDB를 다른 많은 AWS 서비스와 통합할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 다른 AWS 서비스와 함께 DynamoDB 사용
- DynamoDB용 AWS CloudFormation
- DynamoDB에서 AWS Backup 사용
- AWS Identity and Access Management(IAM) 및 DynamoDB
- Amazon DynamoDB에서 AWS Lambda 사용

# DynamoDB local 설정(다운로드 가능 버전)

Amazon DynamoDB의 다운로드 가능 버전을 사용하면 DynamoDB 웹 서비스에 액세스하지 않고도 애플리케이션을 개발하고 테스트할 수 있습니다. 대신, 데이터베이스가 컴퓨터에서 독립형으로 실행됩니다. 프로덕션 환경에서 애플리케이션을 배포할 준비가 되면 코드에서 로컬 엔드포인트를 제거한 다음 DynamoDB 웹 서비스를 가리킵니다.

이 로컬 버전을 통해 처리량, 데이터 스토리지 및 데이터 전송 요금을 절감할 수 있습니다. 또한, 애플리케이션 개발 중에 인터넷 연결이 필요 없습니다.

DynamoDB Local은 다운로드(JRE 필요), Apache Maven 종속 항목 또는 도커 이미지로 사용할 수 있습니다.

대신 Amazon DynamoDB 웹 서비스를 사용하려면 DynamoDB 설정(웹 서비스) 섹션을 참조하세요.

주제

- 컴퓨터에 로컬로 DynamoDB 배포
- DynamoDB local 사용 참고 사항
- DynamoDB Local 릴리스 기록
- DynamoDB Local에서의 텔레메트리

## 컴퓨터에 로컬로 DynamoDB 배포

> **Note**
>
> - DynamoDB 로컬은 v3.x(현재), v2.x(레거시) 및 v1.x(사용되지 않음)의 세 가지 버전으로 제공됩니다.
> - DynamoDB v3.x는 로컬 테스트 및 개발 사용에 권장됩니다.
> - DynamoDB 로컬 V2.x에서 V3.x로 마이그레이션하려면 `com.amazonaws.services.dynamodbv2`에서 `software.amazon.dynamodb`로 가져오기 문을 업데이트하고 Maven 사용자에 대한 Maven 종속성을 업데이트해야 합니다.
> - SDK for Java v1.x를 사용하는 애플리케이션을 SDK for Java 2.x로 마이그레이션하는 경우 AWS SDK for Java 2.x의 단계를 따릅니다.

### DynamoDB 로컬 다운로드

다음 단계에 따라 컴퓨터에서 DynamoDB를 설정하고 실행합니다.

컴퓨터에서 DynamoDB를 설정하려면

1. 다음 중 한 곳에서 무료로 DynamoDB 로컬을 다운로드합니다.

| 다운로드 링크 | 체크섬 |
| --- | --- |
| .tar.gz \| .zip | .tar.gz.sha256 \| .zip.sha256 |

> **Important**
>
> 컴퓨터에서 DynamoDB v2.6.0 이상을 실행하려면 Java 런타임 환경(JRE) 버전 17.x 이상이 필요합니다. 애플리케이션은 이전 JRE 버전에서 실행되지 않습니다.

2. 아카이브를 다운로드한 뒤 콘텐츠의 압축을 풀고 압축 해제된 디렉터리를 원하는 위치로 복사합니다.
3. 컴퓨터에서 DynamoDB를 시작하려면 명령 프롬프트 창을 열고 `DynamoDBLocal.jar`의 압축을 해제한 디렉터리로 이동한 후 다음 명령을 입력합니다.

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
```

> **Note**
>
> Windows PowerShell을 사용하는 경우 파라미터 이름이나 전체 이름 및 다음과 비슷한 값을 묶어야 합니다.
> `java -D"java.library.path=./DynamoDBLocal_lib" -jar DynamoDBLocal.jar`
> 중지하기 전까지 DynamoDB는 수신 요청을 처리합니다. DynamoDB를 중지하려면 명령 프롬프트에서 Ctrl+C를 누릅니다.
> DynamoDB는 기본적으로 8000번 포트를 사용합니다. 8000번 포트를 사용할 수 없는 경우에는 이 명령에서 예외가 발생합니다. `-port`를 포함한 DynamoDB 런타임 옵션의 전체 목록을 보려면 이 명령을 입력합니다.

```
java -Djava.library.path=./DynamoDBLocal_lib -jar
DynamoDBLocal.jar -help
```

4. 프로그래밍 방식이나 AWS Command Line Interface(AWS CLI)를 통해 DynamoDB에 액세스하기 전에 애플리케이션에 권한 부여를 활성화하기 위한 자격 증명을 구성해야 합니다. 다음 예제와 같이 다운로드 가능한 DynamoDB가 작동하려면 자격 증명이 필요합니다.

   ```
   AWS Access Key ID: "fakeMyKeyId"
   AWS Secret Access Key: "fakeSecretAccessKey"
   Default Region Name: "fakeRegion"
   ```

   AWS CLI의 `aws configure` 명령을 사용하여 자격 증명을 설정할 수 있습니다. 자세한 내용은 AWS CLI 사용 섹션을 참조하세요.

5. 애플리케이션 작성을 시작합니다. AWS CLI를 통해 로컬에서 실행되는 DynamoDB에 액세스하려면 `--endpoint-url` 파라미터를 사용합니다. 예를 들어, 다음 명령을 사용하여 DynamoDB 테이블을 나열합니다.

   ```
   aws dynamodb list-tables --endpoint-url http://localhost:8000
   ```

DynamoDB 로컬을 Docker 이미지로 실행

Amazon DynamoDB의 다운로드 가능 버전은 도커 이미지로 제공됩니다. 자세한 내용은 dynamodb-local 섹션을 참조하세요. 현재 DynamoDB 로컬 버전을 확인하려면 다음 명령을 입력합니다.

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -version
```

DynamoDB Local을 AWS Serverless Application Model(AWS SAM)에 빌드된 REST 애플리케이션의 일부로 사용하는 예제는 주문 관리를 위한 SAM DynamoDB 애플리케이션을 참조하세요. 이 샘플 애플리케이션은 테스트를 위해 DynamoDB Local을 사용하는 방법을 보여 줍니다.

DynamoDB Local 컨테이너도 사용하는 다중 컨테이너 애플리케이션을 실행하려면 Docker Compose를 사용하여 DynamoDB Local을 비롯한 애플리케이션의 모든 서비스를 정의하고 실행합니다.

Docker compose를 사용하여 DynamoDB local을 설치하고 실행하는 방법

1. Docker desktop을 다운로드하여 설치합니다.
2. 다음 코드를 파일에 복사하고 `docker-compose.yml`로 저장합니다.

```
services:
 dynamodb-local:
   command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
   image: "amazon/dynamodb-local:latest"
   container_name: dynamodb-local
   ports:
     - "8000:8000"
   volumes:
     - "./docker/dynamodb:/home/dynamodblocal/data"
   working_dir: /home/dynamodblocal
```

애플리케이션과 DynamoDB Local을 별도의 컨테이너에 보관하려면 다음 yaml 파일을 사용합니다.

```
version: '3.8'
services:
 dynamodb-local:
   command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
   image: "amazon/dynamodb-local:latest"
   container_name: dynamodb-local
   ports:
     - "8000:8000"
   volumes:
     - "./docker/dynamodb:/home/dynamodblocal/data"
   working_dir: /home/dynamodblocal
 app-node:
   depends_on:
     - dynamodb-local
   image: amazon/aws-cli
   container_name: app-node
   ports:
    - "8080:8080"
   environment:
     AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
     AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
   command:
     dynamodb describe-limits --endpoint-url http://dynamodb-local:8000 --region
 us-west-2
```

이 docker-compose.yml 스크립트는 app-node 컨테이너와 dynamodb-local 컨테이너를 만듭니다. 이 스크립트는 app-node 컨테이너에서 AWS CLI를 사용하여 dynamodb-local 컨테이너에 연결하고 계정 및 테이블 제한을 설명하는 명령을 실행합니다.

고유한 애플리케이션 이미지와 함께 사용하려면 아래 예제의 image 값을 해당 애플리케이션의 값으로 바꿉니다.

```
version: '3.8'
services:
 dynamodb-local:
   command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
   image: "amazon/dynamodb-local:latest"
   container_name: dynamodb-local
   ports:
     - "8000:8000"
   volumes:
     - "./docker/dynamodb:/home/dynamodblocal/data"
   working_dir: /home/dynamodblocal
 app-node:
   image: location-of-your-dynamodb-demo-app:latest
   container_name: app-node
   ports:
     - "8080:8080"
   depends_on:
     - "dynamodb-local"
   links:
     - "dynamodb-local"
   environment:
     AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
     AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
     REGION: 'eu-west-1'
```

> **Note**
>
> YAML 스크립트를 사용하려면 AWS 액세스 키와 AWS 보안 키를 지정해야 하지만 유효한 AWS 키가 없어도 DynamoDB Local에 액세스할 수 있습니다.

3. 다음 명령줄 명령을 실행합니다.

```
docker-compose up
```

### DynamoDB 로컬을 Apache Maven 종속성으로 실행

> **Note**
>
> SDK for Java v1.x를 사용하는 애플리케이션을 SDK for Java 2.x로 마이그레이션하는 경우 AWS SDK for Java 2.x의 단계를 따릅니다.

다음 단계에 따라 애플리케이션의 Amazon DynamoDB를 종속 항목으로 사용합니다.

DynamoDB를 Apache Maven 리포지토리로 배포하려면

1. Apache Maven을 다운로드하고 설치합니다. 자세한 내용은 Downloading Apache Maven 및 Installing Apache Maven을 참조하세요.
2. 애플리케이션의 POM(Project Object Model) 파일에 DynamoDB Maven 리포지토리를 추가합니다.

```
<!--Dependency:-->
<dependencies>
   <dependency>
      <groupId>software.amazon.dynamodb</groupId>
      <artifactId>DynamoDBLocal</artifactId>
      <version>3.0.0</version>
   </dependency>
</dependencies>
```

Spring Boot 3 및/또는 Spring Framework 6와 함께 사용할 수 있는 예제 템플릿:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>org.example</groupId>
<artifactId>SpringMavenDynamoDB</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
   <spring-boot.version>3.0.1</spring-boot.version>
```

```
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.0.1</version>
    </parent>

<dependencies>
    <dependency>
        <groupId>software.amazon.dynamodb</groupId>
        <artifactId>DynamoDBLocal</artifactId>
        <version>3.0.0</version>
    </dependency>
    <!-- Spring Boot -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Spring Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Spring Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Other Spring dependencies -->
    <!-- Replace the version numbers with the desired version -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>6.0.0</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
```

```
        <artifactId>spring-core</artifactId>
        <version>6.0.0</version>
    </dependency>
    <!-- Add other Spring dependencies as needed -->
    <!-- Add any other dependencies your project requires -->
</dependencies>
</project>
```

> **Note**
>
> Maven 중앙 리포지토리 URL을 사용할 수도 있습니다.

AWS CloudShell에서 DynamoDB 로컬 실행

AWS CloudShell은 브라우저 기반의 사전 인증된 쉘로, AWS Management Console에서 바로 시작할 수 있습니다. AWS CloudShell에서 AWS Management Console로 이동하는 몇 가지 방법이 있습니다. 자세한 내용은 AWS CloudShell 시작하기를 참조하십시오.

AWS Management Console의 어느 곳에서나 AWS CloudShell에서 DynamoDB 로컬을 실행하려면 다음 단계를 따릅니다.

AWS Management Console의 AWS CloudShell에서 DynamoDB 로컬을 실행하려면

1. 콘솔 인터페이스에서 AWS CloudShell을 시작하고, 사용 가능한 AWS 리전을 선택한 다음, Bash, PowerShell 또는 Z 쉘과 같은 선호하는 쉘로 전환합니다.
2. AWS 리전을 선택하려면 리전 선택 메뉴로 이동하여 지원되는 AWS 리전을 선택합니다. (사용 가능한 지역은 강조 표시됩니다.)
3. AWS Management Console에서는 다음 옵션 중 하나를 선택하여 AWS CloudShell을 시작할 수 있습니다.
    a. 탐색 모음에서 AWS CloudShell 아이콘을 선택합니다.
    b. 검색 상자에 CloudShell이라는 단어를 입력한 다음, CloudShell을 선택합니다.
    c. 최근 방문한 위젯에서 CloudShell을 선택합니다.
    d. 콘솔 도구 모음에서 CloudShell을 선택합니다.
4. AWS CloudShell에서 DynamoDB 로컬을 실행하려면 `dynamodb-local` 별칭을 사용합니다. DynamoDB 로컬 설정을 변경하기 위한 추가 명령줄 옵션을 지정할 수 있습니다. 사용할 수 있는 옵션은 the section called “사용 노트” 섹션을 참조하세요.

> **Note**
>
> 백그라운드에서 DynamoDB 로컬을 실행하려면 AWS CloudShell에서 `dynamodb-local &`을 사용하여 DynamoDB 로컬을 실행합니다.

5. AWS CLI를 통해 AWS CloudShell에서 로컬로 실행되는 DynamoDB에 액세스하려면 `--endpoint-url` 파라미터를 사용합니다. 예를 들어, 다음 명령을 사용하여 DynamoDB 테이블을 나열합니다.

   ```
   aws dynamodb list-tables --endpoint-url http://localhost:8000
   ```

JAR 파일 다운로드, Docker 이미지로 실행, Maven 종속성으로 사용 등 DynamoDB 로컬을 설정하고 사용하기 위한 다양한 접근 방식을 보여주는 샘플 프로젝트의 예는 DynamoDB Local Sample Java Project를 참조하세요.

## DynamoDB local 사용 참고 사항

엔드포인트를 제외하고 Amazon DynamoDB의 다운로드 가능 버전으로 실행되는 애플리케이션은 DynamoDB 웹 서비스로도 작동됩니다. 그러나 DynamoDB를 로컬에서 사용하는 경우에는 다음 사항을 숙지해야 합니다.

- `-sharedDb` 옵션을 사용하면 DynamoDB에서 이름이 shared-local-instance.db인 단일 데이터베이스 파일이 생성합니다. DynamoDB에 연결되는 모든 프로그램은 이 파일에 액세스합니다. 이 파일을 삭제하면 파일에 저장된 모든 데이터가 손실됩니다.
- `-sharedDb`를 누락하면 데이터베이스 파일의 이름은 myaccesskeyid_region.db로 설정되고, AWS 액세스 키 ID 및 AWS 리전은 애플리케이션 구성에 표시된 대로 지정됩니다. 이 파일을 삭제하면 파일에 저장된 모든 데이터가 손실됩니다.
- `-inMemory` 옵션을 사용하면 DynamoDB는 데이터베이스 파일을 기록하지 않습니다. 대신 모든 데이터가 메모리에 기록되지만 DynamoDB를 종료할 때 데이터가 저장되지 않습니다.
- `-inMemory` 옵션을 사용하는 경우 `-sharedDb` 옵션도 필요합니다.
- `-optimizeDbBeforeStartup` 옵션을 사용하는 경우 DynamoDB가 데이터베이스 파일을 찾을 수 있도록 `-dbPath` 파라미터도 지정해야 합니다.
- DynamoDB용 AWS SDK의 경우 애플리케이션 구성에서 액세스 키 값과 AWS 리전 값을 지정해야 합니다. `-sharedDb` 또는 `-inMemory` 옵션을 사용하지 않는 한, DynamoDB는 이러한 값을 사용하여 로컬 데이터베이스 파일의 이름을 지정합니다. 이 값들은 로컬에서 실행하기 위해서 유효한 AWS

값일 필요는 없습니다. 그러나 유효한 값을 사용할 경우 나중에 사용 중인 엔드포인트를 변경하면 클라우드에서 코드를 실행할 수 있으므로 유용할 것입니다.

- DynamoDB Local은 `billingModeSummary.`에 대해 항상 null을 반환합니다.
- DynamoDB Local `AWS_ACCESS_KEY_ID`에는 문자(A~Z, a~z)와 숫자(0~9)만 포함될 수 있습니다.
- DynamoDB Local은 시점 복구(PITR)를 지원하지 않습니다.

주제

- 명령줄 옵션
- 로컬 엔드포인트 설정
- 다운로드 가능한 DynamoDB와 DynamoDB 웹 서비스의 차이점

## 명령줄 옵션

DynamoDB 다운로드 가능 버전과 함께 다음 명령줄 옵션을 사용할 수 있습니다.

- `-cors value` - JavaScript용 CORS(cross-origin 리소스 공유) 지원을 활성화합니다. 특정 도메인에 대해 쉼표로 구분된 "허용" 목록을 제공해야 합니다. `-cors`의 기본 설정은 별표(*)이며, 퍼블릭 액세스를 허용합니다.
- `-dbPath value` - DynamoDB가 데이터베이스 파일을 기록할 디렉터리입니다. 이 옵션을 지정하지 않으면 현재 디렉터리에 파일이 기록됩니다. `-dbPath` 및 `-inMemory`를 동시에 지정할 수 없다는 점을 유의하세요.
- `-delayTransientStatuses` - DynamoDB가 특정 작업을 지연시키도록 합니다. DynamoDB(다운로드 가능 버전)는 테이블 및 인덱스의 생성/업데이트/삭제 작업과 같은 일부 태스크를 거의 즉각적으로 수행할 수 있습니다. 그러나 DynamoDB 서비스에서는 이러한 태스크를 수행하는 데 시간이 더 소요됩니다. 이 파라미터를 설정하면 컴퓨터에서 실행되는 DynamoDB에서 DynamoDB 웹 서비스의 동작을 더욱 면밀하게 시뮬레이션할 수 있습니다. (현재 이 파라미터는 생성 또는 삭제 상태의 글로벌 보조 인덱스만 지연시킬 수 있습니다.)
- `-help` - 사용 요약 및 옵션을 인쇄합니다.
- `-inMemory` - DynamoDB는 데이터베이스 파일을 사용하는 대신 메모리에서 실행됩니다. DynamoDB를 중지하면 데이터가 저장되지 않습니다. `-dbPath` 및 `-inMemory`를 동시에 지정할 수 없다는 점을 유의하세요.
- `-optimizeDbBeforeStartup` - 컴퓨터에서 DynamoDB를 시작하기 전에 기본 데이터베이스 테이블을 최적화합니다. 이 파라미터를 사용할 때에는 `-dbPath`도 함께 지정해야 합니다.

- `-port value` - DynamoDB가 애플리케이션과 통신하기 위해 사용할 포트 번호입니다. 이 옵션을 지정하지 않으면 기본 포트는 `8000`입니다.

  > **Note**
  >
  > DynamoDB는 기본적으로 8000번 포트를 사용합니다. 8000번 포트를 사용할 수 없는 경우에는 이 명령에서 예외가 발생합니다. `-port` 옵션을 사용하여 다른 포트 번호를 지정할 수 있습니다. `-port`를 포함한 DynamoDB 런타임 옵션의 전체 목록을 보려면 다음 명령어를 입력합니다.
  >
  > ```
  > java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar
  > -help
  > ```

- `-sharedDb` - `-sharedDb`를 지정하는 경우, DynamoDB에서는 자격 증명과 리전마다 별도의 파일을 사용하는 대신 데이터베이스 파일 하나를 사용합니다.
- `-disableTelemetry` - 지정된 경우 DynamoDB Local은 텔레메트리를 전송하지 않습니다.
- `-version` - DynamoDB 로컬 버전을 인쇄합니다.

### 로컬 엔드포인트 설정

기본적으로 AWS SDK 및 도구는 Amazon DynamoDB 웹 서비스에 대한 엔드포인트를 사용합니다. DynamoDB 다운로드 가능 버전에서 SDK 및 도구를 사용하려면 로컬 엔드포인트를 지정해야 합니다.

`http://localhost:8000`

#### AWS Command Line Interface

AWS Command Line Interface(AWS CLI)를 사용하여 DynamoDB 다운로드 가능 버전과 상호 작용할 수 있습니다.

로컬에서 실행되는 DynamoDB에 액세스하려면 `--endpoint-url` 파라미터를 사용합니다. 다음은 AWS CLI를 사용하여 컴퓨터의 DynamoDB에 테이블을 나열하는 예입니다.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

> **Note**
>
> AWS CLI는 기본 엔드포인트로 DynamoDB 다운로드 가능 버전을 사용할 수 없습니다. 따라서 각 AWS CLI 명령을 사용하여 `--endpoint-url`을 지정해야 합니다.

AWS SDK

엔드포인트를 지정하는 방법은 사용하는 프로그래밍 언어 및 AWS SDK에 따라 다릅니다. 다음 섹션에서 그 방법에 대해 설명합니다.

- Java: AWS 리전 및 엔드포인트 설정(DynamoDB Local은 AWS SDK for Java V1 및 V2 지원)
- CodeSamples.Java.RegionAndEndpoint .NET: AWS 리전 및 엔드포인트 설정

다운로드 가능한 DynamoDB와 DynamoDB 웹 서비스의 차이점

DynamoDB 다운로드 가능 버전은 개발 및 테스트용일 뿐입니다. 비교하자면 DynamoDB 웹 서비스는 확장성, 가용성 및 내구성을 갖춘 관리형 서비스로서 프로덕션 환경에 이상적입니다.

DynamoDB 다운로드 가능 버전은 다음 방식에서 웹 서비스와 차이가 있습니다.

- AWS 리전과 별개의 AWS 계정은 클라이언트 수준에서 지원되지 않습니다.
- `CreateTable` 작업에서 필요하더라도 프로비저닝된 처리량 설정이 DynamoDB 다운로드 가능 버전에서 무시됩니다. `CreateTable`의 경우, 실제로 사용되지 않더라도 할당된 읽기 및 쓰기 처리량에 원하는 숫자를 지정할 수 있습니다. 하루에도 언제든지 필요한 만큼 `UpdateTable`을 호출할 수 있습니다. 그러나 프로비저닝된 처리량 값의 변경 내용은 무시됩니다.
- `Scan` 작업이 순차적으로 수행되고 있습니다. 병렬 검색은 지원되지 않습니다. `Scan` 작업의 `Segment` 및 `TotalSegments` 파라미터는 무시됩니다.
- 테이블 데이터의 읽기 및 쓰기 작업 속도는 컴퓨터 속도에 의해서만 제한됩니다. `CreateTable`, `UpdateTable` 및 `DeleteTable` 작업이 즉시 발생하며 테이블은 항상 활성 상태입니다. 테이블 또는 글로벌 보조 인덱스에서 프로비저닝된 처리량 설정만 변경하는 `UpdateTable` 작업이 즉시 발생합니다. `UpdateTable` 작업이 글로벌 보조 인덱스를 생성하거나 삭제하는 경우, 해당 인덱스가 활성 상태가 되기 전에 정상 상태(생성 또는 삭제)로 전환됩니다. 이 때 테이블은 계속 활성 상태로 유지됩니다.
- 읽기 작업은 일정하게 유지됩니다. 그러나 컴퓨터에서 실행되는 DynamoDB 로컬의 속도 때문에 대부분의 읽기 작업은 매우 일정하게 유지됩니다.
- 항목 컬렉션 지표 및 항목 컬렉션 크기는 추적되지 않습니다. 작업 응답에서 항목 컬렉션 지표 대신 null 값이 반환됩니다.
- DynamoDB의 경우, 결과 집합당 반환되는 데이터는 최대 1MB로 제한됩니다. DynamoDB 웹 서비스 및 다운로드 가능 버전 모두에서 이 제한이 적용됩니다. 그러나 인덱스를 쿼리하면 DynamoDB 서비스는 프로젝션된 키와 속성의 크기만 계산합니다. 이와 달리, 다운로드 버전 DynamoDB는 전체 항목의 크기를 계산합니다.

- DynamoDB Streams를 사용할 경우 샤드의 생성 속도가 다를 수 있습니다. DynamoDB 웹 서비스에서 샤드 생성 동작은 부분적으로 테이블 분할 작업의 영향을 받습니다. 로컬에서 DynamoDB를 실행할 경우에는 테이블 분할 기능이 없습니다. 두 경우 모두 샤드는 일시적으로만 존재하므로 애플리케이션이 샤드 동작에 의존해서는 안 됩니다.
- `TransactionConflictExceptions`는 트랜잭션 API에 대해 DynamoDB(다운로드 가능 버전)에서 생성하지 않습니다. Java 모의 프레임워크를 통해 DynamoDB 핸들러에서 `TransactionConflictExceptions`를 시뮬레이션하여 애플리케이션이 충돌하는 트랜잭션에 대응하는 방식을 테스트합니다.
- DynamoDB 웹 서비스에서, 콘솔을 통해 액세스하든 AWS CLI를 통해 액세스하든 테이블 이름은 대소문자를 구분합니다. `Authors`라는 테이블과 `authors`이라는 테이블이 별도의 테이블로 공존할 수 있습니다. 다운로드가 가능한 버전에서 테이블 이름은 대/소문자를 구분하며, 이러한 두 테이블을 생성하려고 시도하면 오류가 발생할 수 있습니다.
- DynamoDB의 다운로드 가능 버전에서는 태깅이 지원되지 않습니다.
- DynamoDB 다운로드 가능 버전은 ExecuteStatement의 Limit 파라미터를 무시합니다.

## DynamoDB Local 릴리스 기록

다음 표에서는 DynamoDB Local의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| 3.0.0 | AWS SDK Java V1에서 V2로 마이그레이션 | • AWS SDK Java V1에서 V2로 마이그레이션<br>• 패키지 구조를 `com.amazonaws.services.dynamodbv2` 에서 `software.amazon.dynamodb.services` 로 업데이트 | 2025년 7월 17일 |

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| | | • AWS SDK Java V1 종속성 제거 | |
| 2.6.0 | DynamoDB API에서 테이블 이름으로 테이블 ARN 지원<br><br>성능 수정 및 보안 업데이트 | • 여러 DynamoDB API에서 테이블 이름으로 테이블 ARN을 사용하는 지원 추가<br>• Mac M3와 같은 고성능 시스템의 `CreateStreamTable` 버그 수정<br>• 취약성 문제를 해결하기 위한 종속성 업그레이드 (CVE-2022-49043, CVE-2024-56732, CVE-2020-29582, CVE-2025-21502, CVE-2024-50602, CVE-2025-24970, CVE-2025-25193) | 2025년 3월 13일 |
| 2.5.4 | Jetty 종속성으로 업그레이드 | • Jetty 12.0.8에서 Jetty 12.0.14로 업그레이드 (CVE-2024-6763, CVE-2024-8184, CVE-2024-47535 해결)<br>(CVE-2024-21634) 완화 | 2024년 12월 12일 |

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| 2.5.3 | Log4j Core에서 Jackson 종속성을 2.17.x로 업그레이드(CVE-2022-1471 해결) | • Log4j Core에서 Jackson 종속성을 2.17.x로 업그레이드하여(CVE-2022-1471 해결) 전이적 종속성인 SnakeYAML 라이브러리의 중요한 보안 취약성 해결 | 2024년 11월 6일 |
| 2.5.2 | 테이블 업데이트 워크플로에 대한 버그 수정 | • 테이블 업데이트 시 GSI를 사용하여 온디맨드 결제 모드에서 프로비저닝된 결제 모드로 테이블을 업데이트하려고 시도하는 워크플로 버그 수정 | 2024년 6월 20일 |
| 2.5.1 | `OndemandThroughPut` 기능에 도입된 버그에 대한 패치 | • `OndemandThroughPut` 와 관련된 몇 가지 버그 수정 | 2024년 6월 5일 |
| 2.5.0 | 온디맨드 테이블 `ReturnValuesOnConditionCheckFailure` , `BatchExecuteStatement` , `ExecuteTransaction` `Request` 의 구성 가능한 최대 처리량 지원 | • 임베디드 모드에 원격 측정 추가<br>• `ConditionalCheckException` 에 대한 SDKv2 변환 수정 | 2024년 5월 28일 |

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| 2.4.0 | `ReturnValuesOnConditionCheckFailure` 지원 – 임베디드 모드 | • 다중 스트림에 대한 작업을 위한 `TrimmedDataAccessException` 의 임베디드 모드 수정<br>• 임베디드 모드에서 SDKv2의 예외 변환 수정 | 2024년 4월 17일 |
| 2.3.0 | Jetty 및 JDK 업그레이드 | • Jetty 12.0.2로 업그레이드<br>• JDK 17로 업그레이드<br>• ANTLR4를 4.10.1로 업그레이드 | 2024년 3월 14일 |
| 2.2.0 | 테이블 삭제 방지 및 `ReturnValuesOnConditionCheckFailure` 파라미터에 대한 지원 추가 | • 테이블 삭제 방지 지원 추가<br>• ReturnValuesOnConditionCheckFailure에 대한 지원 추가<br>• -버전 플래그에 대한 지원 추가 | 2023년 12월 14일 |

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| 2.1.0 | Maven 프로젝트용 SQLLite 네이티브 라이브러리 지원 및 텔레메트리 추가 | • DynamoDB Local에 텔레메트리 추가<br>• 동적인 방식으로 Maven 프로젝트용 SQLLite 네이티브 라이브러리 복사<br>• Maven 종속성에서 io.github.ganadist.sqlite4java 라이브러리 제거<br>• 32.1.1-jre로 GoogleGuava 업그레이드 | 2023년 10월 23일 |
| 2.0.0 | javax에서 jakarta 네임스페이스로 마이그레이션 및 JDK11 지원 | • javax에서 jakarta 네임스페이스로 마이그레이션 및 JDK11 지원<br>• 서버 시작 시 잘못된 액세스 및 비밀 키 처리 문제 수정<br>• 종속성을 업데이트하여 Maven에서 식별된 취약성 수정 | 2023년 7월 5일 |
| 1.25.1 | Log4j Core에서 Jackson 종속성을 2.17.x로 업그레이드(CVE-2022-1471 해결) | Log4j Core에서 Jackson 종속성을 2.17.x로 업그레이드하여(CVE-2022-1471 해결) 전이적 종속성인 SnakeYAML 라이브러리의 중요한 보안 취약성 해결 | 2024년 11월 6일 |

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| 1.25.0 | 테이블 삭제 방지 및 `ReturnValuesOnConditionCheckFailure` 파라미터에 대한 지원 추가 | • 테이블 삭제 방지 지원 추가<br>• ReturnValuesOnConditionCheckFailure에 대한 지원 추가<br>• -버전 플래그에 대한 지원 추가 | 2023년 12월 18일 |
| 1.24.0 | Maven 프로젝트용 SQLLite 네이티브 라이브러리 지원 및 텔레메트리 추가 | • DynamoDB Local에 텔레메트리 추가<br>• 동적인 방식으로 Maven 프로젝트용 SQLLite 네이티브 라이브러리 복사<br>• Maven 종속성에서 io.github.ganadist.sqlite4java 라이브러리 제거<br>• 32.1.1-jre로 GoogleGuava 업그레이드 | 2023년 10월 23일 |
| 1.23.0 | 서버 시작 시 잘못된 액세스 및 비밀 키 처리 | • 서버 시작 시 잘못된 액세스 및 비밀 키 처리 문제 수정<br>• 종속성을 업데이트하여 Maven에서 식별된 취약성 수정 | 2023년 6월 28일 |

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| 1.22.0 | PartiQL Limit Operation 지원 | • PartiQL IN 절 최적화<br>• Limit Operation 지원<br>• Maven 프로젝트 M1 지원 | 2023년 6월 8일 |
| 1.21.0 | 트랜잭션당 100개 작업 지원 | • 트랜잭션당 작업 수 25개에서 100개로 증가<br>• 도커 이미지 Open JDK를 11로 업그레이드<br>• BatchExecuteStatement에서 항목이 중복될 때 발생하는 예외에 대한 패리티 수정 | 2023년 1월 26일 |
| 1.20.0 | M1 Mac에 대한 지원 추가 | • M1 Mac에 대한 지원 추가<br>• Jetty 종속성을 9.4.48.v20220622로 업그레이드 | 2022년 9월 12일 |
| 1.19.0 | PartiQL 파서 업그레이드 | PartiQL 파서 및 기타 관련 라이브러리 업그레이드 | 2022년 7월 27일 |
| 1.18.0 | log4j-core 및 Jackson-core 업그레이드 | log4j-core를 2.17.1로, Jackson-core 2.10.x를 2.12.0으로 업그레이드 | 2022년 1월 10일 |

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| 1.17.2 | log4j-core 업그레이드 | log4j-core 종속성을 버전 2.16으로 업그레이드 | 2021년 1월 16일 |
| 1.17.1 | log4j-core 업그레이드 | 원격 코드 실행 방지를 위해 제로 데이 익스플로잇을 패치하도록 log4j-core 종속성 업데이트 - Log4Shel | 2021년 1월 10일 |
| 1.17.0 | Javascript 웹 쉘 사용되지 않음 | • AWS SDK 종속성을 AWS SDK for Java 1.12.x로 업데이트<br>• Javascript 웹 쉘 사용되지 않음 | 2021년 1월 8일 |

## DynamoDB Local에서의 텔레메트리

AWS에서는 고객과의 상호 작용을 통해 배운 내용을 기반으로 서비스를 개발 및 출시하고 고객 피드백을 활용하여 제품을 제작합니다. 텔레메트리는 고객 요구 사항을 더 잘 이해하고, 문제를 진단하고, 고객 경험을 개선하는 기능을 제공하는 데 도움이 되는 추가 정보입니다.

DynamoDB Local은 일반 사용 지표, 시스템 및 환경 정보, 오류 등의 텔레메트리를 수집합니다. 수집되는 텔레메트리 유형에 대한 자세한 내용은 수집되는 정보의 유형 섹션을 참조하세요.

DynamoDB Local은 사용자 이름 또는 이메일 주소와 같은 개인 정보를 수집하지 않습니다. 또한 민감한 프로젝트 수준 정보를 추출하지 않습니다.

고객은 텔레메트리 사용 여부를 제어할 수 있으며 언제든지 설정을 변경할 수 있습니다. 텔레메트리가 켜져 있는 경우 DynamoDB Local은 추가 고객 상호 작용 없이 백그라운드에서 텔레메트리 데이터를 전송합니다.

명령줄 옵션을 사용하여 텔레메트리 끄기

`-disableTelemetry` 옵션을 사용하여 DynamoDB Local을 시작할 때 명령줄 옵션을 사용하여 텔레메트리를 끌 수 있습니다. 자세한 내용은 명령줄 옵션 단원을 참조하십시오.

단일 세션에 대한 텔레메트리 끄기

macOS 및 Linux 운영 체제에서는 단일 세션에 대한 텔레메트리를 끌 수 있습니다. 현재 세션의 텔레메트리를 끄려면 다음 명령을 실행하여 환경 변수 DDB_LOCAL_TELEMETRY를 false로 설정합니다. 각 새 터미널 또는 세션에 대해 명령을 반복합니다.

```
export DDB_LOCAL_TELEMETRY=0
```

모든 세션에서 사용자 프로필에 대한 텔레메트리 끄기

운영 체제에서 DynamoDB Local을 실행하는 경우 다음 명령을 실행하여 모든 세션에 대한 텔레메트리를 끕니다.

Linux에서 텔레메트리를 끄려면

1. 실행합니다.

   ```
   echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
   ```

2. 실행합니다.

   ```
   source ~/.profile
   ```

macOS에서 텔레메트리를 끄려면

1. 실행합니다.

   ```
   echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
   ```

2. 실행합니다.

   ```
   source ~/.profile
   ```

Windows에서 텔레메트리를 끄려면

1. 실행합니다.

   ```
   setx DDB_LOCAL_TELEMETRY 0
   ```

2. 실행합니다.

```
refreshenv
```

Maven 프로젝트에 내장된 DynamoDB 로컬을 사용하여 원격 분석을 끕니다.

Maven 프로젝트에 내장된 DynamoDB 로컬을 사용하여 원격 분석을 끌 수 있습니다.

```
boolean disableTelemetry = true;
// AWS SDK v1
 AmazonDynamoDB amazonDynamoDB =
 DynamoDBEmbedded.create(disableTelemetry).amazonDynamoDB();

// AWS SDK v2
DynamoDbClient ddbClientSDKv2Local =
 DynamoDBEmbedded.create(disableTelemetry).dynamoDbClient();
```

수집되는 정보의 유형

- 사용 정보 - 서버 시작/중지, 호출된 API 또는 작업과 같은 일반적인 텔레메트리입니다.
- 시스템 및 환경 정보 - Java 버전, 운영 체제(Windows, Linux 또는 macOS), DynamoDB Local이 실행되는 환경(예: 독립형 JAR, Docker 컨테이너 또는 Maven 종속성), 사용 속성의 해시 값입니다.

자세히 알아보기

DynamoDB Local에서 수집하는 텔레메트리 데이터는 AWS 데이터 프라이버시 정책을 준수합니다. 자세한 내용은 다음 자료를 참조하세요.

- AWS 서비스 약관
- 데이터 프라이버시 FAQ

# 1단계: DynamoDB에 테이블 생성

이 단계에서는 Amazon DynamoDB에 `Music` 테이블을 생성합니다. 이 테이블에는 다음과 같은 세부 정보가 있습니다.

- 파티션 키 - `Artist`
- 정렬 키 - `SongTitle`

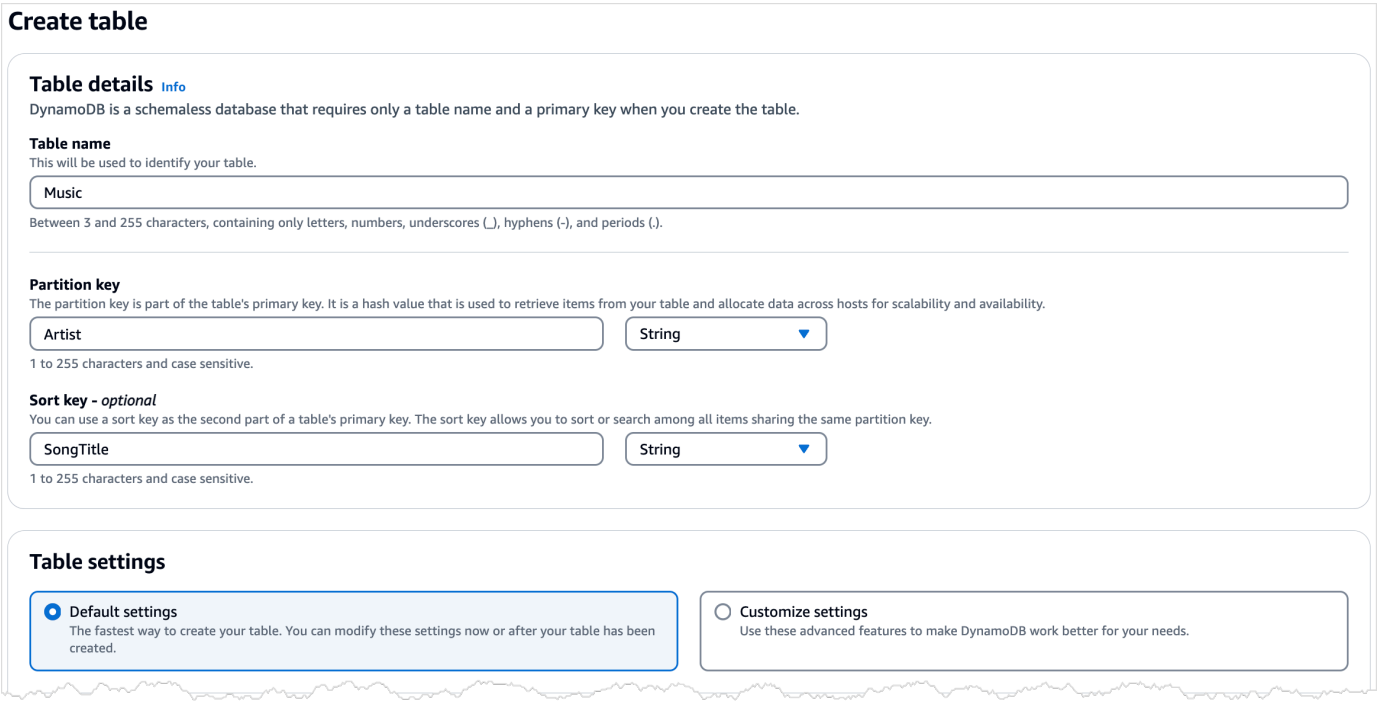
테이블 작업에 대한 자세한 내용은 DynamoDB의 테이블 및 데이터 작업 단원을 참조하세요.

> **Note**
>
> 시작하기 전에 먼저 사전 조건의 단계를 따라야 합니다.

## AWS Management Console

DynamoDB 콘솔을 사용하여 새 `Music` 테이블을 생성하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 테이블을 선택합니다.
3. 테이블 생성을 선택합니다.
4. 다음과 같이 테이블 세부 정보를 입력합니다.
   a. 테이블 이름에 **`Music`**을(를) 입력합니다.
   b. 파티션 키(Partition key)에 **`Artist`**를 입력합니다.
   c. 정렬 키에는 **`SongTitle`**을 입력합니다.
5. 테이블 설정의 경우 기본 설정의 기본 선택을 유지합니다.
6. 테이블 생성을 선택하여 테이블을 생성합니다.



7. 테이블이 ACTIVE 상태가 되면 다음 단계를 수행하여 테이블에서 DynamoDB의 시점 백업를 활성화하는 것이 좋습니다.

   a. 테이블 이름을 선택하여 테이블을 엽니다.
   b. 백업을 선택합니다.
   c. 시점 복구(PITR) 섹션에서 편집을 선택합니다.
   d. 특정 시점으로 복구 설정 편집 페이지에서 특정 시점으로 복구 켜기를 선택합니다.
   e. 변경 사항 저장을 선택합니다.

## AWS CLI

다음 AWS CLI 예제에서는 `create-table`을 사용하여 새 `Music` 테이블을 만듭니다.

Linux

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Windows CMD

```
aws dynamodb create-table ^
    --table-name Music ^
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE ^
    --billing-mode PAY_PER_REQUEST ^
    --table-class STANDARD
```

`create-table`를 사용하면 다음 샘플 결과가 반환됩니다.

```
{
```

```
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "Music",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-03-29T12:11:43.379000-04:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:111122223333:table/Music",
        "TableId": "60abf404-1839-4917-a89b-a8b0ab2a1b87",
        "TableClassSummary": {
            "TableClass": "STANDARD"
        }
    }
}
}
```

`TableStatus` 필드 값은 `CREATING`로 설정됩니다.

DynamoDB에서 `Music` 테이블 생성이 완료되었는지 확인하려면 `describe-table` 명령을 사용합니다.

Linux

```
aws dynamodb describe-table --table-name Music | grep TableStatus
```

Windows CMD

```
aws dynamodb describe-table --table-name Music | findstr TableStatus
```

이 명령은 다음 결과를 반환합니다. DynamoDB에서 테이블 생성이 완료되면 `TableStatus` 필드 값이 `ACTIVE`로 설정됩니다.

```
"TableStatus": "ACTIVE",
```

테이블이 `ACTIVE` 상태가 되면 다음 명령을 실행하여 테이블에서 DynamoDB의 시점 백업를 활성화하는 것이 모범 사례로 간주됩니다.

Linux

```
aws dynamodb update-continuous-backups \
    --table-name Music \
    --point-in-time-recovery-specification \
        PointInTimeRecoveryEnabled=true
```

Windows CMD

```
aws dynamodb update-continuous-backups --table-name Music --point-in-time-recovery-
specification PointInTimeRecoveryEnabled=true
```

이 명령은 다음 결과를 반환합니다.

```
{
    "ContinuousBackupsDescription": {
        "ContinuousBackupsStatus": "ENABLED",
        "PointInTimeRecoveryDescription": {
            "PointInTimeRecoveryStatus": "ENABLED",
            "EarliestRestorableDateTime": "2023-03-29T12:18:19-04:00",
```

```
            "LatestRestorableDateTime": "2023-03-29T12:18:19-04:00"
        }
    }
}
```

> **Note**
>
> 특정 시점 복구를 사용하여 연속 백업을 활성화하면 비용에 영향을 미칩니다. 요금에 대한 자세한 내용은 Amazon DynamoDB 요금을 참조하세요.

## AWS SDK

다음 코드 예제에서는 AWS SDK를 사용하여 DynamoDB 테이블을 생성하는 방법을 보여줍니다.

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
```

```
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);
```

```
                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
{tableName}. {ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateTable을 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      h)
        usage
        return 0
```

```
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
    usage
    return 1
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
#############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
```

```
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 CreateTable을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Create an Amazon DynamoDB table.
/*!
```

```
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;

    Aws::DynamoDB::Model::CreateTableRequest request;

    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);

    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
              request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
```

```
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

테이블이 활성화될 때까지 대기하는 코드입니다.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
```

```
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 CreateTable을 참조하세요.

CLI

AWS CLI

예 1: 태그가 포함된 테이블을 생성하는 방법

다음 `create-table` 예시에서는 지정된 속성과 키 스키마를 사용하여 이름이 `MusicCollection`인 테이블을 생성합니다. 이 테이블은 프로비저닝된 처리량을 사용하며, 기본 AWS 소유 CMK를 사용하여 저장 시 암호화됩니다. 이 명령은 또한 키가 `Owner`이고 값이 `blueTeam`인 태그를 테이블에 적용합니다.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
```

```
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 2: 온디맨드 모드에서 테이블을 생성하는 방법

다음 예시에서는 프로비저닝된 처리량 모드가 아닌 온디맨드 모드를 사용하여 이름이 `MusicCollection`인 테이블을 생성합니다. 이는 예상치 못한 워크로드가 있는 테이블에 유용합니다.

```
aws dynamodb create-table \
    --table-name MusicCollection \
```

```
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
```

```
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 3: 고객 관리형 CMK로 테이블을 생성하고 암호화하는 방법

다음 예시에서는 이름이 MusicCollection인 테이블을 만들고 고객 관리형 CMK를 사용하여 이를 암호화합니다.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
```

```
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 4: 로컬 보조 인덱스가 있는 테이블을 생성하는 방법

다음 `MusicCollection` 예시에서는 지정된 속성과 키 스키마를 사용하여 이름이 `AlbumTitleIndex`인 로컬 보조 인덱스가 있는 이라는 테이블을 생성합니다.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
```

```
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
```

```
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 5: 글로벌 보조 인덱스가 있는 테이블을 생성하는 방법

다음 예시에서는 이름이 GameTitleIndex인 글로벌 보조 인덱스가 있는 GameScores라는 테이블을 생성합니다. 기본 테이블은 파티션 키가 UserId이고 정렬 키가 GameTitle이므로 특정 게임의 개별 사용자 최고 점수를 효율적으로 찾을 수 있는 반면 GSI는 파티션 키가 GameTitle이고 정렬 키가 TopScore이므로 특정 게임의 전체 최고 점수를 빠르게 찾을 수 있습니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
```

```
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
```

```
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 6: 글로벌 보조 인덱스가 있는 테이블 여러 개를 한 번에 생성하는 방법

다음 예시에서는 두 개의 글로벌 보조 인덱스가 있는 GameScores라는 테이블을 생성합니다. GSI 스키마는 명령줄이 아닌 파일을 통해 전달됩니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

gsi.json의 콘텐츠:

```
[
```

```
    {
        "IndexName": "GameTitleIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
```

```
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
```

```
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 7: Streams가 활성화된 테이블을 생성하는 방법

다음 예시에서는 DynamoDB Streams가 활성화된 GameScores라는 테이블을 생성합니다. 각 항목의 새 이미지와 이전 이미지가 모두 스트림에 작성됩니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
```

```
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 8: Keys-Only Stream이 활성화된 테이블을 생성하는 방법

다음 예시에서는 DynamoDB Streams가 활성화된 GameScores라는 테이블을 생성합니다. 수정된 항목의 키 속성만 스트림에 작성됩니다.

```
aws dynamodb create-table \
    --table-name GameScores \
```

```
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
```

```
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB Streams에 대한 변경 데이터 캡처를 참조하세요.

예 9: Standard-Infrequent Access 클래스를 사용하는 테이블을 생성하는 방법

다음 예시에서는 이름이 GameScores인 테이블을 생성하고 Standard-Infrequent Access(DynamoDB Standard-IA) 테이블 클래스를 할당합니다. 이 테이블 클래스는 가장 비용이 많이 드는 스토리지에 최적화되어 있습니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
```

```
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "DeletionProtectionEnabled": false
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 클래스를 참조하세요.

예 10: 삭제 방지가 활성화된 테이블을 생성하는 방법

다음 예시에서는 이름이 GameScores인 테이블을 생성하고 삭제 방지를 활성화합니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
```

```
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
```

```
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 삭제 보호 기능 사용을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 CreateTable을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 CreateTable을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
    public static void main(String[] args) {
        final String usage = """

            Usage:
```

```
                <tableName> <key>

            Where:
                tableName - The Amazon DynamoDB table to create (for example,
Music3).
                key - The key for the Amazon DynamoDB table (for example,
Artist).
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        System.out.println("Creating an Amazon DynamoDB table " + tableName + "
with a simple primary key: " + key);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        String result = createTable(ddb, tableName, key);
        System.out.println("New table is " + result);
        ddb.close();
    }

    public static String createTable(DynamoDbClient ddb, String tableName, String
key) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        CreateTableRequest request = CreateTableRequest.builder()
            .attributeDefinitions(AttributeDefinition.builder()
                .attributeName(key)
                .attributeType(ScalarAttributeType.S)
                .build())
            .keySchema(KeySchemaElement.builder()
                .attributeName(key)
                .keyType(KeyType.HASH)
                .build())
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .tableName(tableName)
            .build();
```

```
        String newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription().tableName();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateTable을 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
```

```
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    BillingMode: "PAY_PER_REQUEST",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 CreateTable을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 CreateTable을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createNewTable(
    tableNameVal: String,
    key: String,
): String? {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef)
            keySchema = listOf(keySchemaVal)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        var tableArn: String
```

```
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 CreateTable를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블을 생성합니다.

```
        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
```

```
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
'WriteCapacityUnits' => 10],
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 CreateTable을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 이 예시는 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만듭니다. 테이블을 구성하는 데 사용되는 스키마는 표시된 대로 또는 -Schema 파라미터를 사용하여 지정한 대로 각 cmdlet에 파이프로 연결할 수 있습니다.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
```

```
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

예 2: 이 예시는 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만듭니다. 로컬 보조 인덱스도 정의됩니다. 로컬 보조 인덱스의 키는 테이블의 프라이머리 해시 키(ForumName)에서 자동으로 설정됩니다. 테이블을 구성하는 데 사용되는 스키마는 표시된 대로 또는 -Schema 파라미터를 사용하여 지정한 대로 각 cmdlet에 파이프로 연결할 수 있습니다.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

예 3: 이 예시는 단일 파이프라인을 사용하여 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만드는 방법을 보여줍니다. Add-DDBKeySchema 및 Add-DDBIndexSchema는 파이프라인 또는 -Schema 파라미터에서 TableSchema 객체가 제공되지 않는 경우 자동으로 새 TableSchema 객체를 만듭니다.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
```

```
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 CreateTable을 참조하세요.

Tools for PowerShell V5

예 1: 이 예시는 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만듭니다. 테이블을 구성하는 데 사용되는 스키마는 표시된 대로 또는 -Schema 파라미터를 사용하여 지정한 대로 각 cmdlet에 파이프로 연결할 수 있습니다.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

예 2: 이 예시는 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만듭니다. 로컬 보조 인덱스도 정의됩니다. 로컬 보조 인덱스의 키는 테이블의 프라이머리 해시 키(ForumName)에서 자동으로 설정됩니다. 테이블을 구성하는 데 사

용되는 스키마는 표시된 대로 또는 -Schema 파라미터를 사용하여 지정한 대로 각 cmdlet에 파이프로 연결할 수 있습니다.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

예 3: 이 예시는 단일 파이프라인을 사용하여 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만드는 방법을 보여줍니다. Add-DDBKeySchema 및 Add-DDBIndexSchema는 파이프라인 또는 -Schema 파라미터에서 TableSchema 객체가 제공되지 않는 경우 자동으로 새 TableSchema 객체를 만듭니다.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
```

```
TableStatus             : CREATING
CreationDateTime        : 10/28/2013 4:39:49 PM
ProvisionedThroughput   : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes          : 0
ItemCount               : 0
LocalSecondaryIndexes   : {LastPostIndex}
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 CreateTable을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

영화 데이터를 저장할 테이블을 생성합니다.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
```

```
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
        else:
            return self.table
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 CreateTable를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        { attribute_name: 'year', key_type: 'HASH' }, # Partition key
        { attribute_name: 'title', key_type: 'RANGE' } # Sort key
```

```
      ],
      attribute_definitions: [
        { attribute_name: 'year', attribute_type: 'N' },
        { attribute_name: 'title', attribute_type: 'S' }
      ],
      billing_mode: 'PAY_PER_REQUEST'
    )
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 CreateTable을 참조하세요.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
```

```
        .attribute_name(&a_name)
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 CreateTable을 참조하세요.

## SAP ABAP

### SDK for SAP ABAP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
```

```
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
      " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
{ lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 CreateTable을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
 "year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
 "title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
 keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
 keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
```

```
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }
```

- API 세부 정보는 Swift용 AWS SDK API 참조의 CreateTable을 참조하세요.

더 많은 DynamoDB 예제는 AWS SDK를 사용한 DynamoDB용 코드 예제 섹션을 참조하세요.

새 테이블을 생성한 후에는 2단계: DynamoDB 테이블에 데이터 쓰기로 이동합니다.

# 2단계: DynamoDB 테이블에 데이터 쓰기

이 단계에서는 1단계: DynamoDB에 테이블 생성에서 생성한 `Music` 테이블에 여러 항목을 삽입합니다.

쓰기 작업에 대한 자세한 내용은 항목 쓰기 단원을 참조하세요.

## AWS Management Console

DynamoDB 콘솔을 사용하여 `Music` 테이블에 데이터를 쓰려면 다음 단계를 따릅니다.

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 테이블을 선택합니다.
3. 테이블 페이지에서 Music 테이블을 선택합니다.
4. 테이블 항목 탐색을 선택합니다.
5. 반환된 항목 섹션에서 항목 생성을 선택합니다.
6. 항목 생성 페이지에서 다음을 수행하여 테이블에 항목을 추가합니다.
    a. 새 속성 추가(Add new attribute)를 선택한 다음 숫자(Number)를 선택합니다.
    b. 속성 이름에 **`Awards`**를 입력합니다.
    c. 이 프로세스를 반복하여 문자열 형식의 **`AlbumTitle`**을 생성합니다.
    d. 항목에 대해 다음과 같은 값을 입력합니다.
        i. [Artist]에 **`No One You Know`**를 입력합니다.

ii. [SongTitle]에 **`Call Me Today`**를 입력합니다.

iii. [AlbumTitle]에 **`Somewhat Famous`**를 입력합니다.

iv. [Awards]에 **`1`**를 입력합니다.

7. 항목 생성(Create Item)을 선택합니다.
8. 이 절차를 반복하여 다음 값으로 다른 항목을 생성합니다.

   a. [Artist]에 **`Acme Band`**를 입력합니다.

   b. [SongTitle]에 **`Happy Day`**를 입력합니다.

   c. [AlbumTitle]에 **`Songs About Life`**를 입력합니다.

   d. [Awards]에 **`10`**를 입력합니다.
9. 이 작업을 한 번 더 수행하여 이전 단계와 Artist는 같지만 다른 속성의 값이 다른 항목을 또 하나 생성합니다.

   a. [Artist]에 **`Acme Band`**를 입력합니다.

   b. [SongTitle]에 **`PartiQL Rocks`**를 입력합니다.

   c. [AlbumTitle]에 **`Another Album Title`**를 입력합니다.

   d. [Awards]에 **`8`**를 입력합니다.

## AWS CLI

다음 AWS CLI 예제에서는 `Music` 테이블에 새로운 항목을 여러 개 생성합니다. 이 작업은 DynamoDB API나 DynamoDB용 SQL 호환 쿼리 언어인 PartiQL을 통해 수행할 수 있습니다.

DynamoDB API

Linux

```
aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

aws dynamodb put-item \
    --table-name Music  \
    --item \
```

```
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Howdy"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "2"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
 "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "PartiQL Rocks"},
 "AlbumTitle": {"S": "Another Album Title"}, "Awards": {"N": "8"}}'
```

### Windows CMD

```
aws dynamodb put-item ^
    --table-name Music  ^
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Call
 Me Today\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\":
 \"1\"}}"

aws dynamodb put-item ^
    --table-name Music  ^
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Howdy
\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\": \"2\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day\"},
 \"AlbumTitle\": {\"S\": \"Songs About Life\"}, \"Awards\": {\"N\": \"10\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"PartiQL Rocks
\"}, \"AlbumTitle\": {\"S\": \"Another Album Title\"}, \"Awards\": {\"N\": \"8\"}}"
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'No One You Know','SongTitle':'Call Me Today',
 'AlbumTitle':'Somewhat Famous', 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'No One You Know','SongTitle':'Howdy',
 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'Acme Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs
 About Life', 'Awards':'10'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'Acme Band','SongTitle':'PartiQL Rocks',
 'AlbumTitle':'Another Album Title', 'Awards':'8'}"
```

Windows CMD

```
aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Call Me Today', 'AlbumTitle':'Somewhat Famous',
 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Howdy', 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs About Life', 'Awards':'10'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'PartiQL Rocks', 'AlbumTitle':'Another Album Title',
 'Awards':'8'}"
```

PartiQL을 사용하여 데이터를 쓰는 방법에 대한 자세한 내용은 PartiQL insert 문을 참조하세요.

DynamoDB에서 지원되는 데이터 형식에 관한 자세한 내용은 데이터 형식을 참조하세요.

DynamoDB 데이터 형식을 JSON으로 표현하는 방법에 관한 자세한 내용은 속성 값을 참조하세요.

## AWS SDK

다음 코드 예제에서는 AWS SDK를 사용하여 DynamoDB 테이블에 항목을 쓰는 방법을 보여줍니다.

### .NET

#### SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };
```

```
            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 PutItem을 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
```

```
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
```

```
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}
```

이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
```

```
}

########################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
########################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

########################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
########################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi
```

```
  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 PutItem을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Put an item in an Amazon DynamoDB table.
/*!
  \sa putItem()
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
                               const Aws::String &artistValue,
                               const Aws::String &albumTitleKey,
                               const Aws::String &albumTitleValue,
                               const Aws::String &awardsKey,
                               const Aws::String &awardsValue,
                               const Aws::String &songTitleKey,
                               const Aws::String &songTitleValue,
```

```
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);

    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

테이블이 활성화될 때까지 대기하는 코드입니다.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
```

```
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 PutItem을 참조하세요.

CLI

AWS CLI

예 1: 테이블에 항목을 추가하는 방법

다음 put-item 예시에서는 MusicCollection 테이블에 새 항목을 추가합니다.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

item.json의 콘텐츠:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

출력:

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 쓰기를 참조하세요.

예 2: 테이블의 항목을 조건부로 덮어쓰는 방법

다음 put-item 예시에서는 기존 항목에 값이 Greatest Hits인 AlbumTitle 속성이 있는 경우에만 MusicCollection 테이블의 기존 항목을 덮어씁니다. 이 명령은 항목의 이전 값을 반환합니다.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

item.json의 콘텐츠:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

names.json의 콘텐츠:

```
{
    "#A": "AlbumTitle"
}
```

values.json의 콘텐츠:

```
{
    ":A": {"S": "Greatest Hits"}
}
```

출력:

```
{
    "Attributes": {
```

```
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
    }
}
```

키가 이미 있는 경우 다음과 같은 출력이 표시됩니다.

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 쓰기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 PutItem을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
```

```
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 PutItem을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDbClient를 사용하여 테이블에 항목 추가

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
 <awards> <awardsval> <Songtitle> <songtitleval>

                Where:
```

```
                    tableName - The Amazon DynamoDB table in which an item is
placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example,
Happy Day).
                **Warning** This program will  place an item that you specify
into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }
```

```
    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {

        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle,
 AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
 AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The
 request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 PutItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 PutCommand를 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 PutItem을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블에 항목을 추가합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

DynamoDB 문서 클라이언트를 사용하여 테이블에 항목을 추가합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 PutItem을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
    albumTitle: String,
    albumTitleValue: String,
```

```
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 PutItem를 참조하세요.

## PHP

### SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
```

```
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 PutItem을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 새 항목을 생성하거나 새 항목으로 기존 항목을 바꿉니다.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 PutItem을 참조하세요.

Tools for PowerShell V5

예 1: 새 항목을 생성하거나 새 항목으로 기존 항목을 바꿉니다.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 PutItem을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
```

```
                "Kevin Costner",
                "Kelly Preston",
                "John C. Reilly"
            ]
        }
    }
"""

def __init__(self, dyn_resource):
    """
    :param dyn_resource: A Boto3 DynamoDB resource.
    """
    self.dyn_resource = dyn_resource
    # The table variable is set during the scenario in the call to
    # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
    self.table = None


def add_movie(self, title, year, plot, rating):
    """
    Adds a movie to the table.

    :param title: The title of the movie.
    :param year: The release year of the movie.
    :param plot: The plot summary of the movie.
    :param rating: The quality rating of the movie.
    """
    try:
        self.table.put_item(
            Item={
                "year": year,
                "title": title,
                "info": {"plot": plot, "rating": Decimal(str(rating))},
            }
        )
    except ClientError as err:
        logger.error(
            "Couldn't add movie %s to table %s. Here's why: %s: %s",
            title,
            self.table.name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 PutItem를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
        'year' => movie[:year],
        'title' => movie[:title],
        'info' => { 'plot' => movie[:plot], 'rating' => movie[:rating] }
      }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 PutItem을 참조하세요.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
```

```
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 PutItem을 참조하세요.

## SAP ABAP

### SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 PutItem을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }
```

```
    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- API 세부 정보는 Swift용 AWS SDK API 참조의 PutItem을 참조하세요.

더 많은 DynamoDB 예제는 AWS SDK를 사용한 DynamoDB용 코드 예제 섹션을 참조하세요.

테이블에 데이터를 쓴 후에 3단계: DynamoDB 테이블의 데이터 읽기로 이동합니다.

# 3단계: DynamoDB 테이블의 데이터 읽기

이 단계에서는 2단계: DynamoDB 테이블에 데이터 쓰기에서 생성한 항목 중 하나를 다시 읽게 됩니다. DynamoDB 콘솔 또는 AWS CLI에서 `Artist` 및 `SongTitle`을 지정하여 `Music` 테이블의 항목을 읽을 수 있습니다.

DynamoDB의 읽기 작업에 대한 자세한 내용은 항목 읽기 단원을 참조하세요.

## AWS Management Console

DynamoDB 콘솔을 사용하여 `Music` 테이블의 데이터를 읽으려면 다음 단계를 따릅니다.

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 테이블을 선택합니다.
3. 테이블 페이지에서 Music 테이블을 선택합니다.
4. 테이블 항목 탐색을 선택합니다.
5. 반환된 항목 섹션에서 테이블에 저장된 항목의 목록을 `Artist` 및 `SongTitle` 기준으로 정렬된 상태로 봅니다. 목록의 첫 번째 항목은 Artist 이름이 Acme Band이고 SongTitle이 PartiQL Rocks인 항목입니다.

## AWS CLI

다음 AWS CLI 예제에서는 `Music`에서 항목을 읽습니다. 이 작업은 DynamoDB API나 DynamoDB용 SQL 호환 쿼리 언어인 PartiQL을 통해 수행할 수 있습니다.

DynamoDB API

> **Note**
>
> DynamoDB의 기본 동작은 최종적으로 일관된 읽기입니다. 아래의 `consistent-read` 파라미터를 사용하여 일관된 읽기를 보여줍니다.

Linux

```
aws dynamodb get-item --consistent-read \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}'
```

Windows CMD

```
aws dynamodb get-item --consistent-read ^
    --table-name Music ^
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}"
```

`get-item`를 사용하면 다음 샘플 결과가 반환됩니다.

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "Awards": {
            "S": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
WHERE Artist='Acme Band' AND SongTitle='Happy Day'"
```

Windows CMD

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band' AND SongTitle='Happy Day'"
```

PartiQL `Select` 문을 사용하면 다음 샘플 결과가 반환됩니다.

```
{
    "Items": [
```

```
        {
            "AlbumTitle": {
                "S": "Songs About Life"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

PartiQL을 사용하여 데이터를 읽는 방법에 대한 자세한 내용은 PartiQL select 문을 참조하세요.

## AWS SDK

다음 코드 예제에서는 AWS SDK를 사용하여 DynamoDB 테이블에서 항목을 읽는 방법을 보여줍니다.

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
```

```
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetItem을 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
```

```
}
query=""
while getopts "n:k:q:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    k) keys="${OPTARG}" ;;
    q) query="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi

if [[ -n "$query" ]]; then
  response=$(aws dynamodb get-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --output text \
    --query "$query")
else
  response=$(
    aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text
```

```
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
```

```
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 GetItem을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Get an item from an Amazon DynamoDB table.
/*!
```

```
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
                               const Aws::String &partitionKey,
                               const Aws::String &partitionValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;

    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cerr << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
```

```
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 GetItem을 참조하세요.

CLI

AWS CLI

예 1: 테이블의 항목을 읽는 방법

다음 `get-item` 예시에서는 `MusicCollection` 테이블에서 항목을 검색합니다. 테이블에는 해시 및 범위 프라이머리 키(`Artist` 및 `SongTitle`)가 있으므로 이 두 속성을 모두 지정해야 합니다. 또한 이 명령은 작업에 사용된 읽기 용량에 대한 정보를 요청합니다.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

`key.json`의 콘텐츠:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

출력:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
```

```
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 읽기를 참조하세요.

예 2: 일관된 읽기를 사용하여 항목을 읽는 방법

다음 예시에서는 강력히 일관되 읽기를 사용하여 MusicCollection 테이블의 항목을 읽습니다.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

key.json의 콘텐츠:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

출력:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
```

```
        "CapacityUnits": 1.0
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 읽기를 참조하세요.

예 3: 항목의 특정 속성을 검색하는 방법

다음 예시에서는 프로젝션 표현식을 사용하여 원하는 항목의 세 가지 속성만 검색합니다.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

names.json의 콘텐츠:

```
{
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

출력:

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 읽기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetItem을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
```

```
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
```

```
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 GetItem을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDbClient를 사용하여 테이블에서 항목을 가져옵니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
```

```
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
 retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
```

```
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
 String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 GetItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 GetCommand를 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 GetItem을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블에서 항목을 가져옵니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

DynamoDB 문서 클라이언트를 사용하여 테이블에서 항목을 가져옵니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};
```

```
docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 GetItem을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun getSpecificItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
```

```
            println(key1.value)
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 GetItem를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";

    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 GetItem을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 파티션 키 SongTitle과 정렬 키 Artist가 있는 DynamoDB 항목을 반환합니다.

```
$key = @{
```

```
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 GetItem을 참조하세요.

Tools for PowerShell V5

예 1: 파티션 키 SongTitle과 정렬 키 Artist가 있는 DynamoDB 항목을 반환합니다.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 GetItem을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 GetItem를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
```

```
  @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
  @table = @dynamo_resource.table(table_name)
end

# Gets movie data from the table for a specific movie.
#
# @param title [String] The title of the movie.
# @param year [Integer] The release year of the movie.
# @return [Hash] The data about the requested movie.
def get_item(title, year)
  @table.get_item(key: { 'year' => year, 'title' => title })
rescue Aws::DynamoDB::Errors::ServiceError => e
  puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
  puts("\t#{e.code}: #{e.message}")
  raise
end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 GetItem을 참조하세요.

## SAP ABAP

### SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
TRY.
    oo_item = lo_dyn->getitem(
      iv_tablename               = iv_table_name
      it_key                     = it_key ).
    DATA(lt_attr) = oo_item->get_item( ).
    DATA(lo_title) = lt_attr[ key = 'title' ]-value.
    DATA(lo_year) = lt_attr[ key = 'year' ]-value.
    DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
    MESSAGE 'Movie name is: ' && lo_title->get_s( )
      && 'Movie year is: ' && lo_year->get_n( )
      && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
```

```
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 GetItem을 참조하세요.

## Swift

### SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
```

```
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }
```

- API 세부 정보는 Swift용 AWS SDK API 참조의 GetItem을 참조하세요.

더 많은 DynamoDB 예제는 AWS SDK를 사용한 DynamoDB용 코드 예제 섹션을 참조하세요.

테이블의 데이터를 업데이트하려면 4단계: DynamoDB 테이블의 데이터 업데이트로 이동합니다.

# 4단계: DynamoDB 테이블의 데이터 업데이트

이 단계에서는 2단계: DynamoDB 테이블에 데이터 쓰기에서 생성한 항목을 업데이트합니다. DynamoDB 콘솔 또는 AWS CLI를 통해 `Artist`, `SongTitle` 및 업데이트된 `AlbumTitle`을 지정하여 `Music` 테이블에서 항목의 `AlbumTitle`을 업데이트할 수 있습니다.

쓰기 작업에 대한 자세한 내용은 항목 쓰기 단원을 참조하세요.

## AWS Management Console

DynamoDB 콘솔을 사용하여 `Music` 테이블의 데이터를 업데이트할 수 있습니다.

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 테이블을 선택합니다.
3. 테이블 목록에서 [Music] 테이블을 선택합니다.
4. 테이블 항목 탐색을 선택합니다.
5. 반환된 항목에서 Acme Band Artist 및 Happy Day SongTitle이 포함된 항목 행에 대해 다음을 수행합니다.

a. Songs About Life라는 AlbumTitle에 커서를 놓습니다.
b. 편집 아이콘을 선택합니다.
c. 문자열 편집 팝업 창에서 **`Songs of Twilight`**를 입력합니다.
d. 저장을 선택합니다.

> **Tip**
>
> 또는 항목을 업데이트하려면 반환된 항목 섹션에서 다음을 수행합니다.
>
> 1. Artist 이름이 Acme Band이고 SongTitle이 Happy Day인 항목 행을 선택합니다.
> 2. 작업 드롭다운 목록에서 항목 편집을 선택합니다.
> 3. AlbumTitle에 **`Songs of Twilight`**를 입력합니다.
> 4. [Save and close]를 선택합니다.

## AWS CLI

다음 AWS CLI 예제에서는 `Music` 테이블의 항목을 업데이트합니다. 이 작업은 DynamoDB API나 DynamoDB용 SQL 호환 쿼리 언어인 PartiQL을 통해 수행할 수 있습니다.

### DynamoDB API

#### Linux

```
aws dynamodb update-item \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}' \
    --update-expression "SET AlbumTitle = :newval" \
    --expression-attribute-values '{":newval":{"S":"Updated Album Title"}}' \
    --return-values ALL_NEW
```

#### Windows CMD

```
aws dynamodb update-item ^
    --table-name Music ^
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}" ^
    --update-expression "SET AlbumTitle = :newval" ^
```

```
    --expression-attribute-values "{\":newval\":{\"S\":\"Updated Album Title\"}}" ^
    --return-values ALL_NEW
```

update-item을 사용하면 return-values ALL_NEW를 지정했으므로 다음 샘플 결과가 반환됩니다.

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Updated Album Title"
        },
        "Awards": {
            "S": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "UPDATE Music  \
    SET AlbumTitle='Updated Album Title'  \
    WHERE Artist='Acme Band' AND SongTitle='Happy Day' \
    RETURNING ALL NEW *"
```

Windows CMD

```
aws dynamodb execute-statement --statement "UPDATE Music SET AlbumTitle='Updated
 Album Title' WHERE Artist='Acme Band' AND SongTitle='Happy Day' RETURNING ALL NEW
 *"
```

Update 문을 사용하면 RETURNING ALL NEW *를 지정했으므로 다음 샘플 결과가 반환됩니다.

```
{
    "Items": [
```

```
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

PartiQL을 사용하여 데이터를 업데이트하는 방법에 대한 자세한 내용은 PartiQL update 문을 참조하세요.

## AWS SDK

다음 코드 예제에서는 AWS SDK를 사용하여 DynamoDB 테이블의 항목을 업데이트하는 방법을 보여줍니다.

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
```

```
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
```

```
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 UpdateItem을 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
#############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
```

```
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
    echo " -e update expression  -- An expression that defines one or more
 attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
  }

  while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
```

```
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi
if [[ -z "$update_expression" ]]; then
  errecho "ERROR: You must provide an update expression with the -e parameter."
  usage
  return 1
fi

if [[ -z "$values" ]]; then
  errecho "ERROR: You must provide a values json file path the -v parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho "    keys:   $keys"
iecho "    update_expression:   $update_expression"
iecho "    values:   $values"

response=$(aws dynamodb update-item \
  --table-name "$table_name" \
  --key file://"$keys" \
  --update-expression "$update_expression" \
  --expression-attribute-values file://"$values")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports update-item operation failed.$response"
```

```
    return 1
  fi

  return 0

}
```

이 예제에 사용된 유틸리티 함수

```
################################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
################################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
```

```
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 UpdateItem을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Update an Amazon DynamoDB table item.
/*!
  \sa updateItem()
  \param tableName: The table name.
```

```
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &attributeKey,
                                  const Aws::String &attributeValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // *** Define UpdateItem request arguments.
    // Define TableName argument.
    Aws::DynamoDB::Model::UpdateItemRequest request;
    request.SetTableName(tableName);

    // Define KeyName argument.
    Aws::DynamoDB::Model::AttributeValue attribValue;
    attribValue.SetS(partitionValue);
    request.AddKey(partitionKey, attribValue);

    // Construct the SET update expression argument.
    Aws::String update_expression("SET #a = :valueA");
    request.SetUpdateExpression(update_expression);

    // Construct attribute name argument.
    Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
    expressionAttributeNames["#a"] = attributeKey;
    request.SetExpressionAttributeNames(expressionAttributeNames);

    // Construct attribute value argument.
```

```
    Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
    attributeUpdatedValue.SetS(attributeValue);
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 expressionAttributeValues;
    expressionAttributeValues[":valueA"] = attributeUpdatedValue;
    request.SetExpressionAttributeValues(expressionAttributeValues);

    // Update the item.
    const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
 dynamoClient.UpdateItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item was updated" << std::endl;
    } else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

테이블이 활성화될 때까지 대기하는 코드입니다.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
```

```
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 UpdateItem을 참조하세요.

## CLI

### AWS CLI

예 1: 테이블의 항목을 업데이트하는 방법

다음 `update-item` 예제에서는 `MusicCollection` 테이블의 항목을 업데이트합니다. 새 속성(`Year`)을 추가하고 `AlbumTitle` 속성을 수정합니다. 업데이트 후에 표시되는 항목 속성이 모두 응답에 반환됩니다.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
```

```
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

key.json의 콘텐츠:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

expression-attribute-names.json의 콘텐츠:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

expression-attribute-values.json의 콘텐츠:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

출력:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
            "N": "2015"
        },
        "SongTitle": {
            "S": "Happy Day"
```

```
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 쓰기를 참조하세요.

예 2: 항목을 조건부로 업데이트하는 방법

다음 예시에서는 기존 항목에 `Year` 속성이 없는 경우에만 `MusicCollection` 테이블의 항목을 업데이트합니다.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

`key.json`의 콘텐츠:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

`expression-attribute-names.json`의 콘텐츠:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

`expression-attribute-values.json`의 콘텐츠:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

항목에 이미 `Year` 속성이 있는 경우 DynamoDB는 다음 출력을 반환합니다.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 쓰기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 UpdateItem을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
```

```
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
```

```
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 UpdateItem을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDbClient를 사용하여 테이블의 항목 업데이트

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
```

```
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
 example, Awards).
                    updateVal - The value used to update an item (for example,
 14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
```

```
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String name,
            String updateVal) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 UpdateCommand를 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 UpdateItem를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
                ]
            ],
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 UpdateItem을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 파티션 키 SongTitle과 정렬 키 Artist가 있는 DynamoDB 항목에서 장르 속성을 'Rap'으로 설정합니다.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Rap
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 UpdateItem을 참조하세요.

Tools for PowerShell V5

예 1: 파티션 키 SongTitle과 정렬 키 Artist가 있는 DynamoDB 항목에서 장르 속성을 'Rap'으로 설정합니다.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
```

```
$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Rap
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 UpdateItem을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

업데이트 표현식을 사용하여 항목을 업데이트합니다.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
```

```
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
```

```
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

산술 연산을 포함하는 업데이트 표현식을 사용하여 항목을 업데이트합니다.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
 operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
 movie.
        :return: The updated rating.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

특정 조건을 충족하는 경우에만 항목을 업데이트합니다.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
        Removes an actor from a movie, but only when the number of actors is
 greater
        than a specified threshold. If the movie does not list more than the
 threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
```

```
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Attributes"]
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Updates rating and plot data for a movie in the table.
  #
  # @param movie [Hash] The title, year, plot, rating of the movie.
  def update_item(movie)
    response = @table.update_item(
      key: { 'year' => movie[:year], 'title' => movie[:title] },
```

```
      update_expression: 'set info.rating=:r',
      expression_attribute_values: { ':r' => movie[:rating] },
      return_values: 'UPDATED_NEW'
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
#{@table.name}\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.attributes
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 UpdateItem을 참조하세요.

## SAP ABAP

### SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        oo_output = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = it_item_key
          it_attributeupdates = it_attribute_updates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 UpdateItem을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
 nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
 [:]
```

```
            if rating != nil {
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }
```

- API에 대한 세부 정보는 AWS Swift용 SDK API 참조의 UpdateItem을 참조하세요.

더 많은 DynamoDB 예제는 AWS SDK를 사용한 DynamoDB용 코드 예제 섹션을 참조하세요.

`Music` 테이블의 데이터를 쿼리하려면 5단계: DynamoDB 테이블의 데이터 쿼리로 이동합니다.

# 5단계: DynamoDB 테이블의 데이터 쿼리

이 단계에서는 `Artist`을 지정하여 the section called “2단계: 데이터 쓰기”에서 `Music` 테이블에 쓴 데이터를 쿼리합니다. 그러면 파티션 키 `Artist`와 연결된 모든 곡이 표시됩니다.

쿼리 작업에 대한 자세한 내용은 DynamoDB에서 테이블 쿼리 단원을 참조하세요.

## AWS Management Console

DynamoDB 콘솔을 사용하여 `Music` 테이블에서 데이터를 쿼리하려면 다음 단계를 따릅니다.

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 테이블을 선택합니다.
3. 테이블 목록에서 [Music] 테이블을 선택합니다.
4. 테이블 항목 탐색을 선택합니다.
5. 항목 스캔 또는 쿼리에서 쿼리가 선택되어 있는지 확인합니다.
6. 파티션 키에 **`Acme Band`**를 입력한 다음 실행(Run)을 선택합니다.

## AWS CLI

다음 AWS CLI 예제에서는 `Music` 테이블의 항목을 쿼리합니다. 이 작업은 DynamoDB API나 DynamoDB용 SQL 호환 쿼리 언어인 PartiQL을 통해 수행할 수 있습니다.

DynamoDB API

`query`를 사용하고 파티션 키를 제공하여 DynamoDB API를 통해 항목을 쿼리합니다.

Linux

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression "Artist = :name" \
    --expression-attribute-values  '{":name":{"S":"Acme Band"}}'
```

Windows CMD

```
aws dynamodb query ^
    --table-name Music ^
    --key-condition-expression "Artist = :name" ^
    --expression-attribute-values  "{\":name\":{\"S\":\"Acme Band\"}}"
```

query를 사용하면 이 특정 Artist와 연결된 모든 곡이 반환됩니다.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "N": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
            "Awards": {
                "N": "8"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
```

```
}
```

PartiQL for DynamoDB

`Select` 문을 사용하고 파티션 키를 제공하여 PartiQL을 통해 항목을 쿼리합니다.

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band'"
```

Windows CMD

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band'"
```

이와 같이 `Select` 문을 사용하면 이 특정 `Artist`와 연결된 모든 곡이 반환됩니다.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
            "Awards": {
                "S": "8"
            },
            "Artist": {
                "S": "Acme Band"
```

```
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

PartiQL을 사용하여 데이터를 쿼리하는 방법에 대한 자세한 내용은 PartiQL select 문을 참조하세요.

## AWS SDK

다음 코드 예제에서는 AWS SDK를 사용하여 DynamoDB 테이블을 쿼리하는 방법을 보여줍니다.

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
```

```
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
```

```
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 Query를 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
```

```
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
    echo " -a attribute_names -- Path to JSON file containing the attribute
 names."
    echo " -v attribute_values -- Path to JSON file containing the attribute
 values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:k:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) key_condition_expression="${OPTARG}" ;;
      a) attribute_names="${OPTARG}" ;;
      v) attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1
```

```
  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$key_condition_expression" ]]; then
    errecho "ERROR: You must provide a key condition expression with the -k
parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
```

```
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
```

```
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 Query를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
```

```
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
 * specified on the command line, only the specified fields and values are
 * returned.
 */

bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::QueryRequest request;

    request.SetTableName(tableName);

    if (!projectionExpression.empty()) {
        request.SetProjectionExpression(projectionExpression);
    }

    // Set query key condition expression.
    request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

    // Set Expression AttributeValues.
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
    attributeValues.emplace(":valueToMatch", partitionValue);

    request.SetExpressionAttributeValues(attributeValues);

    bool result = true;

    // "exclusiveStartKey" is used for pagination.
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 exclusiveStartKey;
    do {
        if (!exclusiveStartKey.empty()) {
            request.SetExclusiveStartKey(exclusiveStartKey);
            exclusiveStartKey.clear();
        }
        // Perform Query operation.
        const Aws::DynamoDB::Model::QueryOutcome &outcome =
 dynamoClient.Query(request);
```

```
        if (outcome.IsSuccess()) {
            // Reference the retrieved items.
            const Aws::Vector<Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
            if (!items.empty()) {
                std::cout << "Number of items retrieved from Query: " <<
 items.size()
                          << std::endl;
                // Iterate each item and print.
                for (const auto &item: items) {
                    std::cout
                            <<
 "******************************************************"
                            << std::endl;
                    // Output each retrieved field and its value.
                    for (const auto &i: item)
                        std::cout << i.first << ": " << i.second.GetS() <<
 std::endl;
                }
            }
            else {
                std::cout << "No item found in table: " << tableName <<
 std::endl;
            }

            exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Query items: " <<
 outcome.GetError().GetMessage();
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 Query를 참조하세요.

CLI

AWS CLI

예 1: 테이블을 쿼리하는 방법

다음 query 예시에서는 MusicCollection 테이블의 항목을 쿼리합니다. 테이블에는 해시 및 범위 프라이머리 키(Artist 및 SongTitle)가 있지만 이 쿼리는 해시 키 값만 지정합니다. 'No One You Know'라는 아티스트의 노래 제목이 반환됩니다.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

expression-attributes.json의 콘텐츠:

```
{
    ":v1": {"S": "No One You Know"}
}
```

출력:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
```

```
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 쿼리 작업을 참조하세요.

예 2: 강력히 일관된 읽기를 사용하여 테이블을 쿼리하고 인덱스를 내림차순으로 탐색하는 방법

다음 예시에서는 첫 번째 예와 동일한 쿼리를 수행하지만 결과를 역순으로 반환하고 강력히 일관된 읽기를 사용합니다.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

expression-attributes.json의 콘텐츠:

```
{
    ":v1": {"S": "No One You Know"}
}
```

출력:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
```

```
        "CapacityUnits": 1.0
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 쿼리 작업을 참조하세요.

예 3: 특정 결과를 필터링하는 방법

다음 예시에서는 MusicCollection을 쿼리하되 AlbumTitle 속성에 특정 값이 있는 결과를 제외합니다. 항목을 읽은 후에 필터가 적용되므로 ScannedCount 또는 ConsumedCapacity에는 영향을 주지 않는다는 점에 유의하세요.

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-consumed-capacity TOTAL
```

values.json의 콘텐츠:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

names.json의 콘텐츠:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

출력:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
```

```
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 쿼리 작업을 참조하세요.

예 4: 항목 수만 검색하는 방법

다음 예시에서는 쿼리와 일치하는 항목 수를 검색하지만 항목 자체는 검색하지 않습니다.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

expression-attributes.json의 콘텐츠:

```
{
    ":v1": {"S": "No One You Know"}
}
```

출력:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 쿼리 작업을 참조하세요.

예 5: 인덱스를 쿼리하는 방법

다음 예시에서는 로컬 보조 인덱스 AlbumTitleIndex를 쿼리합니다. 쿼리는 로컬 보조 인덱스로 프로젝션된 기본 테이블의 모든 속성을 반환합니다. 로컬 보조 인덱스 또는 글로벌 보조 인덱스를 쿼리할 때는 table-name 파라미터를 사용하여 기본 테이블의 이름도 제공해야 한다는 점에 유의하세요.

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

expression-attributes.json의 콘텐츠:

```
{
    ":v1": {"S": "No One You Know"}
}
```

출력:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
```

```
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
                "CapacityUnits": 0.5
            }
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 쿼리 작업을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 Query를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
```

```
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
```

```
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
```

```
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 Query를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDbClient를 사용하여 테이블을 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
```

```
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];

        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }

    public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
            String partitionAlias) {
        // Set up an alias for the partition key name in case it's a reserved
 word.
        HashMap<String, String> attrNameAlias = new HashMap<String, String>();
        attrNameAlias.put(partitionAlias, partitionKeyName);

        // Set up mapping of the partition name with the value.
        HashMap<String, AttributeValue> attrValues = new HashMap<>();
        attrValues.put(":" + partitionKeyName, AttributeValue.builder()
                .s(partitionKeyVal)
                .build());

        QueryRequest queryReq = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
                .expressionAttributeNames(attrNameAlias)
                .expressionAttributeValues(attrValues)
                .build();

        try {
```

```
            QueryResponse response = ddb.query(queryReq);
            return response.count();

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return -1;
    }
}
```

DynamoDbClient 및 보조 인덱스를 사용하여 테이블을 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
```

```
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
            QueryResponse response = ddb.query(request);
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 QueryCommand를 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 Query를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
```

```
        return $this->dynamoDbClient->query($query);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 Query를 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 지정된 SongTitle 및 Artist와 함께 DynamoDB 항목을 반환하는 쿼리를 간접 호출합니다.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 Query를 참조하세요.

Tools for PowerShell V5

예 1: 지정된 SongTitle 및 Artist와 함께 DynamoDB 항목을 반환하는 쿼리를 간접 호출합니다.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
```

```
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 Query를 참조하세요.

## Python

### SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

키 조건 표현식을 사용하여 항목을 쿼리합니다.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
```

```
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]
```

데이터 하위 집합을 반환하도록 항목을 쿼리하고 프로젝션합니다.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
 titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
```

```
                )
                raise
        else:
            return response["Items"]
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 Query를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
      key_condition_expression: '#yr = :year',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: { ':year' => year }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
```

```
  else
    response.items
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 Query를 참조하세요.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

지정된 연도에 제작된 영화를 찾습니다.

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;

    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 Query를 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 Query을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
 "QueryOutput".
```

```
            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }
```

- API에 대한 세부 정보는 AWS Swift용 SDK API 참조의 Query를 참조하세요.

더 많은 DynamoDB 예제는 AWS SDK를 사용한 DynamoDB용 코드 예제 섹션을 참조하세요.

테이블의 글로벌 보조 인덱스를 생성하려면 6단계: (선택 사항) DynamoDB 테이블을 삭제하여 리소스 정리로 이동합니다.

# 6단계: (선택 사항) DynamoDB 테이블을 삭제하여 리소스 정리

자습서용으로 생성한 Amazon DynamoDB 테이블이 더 이상 필요 없는 경우에는 삭제할 수 있습니다. 이렇게 하면 사용하지 않는 리소스에 요금이 청구되지 않습니다. DynamoDB 콘솔 또는 AWS CLI를 사용하여 1단계: DynamoDB에 테이블 생성에서 생성한 `Music` 테이블을 삭제할 수 있습니다.

DynamoDB의 테이블 작업에 대한 자세한 내용은 DynamoDB의 테이블 및 데이터 작업 단원을 참조하세요.

### AWS Management Console

콘솔을 사용하여 `Music` 테이블을 삭제하려면

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 테이블을 선택합니다.
3. 테이블 목록에서 Music 테이블 옆에 있는 확인란을 선택합니다.
4. 삭제를 선택합니다.

### AWS CLI

다음 AWS CLI 예제에서는 `delete-table`을 사용하여 `Music` 테이블을 삭제합니다.

```
aws dynamodb delete-table --table-name Music
```

### AWS SDK

다음 코드 예제에서는 AWS SDK를 사용하여 DynamoDB 테이블을 삭제하는 방법을 보여줍니다.

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
```

```
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteTable을 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
```

```
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
```

```
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
```

```
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 DeleteTable을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Delete an Amazon DynamoDB table.
/*!
  \sa deleteTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteTable(const Aws::String &tableName,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
```

```
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DeleteTable을 참조하세요.

CLI

AWS CLI

테이블을 삭제하는 방법

다음 `delete-table` 예시에서는 `MusicCollection` 테이블을 삭제합니다.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

출력:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 삭제를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteTable을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
```

```
 return err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DeleteTable을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to delete (for example,
 Music3).
```

```
                **Warning** This program will delete the table that you specify!
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Deleting the Amazon DynamoDB table %s...\n",
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteTable을 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteTable을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
```

```
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteTable을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
```

```
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteTable를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
public function deleteTable(string $TableName)
{
    $this->customWaiter(function () use ($TableName) {
        return $this->dynamoDbClient->deleteTable([
            'TableName' => $TableName,
        ]);
    });
}
```

- API 세부 정보는 AWS SDK for PHP API 참조의 DeleteTable을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 지정된 테이블을 삭제합니다. 작업이 진행되기 전에 확인 메시지가 표시됩니다.

```
Remove-DDBTable -TableName "myTable"
```

예 2: 지정된 테이블을 삭제합니다. 작업이 진행되기 전에 확인 메시지가 표시되지 않습니다.

```
Remove-DDBTable -TableName "myTable" -Force
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 DeleteTable을 참조하세요.

Tools for PowerShell V5

예 1: 지정된 테이블을 삭제합니다. 작업이 진행되기 전에 확인 메시지가 표시됩니다.

```
Remove-DDBTable -TableName "myTable"
```

예 2: 지정된 테이블을 삭제합니다. 작업이 진행되기 전에 확인 메시지가 표시되지 않습니다.

```
Remove-DDBTable -TableName "myTable" -Force
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 DeleteTable을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
```

```
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 DeleteTable를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Deletes the table.
  def delete_table
    @table.delete
    @table = nil
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete table. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 DeleteTable을 참조하세요.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
pub async fn delete_table(client: &Client, table: &str) ->
 Result<DeleteTableOutput, Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
```

```
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DeleteTable을 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        " Wait till the table is actually deleted.
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'Table ' && iv_table_name && ' deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 DeleteTable을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }
```

- API 세부 정보는 AWS SDK for Swift API 참조의 DeleteTable을 참조하십시오.

더 많은 DynamoDB 예제는 AWS SDK를 사용한 DynamoDB용 코드 예제 섹션을 참조하세요.

# DynamoDB에 관해 더 알아보기

Amazon DynamoDB에 대한 자세한 내용은 다음 주제를 참조하세요.

- DynamoDB의 테이블 및 데이터 작업
- DynamoDB의 항목 및 속성 작업
- DynamoDB에서 테이블 쿼리
- DynamoDB에서 글로벌 보조 인덱스 사용
- 트랜잭션 작업
- DynamoDB Accelerator(DAX)를 통한 인 메모리 가속화
- DynamoDB 및 AWS SDK를 사용한 프로그래밍

# Amazon DynamoDB: 작동 방식

다음 섹션에서 Amazon DynamoDB 서비스 구성 요소 및 요소 간 상호 작용 방식을 간단히 설명합니다.

주제


- DynamoDB에 대한 치트 시트
- Amazon DynamoDB의 핵심 구성 요소
- DynamoDB API
- Amazon DynamoDB에서 지원되는 데이터 형식 및 이름 지정 규칙
- DynamoDB 테이블 클래스
- DynamoDB의 파티션 및 데이터 배포
- SQL에서 NoSQL로 전환하는 방법 알아보기
- Amazon DynamoDB 학습 리소스 및 도구


## DynamoDB에 대한 치트 시트

이 치트 시트는 Amazon DynamoDB 및 다양한 AWS SDK를 사용하기 위한 빠른 참조를 제공합니다.

### 초기 설정

1. AWS에 가입합니다.
2. 프로그래밍 방식으로 DynamoDB에 액세스하려면 AWS 액세스 키를 얻습니다.
3. DynamoDB 보안 인증 정보를 구성합니다.

다음 사항도 참조하세요.

- DynamoDB 설정(웹 서비스)
- DynamoDB 시작하기
- 핵심 구성 요소의 기본 개요

### SDK 또는 CLI

선호하는 SDK를 선택하거나 AWS CLI를 설정합니다.

> **Note**
>
> Windows에서 AWS CLI를 사용하는 경우 따옴표 안에 없는 백슬래시(\)는 캐리지 리턴으로 처리됩니다. 또한 다른 따옴표 안에 있는 따옴표와 중괄호는 모두 이스케이프 처리해야 합니다. 예를 보려면 다음 섹션의 "테이블 생성"에서 Windows 탭을 참조하세요.

다음 사항도 참조하세요.

- DynamoDB와 함께 AWS CLI 사용
- DynamoDB 시작하기 - 2단계

## 기본 작업

이 섹션에서는 기본 DynamoDB 태스크를 위한 코드를 제공합니다. 이러한 태스크에 대한 자세한 내용은 DynamoDB 및 AWS SDK 시작하기를 참조하세요.

### 테이블 생성

Default

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Windows

```
aws dynamodb create-table ^
    --table-name Music ^
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE ^
```

```
    --billing-mode PAY_PER_REQUEST ^
    --table-class STANDARD
```

### 테이블에 항목 쓰기

```
aws dynamodb put-item \ --table-name Music \ --item file://item.json
```

### 테이블에서 항목 읽기

```
aws dynamodb get-item \ --table-name Music \ --item file://item.json
```

### 테이블에서 항목 삭제

```
aws dynamodb delete-item --table-name Music --key file://key.json
```

### 테이블 쿼리

```
aws dynamodb query --table-name Music
--key-condition-expression "ArtistName=:Artist and SongName=:Songtitle"
```

### 테이블 삭제

```
aws dynamodb delete-table --table-name Music
```

### 테이블 이름 나열

```
aws dynamodb list-tables
```

## 이름 지정 규칙

- 모든 이름은 UTF-8로 인코딩되어야 하며, 대소문자를 구분합니다.
- 테이블 이름 및 인덱스 이름은 3~255자로 이루어져야 하며, 다음 문자만 포함할 수 있습니다.
  - `a-z`
  - `A-Z`

  - `0-9`
  - `_`(밑줄)
  - `-`(대시)
  - `.`(점)
- 속성 이름은 1자 이상이고 크기가 64KB 미만이어야 합니다.

자세한 내용은 이름 지정 규칙을 참조하세요.

## 서비스 할당량 기본 사항

읽기 및 쓰기 단위

- 읽기 용량 단위(RCU) – 초당 강력히 일관된 읽기 1회 또는 초당 최종 읽기 일관성 2회(크기가 최대 4KB인 항목).
- 쓰기 용량 단위(WCU) – 초당 쓰기 1회(크기가 최대 1KB인 항목)

테이블 제한

- 테이블 크기 – 테이블 크기에는 실질적인 제한이 없습니다. 테이블의 항목 수 또는 바이트 수에는 제약이 없습니다.
- 테이블 개수 – 임의의 AWS 계정에 대해 AWS 리전당 테이블 2,500개의 초기 할당량이 있습니다.
- 쿼리 및 스캔의 페이지 크기 제한 – 페이지당, 쿼리 또는 스캔당 1MB로 제한됩니다. 테이블에서 쿼리 파라미터 또는 스캔 작업을 수행한 결과 데이터가 1MB를 초과하는 경우 DynamoDB는 초기 일치 항목을 반환합니다. 또한 새 요청에서 다음 페이지를 읽는 데 사용할 수 있는 `LastEvaluatedKey` 속성도 반환합니다.

인덱스

- 로컬 보조 인덱스(LSI) - 최대 5개의 로컬 보조 인덱스를 정의할 수 있습니다. LSI는 주로 인덱스가 기본 테이블과 강한 일관성을 가져야 할 때 유용합니다.
- 글로벌 보조 인덱스(GSI) – 기본적으로 테이블당 20개의 글로벌 보조 인덱스 할당량이 있습니다.
- 테이블당 프로젝션된 보조 인덱스 속성 – 테이블의 모든 로컬 및 글로벌 보조 인덱스에 속성을 총 100개까지 프로젝션할 수 있습니다. 사용자 지정 프로젝션 속성에만 이 제한이 적용됩니다.

파티션 키

- 파티션 키 값의 최소 길이는 1바이트입니다. 최대 길이는 2,048바이트입니다.
- 고유 파티션 키 값의 수는 테이블 또는 보조 인덱스에 대해 실제로 제한이 없습니다.
- 정렬 키 값의 최소 길이는 1바이트입니다. 최대 길이는 1,024바이트입니다.
- 일반적으로 파티션 키 값당 고유 정렬 키 값의 수는 실제로 제한이 없습니다. 보조 인덱스를 포함하는 테이블은 예외입니다.

보조 인덱스, 파티션 키 설계 및 정렬 키 설계에 대한 자세한 내용은 모범 사례를 참조하세요.

일반적으로 사용되는 데이터 유형의 제한

- 문자열 – 문자열 길이는 최대 항목 크기 400KB의 제한을 받습니다. 문자열은 UTF-8 이진수 인코딩을 사용하는 유니코드입니다.
- 숫자 – 숫자는 소수점 아래 최대 38자리의 양수, 음수 또는 0이 될 수 있습니다.
- 이진수 – 이진수 길이는 최대 항목 크기 400KB의 제한을 받습니다. 이진 속성을 처리하는 애플리케이션에서는 데이터를 DynamoDB로 보내기 전에 base64로 인코딩해야 합니다.

지원되는 데이터 형식의 목록은 데이터 형식을 참조하세요. 자세한 내용은 Service Quotas를 참조하세요.

### 항목, 속성 및 표현식 파라미터

DynamoDB에서 최대 항목 크기가 400KB를 초과할 수 없습니다. 여기에는 속성 이름 이진 길이(UTF-8 길이)와 속성 값 이진 길이(UTF-8 길이)가 모두 포함됩니다. 속성 이름은 크기 제한에 포함됩니다.

목록, 맵 또는 집합 내 값의 수에는 제한이 없습니다. 단, 값을 포함하는 항목이 400KB 항목 크기 제한을 초과하지 않아야 합니다.

표현식 파라미터의 경우 표현식 문자열의 최대 길이는 4KB입니다.

항목 크기, 속성 및 표현식 파라미터에 대한 자세한 내용은 Service Quotas를 참조하세요.

## 추가 정보

- 보안
- 모니터링 및 로깅
- 스트림 작업
- 백업 및 특정 시점으로 복구

- 다른 AWS 서비스와 통합
- API 참조
- 아키텍처 센터: 데이터베이스 모범 사례
- 자습서 비디오
- DynamoDB 포럼

# Amazon DynamoDB의 핵심 구성 요소

DynamoDB에서 테이블, 항목 및 속성이 작업하는 핵심 구성 요소입니다. 테이블은 항목의 컬렉션이고 각 항목은 속성의 컬렉션입니다. DynamoDB는 프라이머리 키를 사용하여 테이블의 각 항목을 고유하게 식별합니다. DynamoDB Streams를 사용해 DynamoDB 테이블의 데이터 수정 이벤트를 캡처할 수 있습니다.

DynamoDB에는 제한이 있습니다. 자세한 내용은 Amazon DynamoDB의 할당량 섹션을 참조하세요.

다음 비디오에서는 테이블, 항목 및 속성에 대해 소개합니다.

테이블, 항목 및 속성

## 테이블, 항목 및 속성

People

| Primary key<br>Partition key: PersonID | Attributes | | | |
|---|---|---|---|---|
| 101 | LastName | FirstName | Phone | |
| | Smith | Fred | 555-4321 | |
| 102 | LastName | FirstName | Address | |
| | Jones | Mary | {"Street":"123 Main","City":"Anytown","State":"OH","ZIPCode":12345} | |
| 103 | LastName | FirstName | FavoriteColor | Address |
| | Stephens | Howard | Blue | {"Street":"123 Main","City":"London","PostalCode":"ER3 5K8"} |

다음은 기본 DynamoDB 구성 요소입니다.

- 테이블 - 다른 데이터베이스 시스템과 마찬가지로 DynamoDB는 데이터를 테이블에 저장합니다. 테이블은 데이터의 집합입니다. 예를 들어, 친구, 가족 또는 기타 관심 있는 사람에 대한 정보를 저장하는 데 사용할 수 있는 People이라는 예제 테이블을 살펴 봅니다. 또한 Cars 테이블에 사람들이 운전하는 차량에 대한 정보를 저장할 수도 있습니다.
- 항목 - 각 테이블에는 0개 이상의 항목이 있습니다. 항목은 모든 기타 항목 중에서 고유하게 식별할 수 있는 속성들의 집합입니다. People 테이블에서 각 항목은 한 사람을 나타냅니다. Cars 테이블의 경우 각 항목은 차량 한 대를 나타냅니다. DynamoDB의 항목은 여러 가지 면에서 다른 데이터베이스 시스템의 행, 레코드 또는 튜플과 유사합니다. DynamoDB에서는 테이블에 저장할 수 있는 항목의 수에 제한이 없습니다.
- 속성 - 각 항목은 하나 이상의 속성으로 구성됩니다. 속성은 기본적인 데이터 요소로서 더 이상 나뉠 필요가 없는 것입니다. 예를 들어 People 테이블의 항목에는 PersonID, LastName, FirstName 등의 속성이 있습니다. Department 테이블의 경우 항목에 DepartmentID, Name, Manager 등의 속성이 있을 수 있습니다. DynamoDB의 속성은 여러 가지 면에서 다른 데이터베이스 시스템의 필드 또는 열과 유사합니다.

다음 다이어그램은 몇 가지 예제 항목과 속성이 있는 People이라는 테이블을 보여 줍니다.

```
People

{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}

{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}

{
```

```
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
                "Street": "123 Main",
                "City": "London",
                "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

People 테이블에 대해 다음을 알아 두세요.

- 테이블의 각 항목에는 항목을 테이블의 다른 모든 항목과 구별해 주는 고유 식별자인 기본 키가 있습니다. People 테이블에서 기본 키는 한 개의 속성(PersonID)으로 구성됩니다.
- 기본 키를 제외하고, People 테이블에는 스키마가 없습니다. 이는 속성이나 데이터 형식을 미리 정의할 필요가 없음을 의미합니다. 각 항목에는 자체의 고유 속성이 있을 수 있습니다.
- 대부분의 속성은 스칼라인데, 이는 하나의 값만 가질 수 있다는 의미입니다. 문자열 및 숫자가 스칼라의 일반적인 예입니다.
- 일부 항목에는 중첩된 속성(Address)이 있습니다. DynamoDB는 최대 32개 깊이 수준까지 중첩된 속성을 지원합니다.

다음은 음악 파일을 추적하는 데 사용할 수 있는 Music이라는 다른 예제 테이블입니다.

```
Music

{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}

{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
```

```
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}

{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150622",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}

{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 0.99,
    "Genre": "Rock"
}
```

Music 테이블에 대해 다음을 알아 두세요.

- Music의 기본 키는 두 개의 속성(Artist 및 SongTitle)으로 구성되어 있습니다. 테이블의 각 항목이 이러한 두 속성을 가지고 있어야 합니다. Artist와 SongTitle의 조합은 테이블의 각 항목을 다른 모든 항목과 구별해 줍니다.
- 기본 키를 제외하고, Music 테이블에는 스키마가 없습니다. 이는 속성이나 데이터 형식을 미리 정의할 필요가 없음을 의미합니다. 각 항목에는 자체의 고유 속성이 있을 수 있습니다.

- 항목 중 하나에 중첩된 속성(PromotionInfo)이 있는데, 이 속성에 다른 중첩된 속성이 포함되어 있습니다. DynamoDB는 최대 32개 깊이 수준까지 중첩된 속성을 지원합니다.

자세한 내용은 DynamoDB의 테이블 및 데이터 작업 섹션을 참조하세요.

## 프라이머리 키

테이블을 생성할 때는 테이블 이름 외에도 테이블의 기본 키를 지정해야 합니다. 기본 키는 테이블의 각 항목을 나타내는 고유 식별자입니다. 따라서 두 항목이 동일한 키를 가질 수는 없습니다.

DynamoDB는 두 가지의 기본 키를 지원합니다.

- 파티션 키 - 파티션 키로 알려진 하나의 속성으로 구성되는 단순 기본 키

  DynamoDB는 내부 해시 함수에 대한 입력으로 파티션 키 값을 사용합니다. 해시 함수 출력에 따라 항목을 저장할 파티션(DynamoDB 내부의 물리적 스토리지)이 결정됩니다.

  파티션 키로만 구성되어 있는 테이블에서는 어떤 두 개의 테이블 항목도 동일한 파티션 키 값을 가질 수 없습니다.

  테이블, 항목 및 속성에 설명된 People 테이블은 단순 기본 키(PersonID)를 사용하는 테이블의 예입니다. People 테이블의 모든 항목은 해당 항목의 PersonId 값을 제공하여 직접 액세스할 수 있습니다.
- 파티션 키 및 정렬 키 - 복합 기본 키로 지칭되는 이 형식의 키는 두 개의 속성으로 구성됩니다. 첫 번째 속성은 파티션 키이고, 두 번째 속성은 정렬 키입니다.

  DynamoDB는 내부 해시 함수에 대한 입력으로 파티션 키 값을 사용합니다. 해시 함수 출력에 따라 항목을 저장할 파티션(DynamoDB 내부의 물리적 스토리지)이 결정됩니다. 파티션 키 값이 동일한 모든 항목은 정렬 키 값을 기준으로 정렬되어 함께 저장됩니다.

  파티션 키와 정렬 키로 구성되어 있는 테이블에서는 여러 항목이 동일한 파티션 키 값을 가질 수 있습니다. 그러나 이러한 아이템의 정렬 키 값은 달라야 합니다.

  테이블, 항목 및 속성에 설명된 Music 테이블은 복합 기본 키(Artist 및 SongTitle)를 사용하는 테이블의 예입니다. Music 테이블의 모든 항목은 해당 항목의 Artist 및 SongTitle 값을 제공하는 경우 직접 액세스할 수 있습니다.

  복합 기본 키를 사용하면 보다 유연하게 데이터를 쿼리할 수 있습니다. 예를 들어, Artist 값만 제공하는 경우, DynamoDB는 해당 아티스트의 모든 노래를 검색합니다. Artist 값과 함께 SongTitle 값 범위를 입력하여 특정 아티스트의 노래 중 일부만 검색할 수도 있습니다.

> **Note**
>
> 항목의 파티션 키를 해시 속성이라고도 합니다. 해시 속성이라는 용어는 파티션 키 값에 따라 파티션에 데이터 항목을 고르게 분산하는 DynamoDB의 내부 해시 함수를 사용하는 것에서 유래합니다.
> 항목의 정렬 키를 범위 속성이라고도 합니다. 범위 속성이라는 용어는 DynamoDB가 파티션 키가 동일한 항목을 정렬 키 값을 기준으로 정렬된 순서로 물리적으로 서로 가깝게 저장하는 방식에서 유래합니다.

각 기본 키 속성은 스칼라여야 합니다(즉, 단일 값만 가질 수 있음). 기본 키 속성에 허용되는 데이터 형식은 문자열, 숫자 또는 이진수뿐입니다. 다른 키가 아닌 속성에는 이러한 제한이 없습니다.

## 보조 인덱스

테이블에서 하나 이상의 보조 인덱스를 생성할 수 있습니다. 보조 인덱스를 사용하면 기본 키에 대한 쿼리는 물론이고 대체 키를 사용하여 테이블의 데이터도 쿼리할 수 있습니다. DynamoDB에서는 인덱스를 사용하지 않아도 되지만, 인덱스를 사용하면 데이터를 쿼리할 때 애플리케이션에 보다 많은 유연성을 제공합니다. 테이블에서 보조 인덱스를 생성한 후에는 테이블에서 데이터를 읽는 것과 같은 방식으로 인덱스에서 데이터를 읽을 수 있습니다.

DynamoDB는 다음과 같이 두 가지 종류의 인덱스를 지원합니다.

- 글로벌 보조 인덱스 - 파티션 키 및 정렬 키가 테이블과 다를 수 있는 인덱스입니다. 글로벌 보조 인덱스의 프라이머리 키 값은 고유할 필요가 없습니다.
- 로컬 보조 인덱스 - 기본 테이블과 파티션 키는 동일하지만 정렬 키가 다른 인덱스입니다.

DynamoDB에서 글로벌 보조 인덱스(GSI)는 테이블 전체에 걸쳐 있는 인덱스이므로 모든 파티션 키를 쿼리할 수 있습니다. 로컬 보조 인덱스(LSI)는 기본 테이블과 파티션키는 동일하지만 정렬 키는 다른 인덱스입니다.

DynamoDB의 각 테이블에는 글로벌 보조 인덱스 20개(기본 할당량)와 로컬 보조 인덱스 5개의 할당량이 있습니다.

앞에 나온 Music 테이블 예제에서는 Artist(파티션 키)를 기준으로 또는 Artist와 SongTitle(파티션 키와 정렬 키)을 기준으로 데이터 항목을 쿼리할 수 있습니다. Genre 및 AlbumTitle로도 데이터를 쿼리하고 싶다면 어떻게 해야 할까요? 그렇게 하려면 Genre 및 AlbumTitle에 대해 인덱스를 생성한 후 Music 테이블을 쿼리한 방법대로 인덱스를 쿼리하면 됩니다.

다음 다이어그램은 Music 테이블과 GenreAlbumTitle이라는 새 인덱스를 보여 줍니다. 인덱스에서 Genre는 파티션 키이고, AlbumTitle은 정렬 키입니다.

| Music 테이블 | GenreAlbumTitle |
| --- | --- |
| {<br>"Artist": "No One You Know",<br>"SongTitle": "My Dog Spot",<br>"AlbumTitle": "Hey Now",<br>"Price": 1.98,<br>"Genre": "Country",<br>"CriticRating": 8.4<br>} | {<br>"Genre": "Country",<br>"AlbumTitle": "Hey Now",<br>"Artist": "No One You Know",<br>"SongTitle": "My Dog Spot"<br>} |
| {<br>"Artist": "No One You Know",<br>"SongTitle": "Somewhere Down The Road",<br>"AlbumTitle": "Somewhat Famous",<br>"Genre": "Country",<br>"CriticRating": 8.4,<br>"Year": 1984<br>} | {<br>"Genre": "Country",<br>"AlbumTitle": "Somewhat Famous",<br>"Artist": "No One You Know",<br>"SongTitle": "Somewhere Down The Road"<br>} |
| {<br>"Artist": "The Acme Band",<br>"SongTitle": "Still in Love",<br>"AlbumTitle": "The Buck Starts Here",<br>"Price": 2.47,<br>"Genre": "Rock",<br>"PromotionInfo": {<br>"RadioStationsPlaying": {<br>"KHCR",<br>"KQBX",<br>"WTNR", | {<br>"Genre": "Rock",<br>"AlbumTitle": "The Buck Starts Here",<br>"Artist": "The Acme Band",<br>"SongTitle": "Still In Love"<br>} |

| Music 테이블 | GenreAlbumTitle |
| --- | --- |
| <pre>"WJJH"<br>},<br>"TourDates": {<br>"Seattle": "20150622",<br>"Cleveland": "20150630"<br>},<br>"Rotation": "Heavy"<br>}<br>}</pre> | |
| <pre>{<br>"Artist": "The Acme Band",<br>"SongTitle": "Look Out, World",<br>"AlbumTitle": "The Buck Starts<br>Here",<br>"Price": 0.99,<br>"Genre": "Rock"<br>}</pre> | <pre>{<br>"Genre": "Rock",<br>"AlbumTitle": "The Buck Starts<br>Here",<br>"Artist": "The Acme Band",<br>"SongTitle": "Look Out, World"<br>}</pre> |

GenreAlbumTitle 테이블에 대해 다음을 알아 두세요.

- 모든 인덱스는 테이블에 속해 있는데, 이를 인덱스의 기본 테이블이라고 합니다. 앞의 예제에서 Music은 GenreAlbumTitle 인덱스의 기본 테이블입니다.
- DynamoDB는 인덱스를 자동으로 유지 관리합니다. 기본 테이블의 항목을 추가, 업데이트 또는 삭제하면 DynamoDB는 해당 테이블에 속하는 모든 인덱스에서 해당 항목을 추가, 업데이트 또는 삭제합니다.
- 인덱스를 생성할 때는 기본 테이블에서 인덱스로 복사하거나 프로젝션할 속성을 지정합니다. 적어도 DynamoDB는 기본 테이블의 키 속성을 인덱스로 프로젝션합니다. `GenreAlbumTitle`의 경우가 그러한데, `Music` 테이블의 키 속성만 인덱스로 프로젝션됩니다.

GenreAlbumTitle 인덱스를 쿼리하여 특정 장르의 앨범을 모두 찾을 수 있습니다(예: 모든 Rock 앨범). 또한 인덱스를 쿼리하여 특정 앨범 제목이 있는 특정 장르에 속하는 모든 앨범을 찾을 수도 있습니다(예: 제목이 알파벳 H로 시작하는 모든 Country 앨범).

자세한 내용은 DynamoDB에서 보조 인덱스를 사용하여 데이터 액세스 개선 섹션을 참조하세요.

## DynamoDB Streams

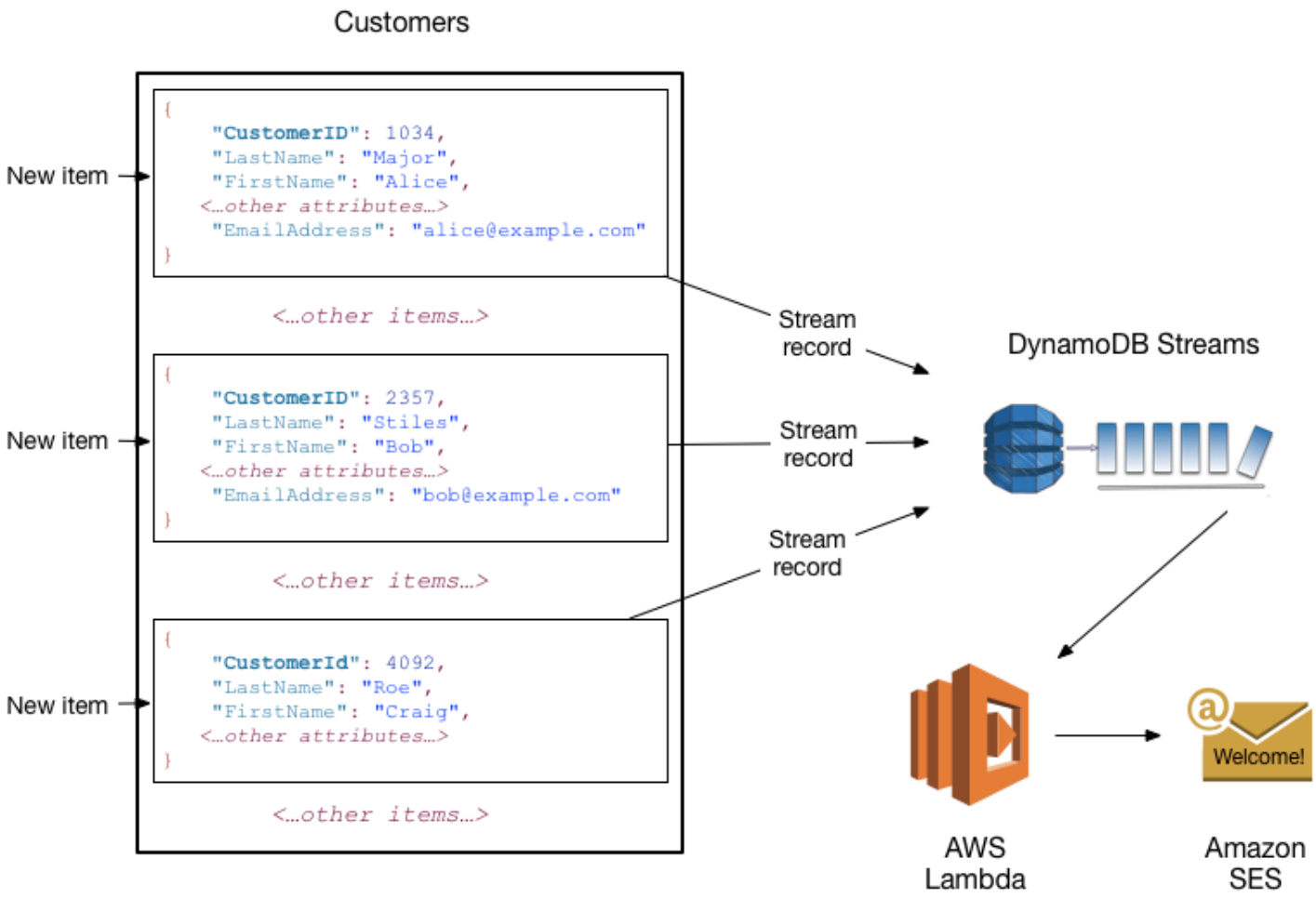
DynamoDB Streams는 DynamoDB 테이블의 데이터 수정 이벤트를 캡처하는 선택적 기능입니다. 이러한 이벤트에 대한 데이터가 이벤트가 발생한 순서대로 거의 실시간으로 스트림에 표시됩니다.

각 이벤트는 스트림 레코드에 의해 나타납니다. 테이블에서 스트림을 활성화하면 다음과 같은 이벤트 중 하나가 발생할 때마다 DynamoDB Streams가 스트림 레코드를 기록합니다.

- 테이블에 새로운 항목이 추가되면 스트림이 해당 속성을 모두 포함하여 전체 항목의 이미지를 캡처합니다.
- 항목이 업데이트되면 스트림이 항목에서 수정된 속성의 "사전" 및 "사후" 이미지를 캡처합니다.
- 테이블에서 항목이 삭제되면 스트림이 항목이 삭제되기 전에 전체 항목의 이미지를 캡처합니다.

각 스트림 레코드에는 또한 테이블의 이름, 이벤트 타임스탬프 및 다른 메타데이터가 포함되어 있습니다. 스트림 레코드의 수명은 24시간이며, 24시간이 지나면 스트림에서 자동으로 제거됩니다.

DynamoDB Streams를 AWS Lambda와 함께 사용하면 트리거, 즉 관심 있는 이벤트가 스트림에 나타날 때마다 자동으로 실행되는 코드를 생성할 수 있습니다. 예를 들어, 회사의 고객 정보가 들어 있는 Customers 테이블을 생각해 볼 수 있습니다. 새 고객마다 "환영" 이메일을 보내려고 한다고 가정해 보세요. 해당 테이블에 스트림을 활성화한 다음, 스트림을 Lambda 함수와 연결할 수 있습니다. Lambda 함수는 새로운 스트림 레코드가 표시될 때마다 실행되지만, Customers 테이블에 추가된 새로운 항목만 처리합니다. `EmailAddress` 속성이 있는 모든 항목에 대해 Lambda 함수는 Amazon Simple Email Service(Amazon SES)를 호출하여 해당 주소에 이메일을 보냅니다.




> **Note**
>
> 이 예제에서 마지막 고객인 Craig Roe는 `EmailAddress`가 없어 이메일을 받지 못합니다.

트리거뿐만 아니라 DynamoDB Streams는 AWS 리전 내나 리전 간의 데이터 복제, DynamoDB 테이블의 구체화된 데이터 보기, Kinesis 구체화된 보기를 사용한 데이터 분석 등의 강력한 솔루션을 지원합니다.

자세한 내용은 DynamoDB Streams에 대한 변경 데이터 캡처 섹션을 참조하세요.

# DynamoDB API

Amazon DynamoDB를 사용하려면 애플리케이션에서 몇 가지 간단한 API 작업을 사용해야 합니다. 다음은 이러한 작업을 카테고리별로 정리한 요약본입니다.

> **Note**
>
> 전체 API 작업 목록은 Amazon DynamoDB API 참조를 참조하세요.

주제


- 컨트롤 플레인
- 데이터 영역
- DynamoDB Streams
- 트랜잭션


## 컨트롤 플레인

제어 영역 작업을 사용하면 DynamoDB 테이블을 생성하고 관리할 수 있습니다. 또한 인덱스, 스트림 및 테이블에 따라 다른 다양한 객체를 사용하도록 해 줍니다.

- `CreateTable` - 새 테이블을 생성합니다. 선택 사항으로, 하나 이상의 보조 인덱스를 생성하고 테이블에 DynamoDB Streams를 활성화할 수 있습니다.
- `DescribeTable` - 기본 키 스키마, 처리량 설정, 인덱스 정보와 같은 테이블에 대한 정보를 반환합니다.
- `ListTables` - 모든 테이블의 이름을 목록으로 반환합니다.
- `UpdateTable` - 테이블 또는 해당 인덱스의 설정을 수정하거나, 테이블에서 새 인덱스를 생성 또는 제거하거나, 테이블의 DynamoDB Streams 설정을 수정합니다.
- `DeleteTable` - DynamoDB에서 테이블 및 해당 종속적 객체 모두를 제거합니다.

## 데이터 영역

데이터 플레인 작업은 테이블의 데이터에 대해 생성, 읽기, 업데이트 및 삭제(CRUD) 작업을 수행하도록 해 줍니다. 또한 일부 데이터 영역 작업을 사용하여 보조 인덱스에서 데이터를 읽을 수 있습니다.

PartiQL - Amazon DynamoDB용 SQL 호환 쿼리 언어을 사용하여 이러한 CRUD 작업을 수행할 수도 있고 각 작업을 고유한 API 호출로 구분하는 DynamoDB의 클래식 CRUD API를 사용할 수도 있습니다.

## PartiQL - SQL 호환 쿼리 언어

- `ExecuteStatement` - 테이블에서 여러 항목을 읽습니다. 테이블에서 단일 항목을 쓰거나 업데이트할 수도 있습니다. 단일 항목을 쓰거나 업데이트할 때는 기본 키 속성을 지정해야 합니다.
- `BatchExecuteStatement` - 테이블에서 여러 항목을 쓰거나, 업데이트하거나, 읽습니다. 항목을 쓰거나 읽을 때 애플리케이션이 네트워크를 한 번만 왕복하면 되므로 `ExecuteStatement`보다 더 효율적입니다.

## 클래식 API

### 데이터 생성

- `PutItem` - 테이블에 단일 항목을 씁니다. 기본 키 속성은 꼭 지정해야 하지만 다른 속성은 지정하지 않아도 됩니다.
- `BatchWriteItem` - 테이블에 최대 25개의 항목을 씁니다. 항목을 쓸 때 애플리케이션이 네트워크를 한 번만 왕복하면 되므로 `PutItem`을 여러 번 호출하는 것보다 이 작업이 효율적입니다.

### 데이터 읽기

- `GetItem` - 테이블에서 단일 항목을 가져옵니다. 원하는 항목의 기본 키를 지정해야 합니다. 전체 항목 또는 속성 일부만 가져올 수 있습니다.
- `BatchGetItem` - 하나 이상의 테이블에서 최대 100개의 항목을 가져옵니다. 항목을 읽을 때 애플리케이션이 네트워크를 한 번만 왕복하면 되므로 `GetItem`을 여러 번 호출하는 것보다 이 작업이 효율적입니다.
- `Query` - 특정 파티션 키가 있는 모든 항목을 가져옵니다. 파티션 키 값을 지정해야 합니다. 전체 항목 또는 속성 일부만 가져올 수 있습니다. 경우에 따라 정렬 키 값에 조건을 적용하여 파티션 키가 동일한 데이터의 하위 집합만 검색할 수도 있습니다. 테이블에 파티션 키와 정렬 키가 모두 있는 경우 이러한 작업을 테이블에 사용할 수 있습니다. 또한 인덱스에 파티션 키와 정렬 키가 모두 있는 경우 이러한 작업을 인덱스에 사용할 수 있습니다.
- `Scan` - 지정한 테이블 또는 인덱스의 모든 항목을 가져옵니다. 전체 항목 또는 속성 일부만 가져올 수 있습니다. 선택 사항으로, 필터링 조건을 적용하여 필요한 값만 반환하고 나머지는 버릴 수 있습니다.

데이터 업데이트

- `UpdateItem` - 항목에서 하나 이상의 속성을 수정합니다. 수정하려는 항목의 기본 키를 지정해야 합니다. 새로운 속성을 추가할 수 있으며 기존 속성을 수정하거나 제거할 수 있습니다. 조건부 업데이트를 수행하여 사용자 정의 조건을 만족할 때만 업데이트가 수행되도록 할 수도 있습니다. 선택 사항으로, 원자성 카운터를 구현할 수 있습니다. 이는 다른 쓰기 요청과 충돌하지 않고도 숫자 속성을 증감합니다.

데이터 삭제

- `DeleteItem` - 테이블에서 단일 항목을 삭제합니다. 삭제하려는 항목의 기본 키를 지정해야 합니다.
- `BatchWriteItem` - 하나 이상의 테이블에서 최대 25개의 항목을 삭제합니다. 항목을 삭제할 때 애플리케이션이 네트워크를 한 번만 왕복하면 되므로 `DeleteItem`을 여러 번 호출하는 것보다 이 작업이 효율적입니다.

> **Note**
>
> 데이터 생성과 삭제에 모두 `BatchWriteItem`을 사용할 수 있습니다.

## DynamoDB Streams

DynamoDB Streams 작업을 사용하면 테이블에서 스트림을 활성화하거나 비활성화할 수 있으며, 스트림에 포함되어 있는 데이터 수정 레코드에 액세스할 수 있습니다.

- `ListStreams` - 모든 스트림 목록 또는 특정 테이블의 스트림만 반환합니다.
- `DescribeStream` - 해당 Amazon 리소스 이름(ARN) 및 애플리케이션이 처음 몇 개 스트림 레코드를 읽기 시작할 수 있는 위치와 같은 스트림에 대한 정보를 반환합니다.
- `GetShardIterator` - 샤드 반복자를 반환합니다. 샤드 반복자는 애플리케이션이 스트림에서 레코드를 가져오는 데 사용하는 데이터 구조입니다.
- `GetRecords` - 지정된 샤드 반복자를 사용하여 하나 이상의 스트림 레코드를 가져옵니다.

## 트랜잭션

트랜잭션은 원자성, 일관성, 격리 및 내구성(ACID)을 제공하여 애플리케이션에서 데이터의 정확성을 더 쉽게 유지할 수 있습니다.

PartiQL - Amazon DynamoDB용 SQL 호환 쿼리 언어을 사용하여 트랜잭션 작업을 수행할 수도 있고 각 작업을 고유한 API 호출로 구분하는 DynamoDB의 클래식 CRUD API를 사용할 수도 있습니다.

### PartiQL - SQL 호환 쿼리 언어

- `ExecuteTransaction` - 양자택일 결과가 보장되는 테이블 내와 테이블 간 모두에서 여러 항목에 대한 CRUD 작업을 허용하는 배치 작업입니다.

### 클래식 API

- `TransactWriteItems` - 양자택일 결과가 보장되는 테이블 내와 테이블 간 모두에서 여러 항목에 대한 `Put`, `Update` 및 `Delete` 작업을 허용하는 배치 작업입니다.
- `TransactGetItems` - 하나 이상의 테이블에서 여러 항목을 가져오는 `Get` 작업을 허용하는 배치 작업입니다.

# Amazon DynamoDB에서 지원되는 데이터 형식 및 이름 지정 규칙

이 섹션에서는 Amazon DynamoDB 이름 지정 규칙 및 DynamoDB가 지원하는 다양한 데이터 형식에 대해 설명합니다. 제한된 데이터 형식이 적용됩니다. 자세한 내용은 데이터 타입 섹션을 참조하세요.

주제

- 이름 지정 규칙
- 데이터 타입
- 데이터 형식 서술자

## 이름 지정 규칙

DynamoDB의 테이블, 속성 및 기타 객체는 이름이 있어야 합니다. 이름은 의미 있고 간결해야 합니다. 예를 들어 Products, Books, Authors와 같이 이름은 쉽게 이해되어야 합니다.

다음은 DynamoDB의 이름 지정 규칙입니다.

- 모든 이름은 UTF-8로 인코딩되어야 하며, 대소문자를 구분합니다.
- 테이블 이름 및 인덱스 이름은 3~255자로 이루어져야 하며, 다음 문자만 포함할 수 있습니다.
  - `a-z`
  - `A-Z`

  - 0-9
  - _ (밑줄)
  - -(대시)
  - .(점)
- 속성 이름은 1자 이상이고 크기가 64KB 미만이어야 합니다. 속성 이름을 최대한 짧게 유지하는 것이 좋습니다. 이렇게 하면 속성 이름이 스토리지 및 처리량 사용량 측정에 포함되므로 사용되는 읽기 요청 단위를 줄일 수 있습니다.

  다음과 같은 예외가 있습니다. 이러한 속성 이름은 255자 이내여야 합니다.
  - 보조 인덱스 파티션 키 이름
  - 보조 인덱스 정렬 키 이름
  - 사용자 지정 프로젝션 속성 이름(로컬 보조 인덱스에만 적용 가능)

### 예약어 및 특수 문자

DynamoDB에 예약어 목록과 특수 문자가 있습니다. 전체 목록은 DynamoDB의 예약어 섹션을 참조하세요. 또한 DynamoDB에서 #(해시) 및 :(콜론)은 특별한 의미를 갖습니다.

DynamoDB에서 이러한 예약어 및 특수 문자를 이름에 사용하도록 허용하는 경우에도 표현식에서 이러한 이름을 사용할 때마다 자리 표시자 변수를 정의해야 하므로 사용하지 않는 것이 좋습니다. 자세한 내용은 DynamoDB의 표현식 속성 이름(별칭) 섹션을 참조하세요.

## 데이터 타입

DynamoDB는 테이블 내 속성에 대해 다양한 데이터 형식을 지원합니다. 이는 다음과 같이 분류할 수 있습니다.

- 스칼라 형식 - 스칼라 형식은 하나의 값만 표현할 수 있습니다. 스칼라 형식은 숫자, 문자열, 이진수, 부울 및 Null입니다.
- 문서 형식 - 문서 형식은 중첩된 속성이 있는 복잡한 구조를 표현할 수 있습니다. 이러한 형식은 JSON 문서에서 찾을 수 있습니다. 문서 형식은 목록 및 맵입니다.
- 집합 형식 - 집합 형식은 여러 스칼라 값을 표현할 수 있습니다. 집합 형식은 문자열 집합, 숫자 집합 및 이진수 집합입니다.

테이블이나 보조 인덱스를 생성할 때는 각 기본 키 속성(파티션 키 및 정렬 키)의 이름 및 데이터 형식을 지정해야 합니다. 또한, 각 기본 키 속성은 문자열, 숫자 또는 이진수 형식으로 정의해야 합니다.

DynamoDB는 스키마 없는 NoSQL 데이터베이스입니다. 이는 테이블을 생성할 때 프라이머리 키 속성 외에는 다른 속성이나 데이터 형식을 정의할 필요가 없다는 의미입니다. 그에 비해, 관계형 데이터베이스는 테이블을 생성할 때 각 열의 이름 및 데이터 형식을 정의해야 합니다.

다음은 각 데이터 형식을 JSON 형식의 예제와 함께 설명한 것입니다.

## 스칼라 형식

스칼라 형식은 숫자, 문자열, 이진수, 부울 및 Null입니다.

### 숫자

번호는 양수, 음수 또는 0일 수 있습니다. 숫자는 최대 38자릿수까지 지원됩니다. 이 제한을 초과하면 예외가 발생합니다. 소수점 아래 38자리보다 높은 정밀도가 필요한 경우 문자열을 사용할 수 있습니다.

- 양의 범위: 1E-130 ~ 9.99999999999999999999999999999999999999E+125
- 음의 범위: -9.99999999999999999999999999999999999999E+125 ~ -1E-130

DynamoDB에서 숫자는 가변 길이로 나타냅니다. 앞과 끝의 0은 잘립니다.

모든 숫자는 언어와 라이브러리 간의 호환성을 극대화하기 위해 네트워크를 통해 DynamoDB에 문자열로 전송됩니다. 하지만 DynamoDB는 해당 문자열을 연산 작업에서 숫자 형식 속성으로 처리합니다.

번호 데이터 형식을 사용하면 날짜 또는 타임스탬프를 표현할 수 있습니다. 이렇게 하는 한 가지 방법은 1970년 1월 1일 00:00:00 UTC 이후 경과된 시간(초)인 epoch 시간을 사용하는 것입니다. 예를 들어, epoch 시간 `1437136300`은 2015년 7월 17일 오후 12:31:40 UTC를 나타냅니다.

자세한 내용은 http://en.wikipedia.org/wiki/Unix_time을 참조하세요.

### String

문자열은 UTF-8 이진수 인코딩을 사용하는 유니코드입니다. 속성이 인덱스 또는 테이블의 키로 사용되지 않고 최대 DynamoDB 항목 크기 제한인 400KB로 제한되는 경우 문자열의 최소 길이는 0일 수 있습니다.

문자열 형식으로 정의된 기본 키 속성에는 다음과 같은 제약이 추가로 적용됩니다.

- 단순 기본 키의 경우 첫 번째 속성 값(파티션 키)의 최대 길이는 2048바이트입니다.
- 복합 기본 키의 경우 두 번째 속성 값(정렬 키)의 최대 길이는 1024바이트입니다.

DynamoDB는 기본 UTF-8 문자열 인코딩의 바이트를 사용하여 문자열을 수집하고 비교합니다. 예를 들어 "a" (0x61)"은 "A" (0x41)보다 크고, "¿" (0xC2BF)는 "z" (0x7A)보다 큽니다.

문자열 데이터 형식을 사용하면 날짜 또는 타임스탬프를 표현할 수 있습니다. 다음 예에서와 같이 ISO 8601 문자열을 사용하여 이러한 작업이 가능합니다.

- `2016-02-15`
- `2015-12-21T17:42:34Z`
- `20150311T122706Z`

자세한 내용은 http://en.wikipedia.org/wiki/ISO_8601을 참조하세요.

> **Note**
>
> 기존 관계형 데이터베이스와 달리 DynamoDB는 기본적으로 날짜 및 시간 데이터 유형을 지원하지 않습니다. 대신 Unix epoch 시간을 사용하여 날짜 및 시간 데이터를 숫자 데이터 유형으로 저장하는 것이 유용할 수 있습니다.

바이너리

이진수 형식의 속성에는 압축 텍스트, 암호화 데이터 또는 이미지 같은 모든 이진수 데이터가 저장될 수 있습니다. DynamoDB는 이진수 값을 비교할 때마다 이진수 데이터의 각 바이트를 부호가 없는 것으로 처리합니다.

속성이 인덱스 또는 테이블의 키로 사용되지 않고 최대 DynamoDB 항목 크기 제한인 400KB로 제한되는 경우 이진 속성의 길이는 0일 수 있습니다.

이진수 형식 속성으로 기본 키 속성을 정의하는 경우 다음 추가 제약이 적용됩니다.

- 단순 기본 키의 경우 첫 번째 속성 값(파티션 키)의 최대 길이는 2048바이트입니다.
- 복합 기본 키의 경우 두 번째 속성 값(정렬 키)의 최대 길이는 1024바이트입니다.

애플리케이션에서는 데이터를 DynamoDB로 보내기 전에 Base64 인코딩 형식으로 이진수 값을 인코딩해야 합니다. DynamoDB가 이러한 값을 받아 데이터를 부호가 없는 바이트 배열로 디코딩하고 이진수 속성 길이로 사용합니다.

다음은 base64 인코딩 텍스트를 사용하여 이진수 속성을 나타낸 예제입니다.

```
dGhpcyB0ZXh0IGlzIGJhc2U2NC1lbmNvZGVk
```

불

부울 형식의 속성은 `true` 또는 `false`를 저장할 수 있습니다.

Null

Null은 알려지지 않았거나 정의되지 않은 상태의 속성을 나타냅니다.

## 문서 형식

문서 형식은 목록 및 맵입니다. 이러한 데이터 형식은 서로 중첩이 가능하여 최대 32개 깊이 수준의 복잡한 데이터 구조까지 나타낼 수 있습니다.

목록 또는 맵 내 값의 수에는 제한이 없습니다. 단, 값을 포함하는 항목이 DynamoDB 항목 크기 제한(400KB)을 초과하지 않아야 합니다.

속성이 테이블 또는 인덱스 키에 사용되지 않는 경우 속성 값은 빈 문자열이거나 빈 이진 값일 수 있습니다. 속성 값은 빈 집합(문자열 집합, 숫자 집합 또는 이진 집합)일 수 없지만 빈 목록 및 맵은 허용됩니다. 목록과 맵 내에서는 빈 문자열과 이진 값이 허용됩니다. 자세한 내용은 속성 섹션을 참조하세요.

나열

목록 형식 속성은 순서가 지정된 값 모음을 저장할 수 있습니다. 목록은 대괄호(`[ ... ]`)로 묶습니다.

목록은 JSON 배열과 유사합니다. 목록 요소에 저장할 수 있는 데이터 형식에는 제한이 없으며, 한 목록 요소에 있는 요소의 형식이 달라도 상관없습니다.

다음은 두 개의 문자열과 하나의 숫자를 포함하는 목록 예제입니다.

```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

> **Note**
>
> DynamoDB는 각 요소가 깊게 중첩되었다고 해도 목록 내의 개별 요소를 사용할 수 있습니다. 자세한 내용은 DynamoDB에서 표현식 사용 섹션을 참조하세요.

### 맵

맵 형식 속성은 정렬되지 않은 이름-값 페어의 모음을 저장할 수 있습니다. 맵은 중괄호({ ... })로 묶습니다.

맵은 JSON 객체와 유사합니다. 맵 요소에 저장할 수 있는 데이터 형식에는 제한이 없으며, 한 맵에 형식이 다른 요소도 함께 있을 수 있습니다.

맵은 DynamoDB에 JSON 문서를 저장하는 데 이상적입니다. 다음은 문자열, 숫자 및 다른 맵을 포함하는 중첩 목록이 저장된 맵을 나타내는 예제입니다.

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

> **Note**
>
> DynamoDB는 각 요소가 깊게 중첩되었다고 해도 맵 내의 개별 요소를 사용할 수 있습니다. 자세한 내용은 DynamoDB에서 표현식 사용 섹션을 참조하세요.

### 집합

DynamoDB에서도 숫자, 문자열 또는 이진수 값의 집합을 나타내는 형식을 지원합니다. 집합 내의 모든 요소의 형식은 동일해야 합니다. 예를 들어, 숫자 집합은 숫자만 포함할 수 있으며, 문자열 집합은 문자열만 포함할 수 있는 식입니다.

집합 내 값의 수에는 제한이 없습니다. 단, 값을 포함하는 항목이 DynamoDB 항목 크기 제한(400KB)을 초과하지 않아야 합니다.

집합 내의 각 값은 고유해야 합니다. 집합 내 값의 순서는 유지되지 않습니다. 따라서 애플리케이션이 집합 내에서 요소가 특정 순서로 유지된다는 가정 하에 실행되지 않아야 합니다. DynamoDB는 빈 집합을 지원하지 않지만 집합 내에서 빈 문자열과 이진 값이 허용됩니다.

다음은 하나의 문자열 집합, 하나의 숫자 집합과 이진 집합이 포함된 예제입니다.

```
["Black", "Green", "Red"]

[42.2, -19, 7.5, 3.14]

["U3Vubnk=", "UmFpbnk=", "U25vd3k="]
```

## 데이터 형식 서술자

하위 수준 DynamoDB API 프로토콜은 데이터 형식 서술자를 DynamoDB에 각 속성을 해석하는 방법을 알려 주는 토큰으로 사용합니다.

다음은 DynamoDB 데이터 형식 서술자의 전체 목록입니다.

- **S** - 문자열
- **N** - 숫자
- **B** - 이진수
- **BOOL** - 부울
- **NULL** - Null
- **M** - 맵
- **L** - 목록
- **SS** - 문자열 집합
- **NS** - 숫자 집합
- **BS** - 이진수 집합

# DynamoDB 테이블 클래스

DynamoDB는 비용을 최적화할 수 있도록 설계된 두 가지 테이블 클래스를 제공합니다. DynamoDB Standard 테이블 클래스가 기본값이며 대다수의 워크로드에 권장됩니다. DynamoDB Standard-Infrequent Access(DynamoDB Standard-IA) 테이블 클래스는 스토리지 비용이 많이 드는 테이블에 최적화되어 있습니다. 예를 들어 애플리케이션 로그, 이전 소셜 미디어 게시물, 전자 상거래 주문 내역 및

과거 게임 성과 같이 자주 액세스하지 않는 데이터를 저장하는 테이블은 Standard-IA 테이블 클래스에 적합합니다. 요금 세부 정보는 Amazon DynamoDB 요금을 참조하세요.

모든 DynamoDB 테이블은 테이블 클래스(기본적으로 DynamoDB Standard)와 연결됩니다. 테이블과 연결된 모든 보조 인덱스는 동일한 테이블 클래스를 사용합니다. 각 테이블 클래스는 데이터 스토리지와 읽기 및 쓰기 요청에 대해 서로 다른 요금이 적용됩니다. 스토리지 및 처리량 사용 패턴에 따라 테이블에 가장 비용 효율적인 테이블 클래스를 선택할 수 있습니다.

테이블 클래스의 선택은 영구적이지 않습니다.AWS Management Console, AWS CLI, 또는 AWS SDK를 사용하여 이 설정을 변경할 수 있습니다. DynamoDB는 또한 단일 리전 테이블 및 글로벌 테이블에 대해 AWS CloudFormation을 사용하여 테이블 클래스를 관리하는 것을 지원합니다. 테이블 클래스 선택에 대한 자세한 내용은 DynamoDB에서 테이블 클래스 선택 시 고려 사항 섹션을 참조하세요.

# DynamoDB의 파티션 및 데이터 배포

Amazon DynamoDB는 데이터를 파티션에 저장합니다. 파티션은 SSD(Solid State Drive)로 백업되는 테이블용 스토리지 할당으로, AWS 리전 내의 여러 가용 영역에 자동으로 복제됩니다. 파티션 관리는 DynamoDB에서 전적으로 처리하므로 사용자가 파티션을 직접 관리할 필요가 없습니다.

테이블을 생성할 때 테이블 초기 상태는 `CREATING`입니다. 이 단계 동안, DynamoDB는 테이블에 충분한 파티션을 할당하여 프로비저닝된 처리량 요구 사항을 만족할 수 있도록 합니다. 테이블 상태가 `ACTIVE`로 전환되면 테이블 데이터를 쓰거나 읽을 수 있습니다.

DynamoDB는 다음과 같은 상황에서 테이블에 추가 파티션을 할당합니다.

- 기존 파티션이 지원할 수 있는 한도를 초과하여 테이블의 할당된 처리량 설정을 늘리는 경우.
- 기존 파티션 용량이 다 차서 추가 스토리지 공간이 필요한 경우.

파티션 관리는 백그라운드에서 자동으로 이루어지므로 애플리케이션에는 표시되지 않습니다. 테이블은 사용 가능한 처리량을 유지하며 프로비저닝된 처리량 요구 사항을 완전히 지원합니다.

자세한 내용은 파티션 키 설계을 참조하세요.

DynamoDB의 글로벌 보조 인덱스도 파티션으로 구성됩니다. 글로벌 보조 인덱스의 데이터는 기본 테이블의 데이터와 별도로 저장되지만 인덱스 파티션은 테이블 파티션과 동일한 방식으로 동작합니다.

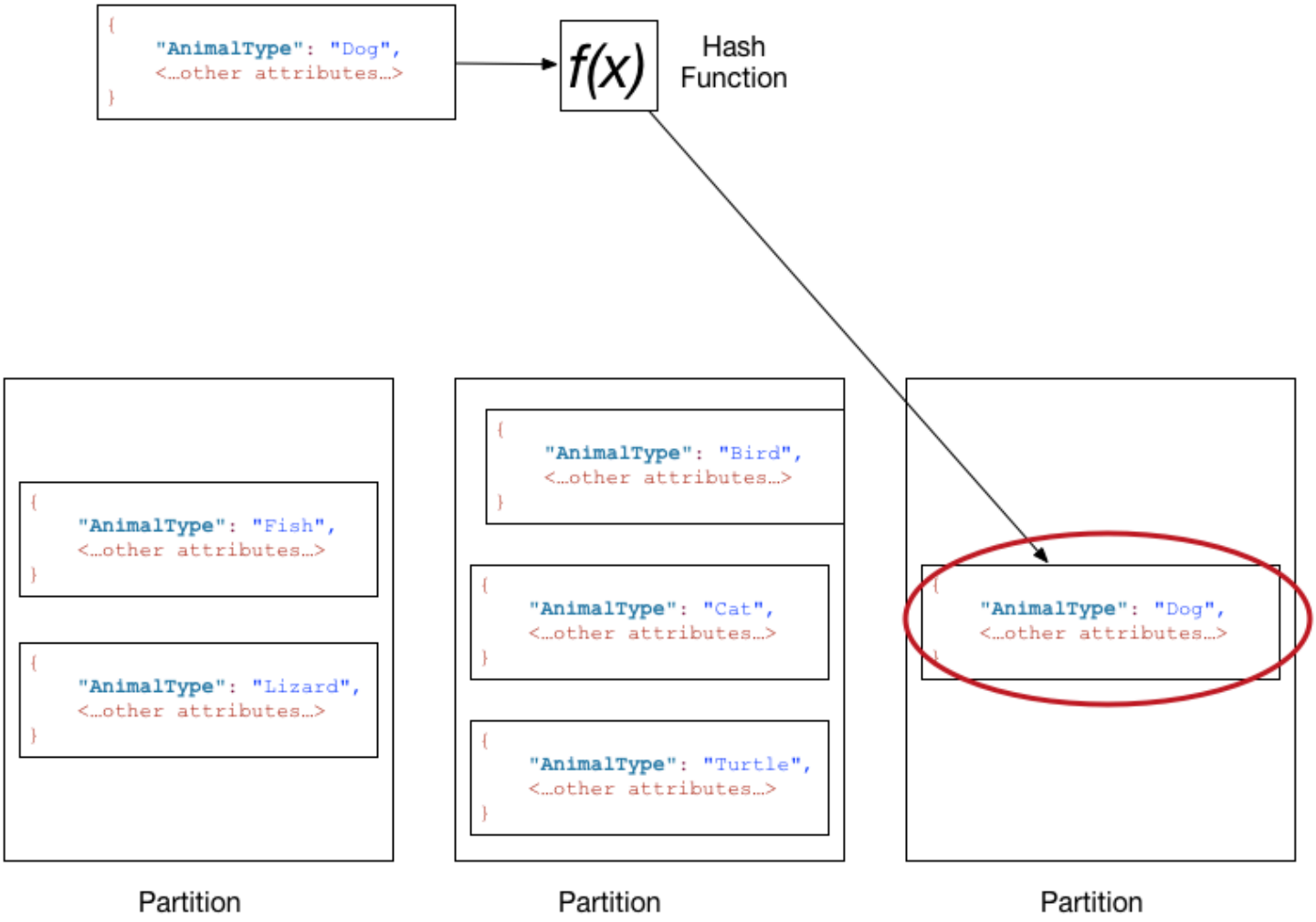
## 데이터 배포: 파티션 키

테이블이 단순 기본 키를 가질 경우(파티션 키만 있음) DynamoDB가 파티션 키 값을 기준으로 각 항목을 저장하고 검색합니다.

테이블에 항목을 쓰기 위해 DynamoDB는 내부 해시 함수에 대한 입력으로 파티션 키 값을 사용합니다. 해시 함수 출력 값은 항목을 저장할 파티션을 결정합니다.

테이블에서 항목을 읽으려면 항목의 파티션 키 값을 지정해야 합니다. DynamoDB는 이 값을 해당 해시 함수의 입력으로 사용하여 항목을 찾을 수 있는 파티션을 결정합니다.

다음 다이어그램은 여러 파티션에 걸쳐 데이터가 저장된 Pets라는 테이블을 보여줍니다. 테이블의 기본 키는AnimalType입니다(이 키 속성만 표시됨). DynamoDB는 해시 함수를 사용하여 새 항목을 저장할 위치를 결정합니다. 이 경우 문자열 Dog의 해시 값이 기준으로 사용됩니다. 항목은 정렬 순서대로 저장되지 않습니다. 각 항목의 위치는 파티션 키의 해시 값으로 결정됩니다.




> **Note**
>
> DynamoDB는 파티션 수와 상관없이 항목을 테이블의 파티션에 균일하게 분배하는 데 최적화되어 있습니다. 테이블 항목 수에 비해 많은 수의 고유 값을 가질 수 있는 파티션 키를 선택할 것을 권장합니다.

# 데이터 배포: 파티션 키 및 정렬 키

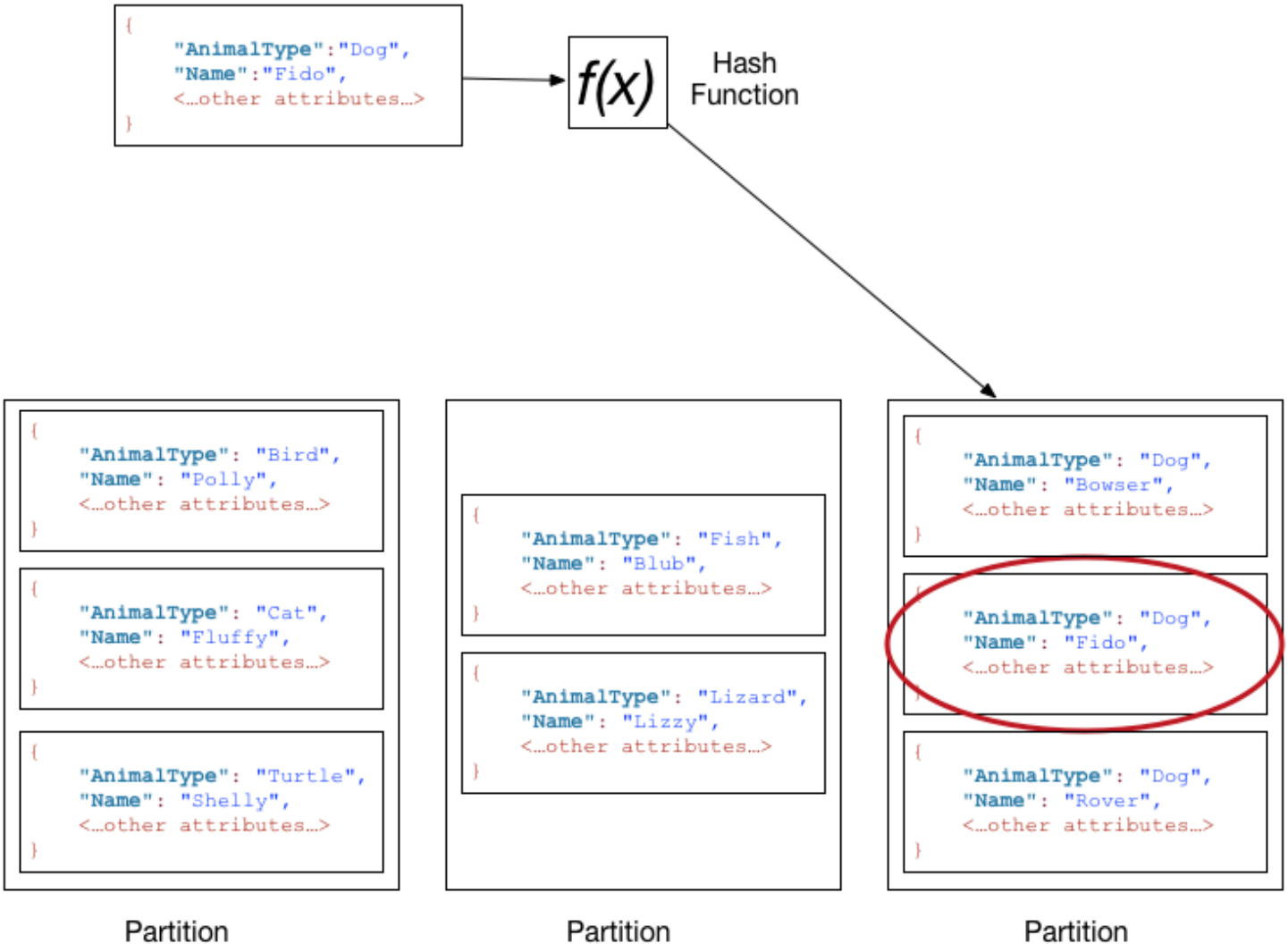
테이블에 복합 기본 키(파티션 키 및 정렬 키)가 있는 경우 데이터 배포: 파티션 키에 설명된 동일한 방식으로 DynamoDB에서 파티션 키의 해시 값을 계산합니다. 하지만 파티션 키 값이 같은 항목을 서로 가깝게 유지하고 정렬 키 속성 값을 기준으로 정렬하는 경향이 있습니다. 동일한 파티션 키 속성 값을 가진 항목 세트를 항목 모음이라고 합니다. 항목 모음은 모음 내 항목 범위를 효율적으로 검색할 수 있도록 최적화되어 있습니다. 테이블에 로컬 보조 인덱스가 없는 경우 DynamoDB는 데이터를 저장하고 읽기 및 쓰기 처리량을 제공하는 데 필요한 만큼의 파티션으로 항목 모음을 자동으로 분할합니다.

테이블에 항목을 기록하기 위해 DynamoDB는 파티션 키의 해시 값을 계산하여 항목을 저장할 파티션을 결정합니다. 해당 파티션에서는 여러 항목이 동일한 파티션 키 값을 가질 수 있습니다. 따라서 DynamoDB에서 동일한 파티션 키의 다른 항목 간에 정렬 키에 의해 오름차순으로 항목을 저장합니다.

테이블에서 항목을 읽으려면 항목의 파티션 키 값과 정렬 키 값을 지정해야 합니다. DynamoDB는 파티션 키의 해시 값을 계산하여 항목을 찾을 수 있는 파티션을 결정합니다.

원하는 항목이 동일한 파티션 키 값을 가질 경우 단일 작업(`Query`)으로 테이블에서 여러 항목을 읽을 수 있습니다. DynamoDB는 해당 파티션 키 값을 갖는 모든 항목을 반환합니다. 특정 값 범위의 항목만 반환되도록 정렬 키에 조건을 적용할 수 있습니다(선택 사항).

Pets 테이블이 AnimalType(파티션 키)와 Name(정렬 키)으로 구성된 복합 기본 키를 가진다고 가정하겠습니다. 다음 다이어그램에서는 DynamoDB가 파티션 키 값 Dog와 정렬 키 값 Fido를 사용하여 항목을 씁니다.




Pets 테이블에서 동일한 항목을 읽기 위해 DynamoDB는 Dog의 해시 값을 계산하여 이들 항목이 저장된 파티션을 알아냅니다. 그런 다음 DynamoDB는 Fido를 찾을 때까지 정렬 키 속성 값을 스캔합니다.

AnimalType이 Dog인 항목을 모두 읽으려면 정렬 키 조건을 지정하지 않고 `Query` 작업을 실행합니다. 기본적으로 항목은 정렬 순서대로, 즉 정렬 키 기준 오름차순으로 반환됩니다. 선택 사항으로 내림차순을 요청할 수 있습니다.

일부 Dog 항목만 쿼리하려면 정렬 키에 조건을 적용할 수 있습니다(예: Name이 `A~K` 범위 내 문자로 시작하는 Dog 항목만).

> **Note**
>
> DynamoDB 테이블에서 파티션 키 값당 고유 정렬 키 값의 수는 상한이 없습니다. Pets 테이블에서 수십억 개의 Dog 항목을 저장해야 하는 경우 DynamoDB는 충분한 스토리지를 할당하여 이 요구 사항을 자동으로 처리합니다.

# SQL에서 NoSQL로 전환하는 방법 알아보기

애플리케이션 개발자라면 RDBMS(관계형 데이터베이스 관리 시스템) 및 SQL(Structured Query Language)을 사용한 경험이 어느 정도 있을 것입니다. Amazon DynamoDB 작업을 시작해 보면 비슷한 점도 많겠지만 다른 점도 상당히 많습니다. NoSQL은 가용성과 확장성이 높고 고성능에 최적화된 비관계형 데이터베이스 시스템을 설명하는 데 사용되는 용어입니다. NoSQL 데이터베이스(예: DynamoDB)는 관계형 모델 대신 키 값 페어나 문서 스토리지 같은 대체 모델을 데이터 관리에 사용합니다. 자세한 내용은 NoSQL이란 무엇입니까?를 참조하세요.

Amazon DynamoDB는 오픈 소스 SQL 호환 쿼리 언어인 PartiQL을 지원합니다. 이 언어를 사용하면 데이터가 저장되는 위치 또는 형식과 관계없이 데이터를 효율적으로 쿼리할 수 있습니다. PartiQL을 사용하면 관계형 데이터베이스의 구조화된 데이터, 개방형 데이터 형식의 반정형 및 중첩 데이터, 행에 따라 다른 속성을 지정할 수 있는 NoSQL 또는 문서 데이터베이스의 스키마 없는 데이터까지도 손쉽게 처리할 수 있습니다. 자세한 내용은 PartiQL 쿼리 언어를 참조하세요.

다음 섹션에서는 SQL 문을 그에 상당하는 DynamoDB 작업과 비교 대조하면서 공통 데이터베이스 작업에 대해 설명합니다.

> **Note**
>
> 이 섹션의 SQL 예제는 MySQL RDBMS와 호환됩니다.
> 이 섹션의 DynamoDB 예제는 DynamoDB 작업의 이름과 함께 JSON 형식의 해당 작업 파라미터를 보여 줍니다.

주제

- 관계형(SQL)과 NoSQL 중에서 선택
- 관계형(SQL) 데이터베이스 및 DynamoDB 액세스 비교
- 테이블 생성 시 관계형(SQL) 데이터베이스 및 DynamoDB 비교
- 관계형(SQL) 데이터베이스 및 DynamoDB에서 테이블 정보 가져오기 비교

- 테이블에 데이터를 쓸 때의 관계형(SQL) 데이터베이스와 DynamoDB 비교
- 테이블에서 데이터를 읽을 때의 관계형(SQL) 데이터베이스와 DynamoDB 비교
- 인덱스 관리 시 관계형(SQL) 데이터베이스와 DynamoDB 비교
- 테이블에서 데이터를 수정할 때 관계형(SQL) 데이터베이스와 DynamoDB 비교
- 테이블에서 데이터를 삭제할 때 관계형(SQL) 데이터베이스와 DynamoDB 비교
- 테이블 제거 시 관계형(SQL) 데이터베이스와 DynamoDB 비교

## 관계형(SQL)과 NoSQL 중에서 선택

오늘날의 애플리케이션의 요구 사항은 그 어느 때보다 까다롭습니다. 예를 들어 온라인 게임은 소수의 사용자와 아주 작은 양의 데이터만으로 시작될 수 있습니다. 하지만 게임이 성공을 거두면 기본 데이터베이스 시스템의 리소스를 가볍게 초과할 수 있습니다. 웹 기반 애플리케이션의 동시 사용자가 수백 명, 수천 명 또는 수백만 명에 이르고, 매일 테라바이트 규모의 데이터가 새로 생성되는 경우는 드물지 않습니다. 이런 애플리케이션의 데이터베이스는 초당 수만(혹은 수십만) 건의 읽기 및 쓰기를 처리해야 합니다.

Amazon DynamoDB는 이런 종류의 워크로드에 적합합니다. 개발자는 작은 크기로 시작하고, 애플리케이션의 인기가 높아짐에 따라 사용률을 점차로 늘릴 수 있습니다. DynamoDB는 대량의 데이터와 매우 많은 수의 사용자를 처리할 수 있도록 크기를 원활하게 조정합니다.

기존 관계형 데이터베이스 모델링과 이를 DynamoDB에 맞게 조정하는 방법에 대한 자세한 내용은 DynamoDB의 관계형 데이터 모델링 모범 사례 섹션을 참조하세요.

다음 표에서는 관계형 데이터베이스 관리 시스템(RDBMS)과 DynamoDB의 몇 가지 중요한 차이점을 보여줍니다.

| 기능 | 관계형 데이터베이스 관리 시스템(RDBMS) | Amazon DynamoDB |
| --- | --- | --- |
| 최적의 워크로드 | 임시 쿼리, 데이터 웨어하우징, OLAP(Online Analytical Processing). | 소셜 네트워크, 게이밍, 미디어 공유, IoT(사물 인터넷) 등 웹 규모의 애플리케이션입니다. |
| 데이터 모델 | 관계형 모델의 경우 데이터가 테이블, 행 및 열로 정규화되는 잘 정의된 스키마가 필요합니다. 뿐만 아니라 테이블, 행, 인 | DynamoDB는 스키마가 없습니다. 각 테이블에는 각각의 데이터 항목을 고유하게 식별하는 기본 키가 있어야 하지만 키가 |

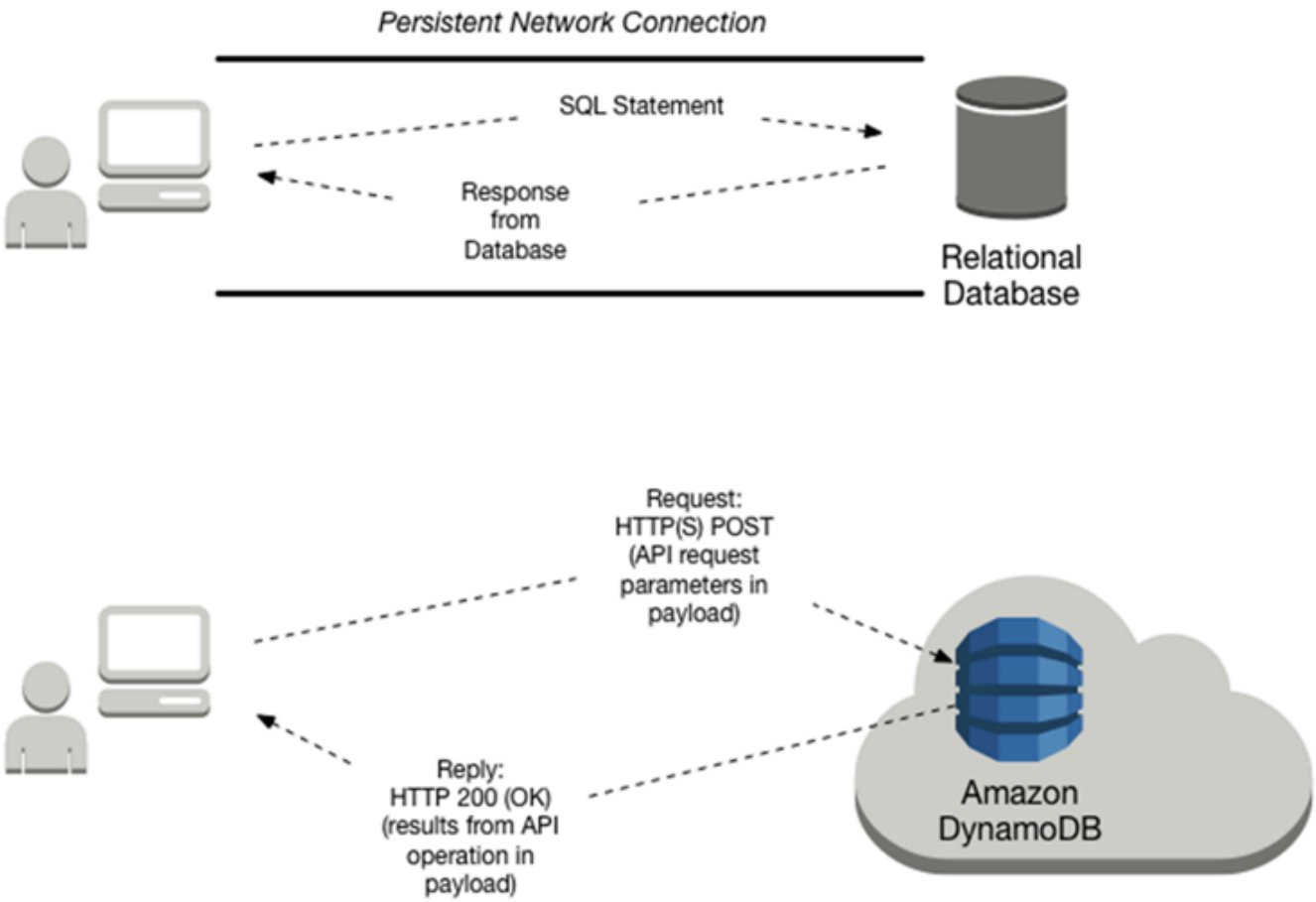
| 기능 | 관계형 데이터베이스 관리 시스템(RDBMS) | Amazon DynamoDB |
| --- | --- | --- |
| | 덱스 및 기타 데이터베이스 간의 모든 관계가 정의됩니다. | 아닌 다른 속성에는 비슷한 제한이 없습니다. DynamoDB는 JSON 문서를 비롯한 정형 또는 반정형 데이터를 관리할 수 있습니다. |
| 데이터 액세스 | SQL은 데이터 저장과 검색을 위한 표준입니다. 관계형 데이터베이스는 데이터베이스 중심 애플리케이션 개발을 단순화하는 풍부한 도구 세트를 제공하지만 이 도구 모두가 SQL을 사용하는 것은 아닙니다. | AWS Management Console, AWS CLI 또는 NoSQL WorkBench를 사용하여 DynamoDB 작업을 수행하고 임시 태스크를 수행할 수 있습니다. SQL 호환 쿼리 언어인 PartiQL을 사용하여 DynamoDB에서 데이터를 선택, 삽입, 업데이트, 삭제할 수 있습니다. 애플리케이션은 AWS 소프트웨어 개발 키트(SDK)를 활용하여 객체 기반, 문서 중심 또는 하위 수준 인터페이스를 통해 DynamoDB 작업을 할 수 있습니다. |
| 성능 | 관계형 데이터베이스는 스토리지에 최적화되어 있으므로 일반적으로 성능은 디스크 하위 시스템에 좌우됩니다. 개발자와 데이터베이스 관리자는 성능 극대화를 위해 쿼리, 인덱스 및 테이블 구조를 최적화해야 합니다. | DynamoDB는 컴퓨팅에 최적화되어 있기 때문에 성능은 주로 기본 하드웨어와 네트워크 대기 시간에 달려 있습니다. 관리형 서비스인 DynamoDB는 사용자와 사용자 애플리케이션이 이런 구현 세부 사항에 영향을 받지 않도록 보호하여 견고한 고성능 애플리케이션을 설계하고 구축하는 데 집중할 수 있게 합니다. |

| 기능 | 관계형 데이터베이스 관리 시스템(RDBMS) | Amazon DynamoDB |
| --- | --- | --- |
| 스케일링 | 보다 빠른 하드웨어를 통한 확장은 더할 나위 없이 쉽습니다. 데이터베이스 테이블을 분산 시스템의 여러 호스트에 배치하는 것도 가능하지만 여기에는 추가 투자가 필요합니다. 관계형 데이터베이스는 파일 숫자와 크기에 최대치가 있어 이것이 확장성의 상한선이 됩니다. | DynamoDB는 분산된 하드웨어 클러스터를 사용하여 확장하도록 설계되었습니다. 이런 설계 덕에 지연 시간 증가 없는 처리 능력 증대가 가능합니다. 고객이 처리량 요구 사항을 지정하면 DynamoDB는 충분한 리소스를 할당하여 이 요구 사항을 충족합니다. 테이블당 항목 수에 상한선이 없고, 테이블의 총 크기 역시 마찬가지입니다. |

## 관계형(SQL) 데이터베이스 및 DynamoDB 액세스 비교

애플리케이션에서 데이터베이스에 액세스할 수 있으려면 애플리케이션에서 해당 데이터베이스를 사용할 수 있도록 애플리케이션이 인증되어야 합니다. 애플리케이션은 해당 권한을 가진 작업만 수행할 수 있도록 권한 부여되어야 합니다.

다음 다이어그램은 관계형 데이터베이스 및 Amazon DynamoDB와의 클라이언트 상호 작용을 보여줍니다.




다음 표에는 클라이언트 상호 작용 작업에 대한 자세한 내용이 나와 있습니다.

| 기능 | 관계형 데이터베이스 관리 시스템(RDBMS) | Amazon DynamoDB |
| --- | --- | --- |
| 데이터베이스 액세스 도구 | 대부분의 관계형 데이터베이스는 명령줄 인터페이스(CLI)를 제공하므로 특별 SQL 문을 입력하고 결과를 즉시 확인할 수 있습니다. | 대부분의 경우에는 애플리케이션 코드를 작성합니다. AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 NoSQL Workbench를 사용하여 DynamoDB에 임시 요청을 보내고 결과를 볼 수도 있습니다. SQL 호환 쿼리 언어인 PartiQL을 사용하여 DynamoDB에서 데이터를 선 |

| 기능 | 관계형 데이터베이스 관리 시스템(RDBMS) | Amazon DynamoDB |
| --- | --- | --- |
| | | 택, 삽입, 업데이트, 삭제할 수 있습니다. |
| 데이터베이스에 연결 | 애플리케이션은 데이터베이스와의 네트워크 연결을 구축하고 유지합니다. 애플리케이션은 종료될 때 연결을 끊습니다. | DynamoDB는 웹 서비스이며, 해당 데이터베이스와 상호 작용은 상태 비저장입니다. 애플리케이션은 지속적인 네트워크 연결을 유지할 필요가 없습니다. 그 대신 DynamoDB와의 상호 작용은 HTTP(S) 요청 및 응답을 사용하여 이루어집니다. |
| 인증 | 애플리케이션은 인증되기 전에는 데이터베이스에 연결할 수 없습니다. RDBMS는 직접 인증을 수행하거나 호스트 운영 체제 또는 디렉터리 서비스에 이 작업을 오프로드할 수 있습니다. | DynamoDB에 대한 모든 요청에는 해당 특정 요청을 인증하는 암호화 서명이 함께 제공되어야 합니다. AWS SDK는 서명 및 서명 요청을 생성하는 데 필요한 모든 로직을 제공합니다. 자세한 내용은 AWS 일반 참조의 AWS API 요청 서명을 참조하세요. |

| 기능 | 관계형 데이터베이스 관리 시스템(RDBMS) | Amazon DynamoDB |
| --- | --- | --- |
| 권한 부여 | 애플리케이션은 권한을 부여받은 작업만 수행할 수 있습니다. 데이터베이스 관리자나 애플리케이션 소유자는 SQL GRANT 및 REVOKE 문을 사용하여 (테이블 등의) 데이터베이스 객체, (테이블 내의 행 등의) 데이터 또는 특정 SQL 문 발행 능력에 대한 액세스를 제어할 수 있습니다. | DynamoDB에서는 AWS Identity and Access Management(IAM)에 의해 권한 부여가 처리됩니다. 테이블과 같은 DynamoDB 리소스에 대한 권한을 부여하는 IAM 정책을 작성한 다음 사용자 및 역할이 해당 정책을 사용하도록 허용할 수 있습니다. IAM은 DynamoDB 테이블의 개별 항목에 대한 액세스를 세부적으로 제어할 수도 있습니다. 자세한 내용은 Amazon DynamoDB의 Identity and Access Management 섹션을 참조하세요. |
| 요청 전송 | 애플리케이션은 수행하려는 모든 데이터베이스 작업에 대해 SQL 문을 발행합니다. SQL 문을 수신하면 RDBMS는 구문을 확인하고 작업 수행 계획을 생성한 다음 계획을 실행합니다. | 애플리케이션은 HTTP(S) 요청을 DynamoDB에 보냅니다. 요청에는 수행할 DynamoDB 작업의 이름과 함께 파라미터가 포함되어 있습니다. DynamoDB는 요청을 즉시 실행합니다. |
| 응답 수신 | RDBMS는 SQL 문의 결과를 반환합니다. 오류가 있는 경우, RDBMS는 오류 상태와 메시지를 반환합니다. | DynamoDB는 작업 결과가 포함된 HTTP(S) 응답을 반환합니다. 오류가 있는 경우 DynamoDB는 HTTP 오류 상태 및 메시지를 반환합니다. |

# 테이블 생성 시 관계형(SQL) 데이터베이스 및 DynamoDB 비교

테이블은 관계형 데이터베이스와 Amazon DynamoDB의 기본 데이터 구조입니다. 관계형 데이터베이스 관리 시스템(RDBMS)에서는 테이블을 생성할 때 테이블의 스키마를 정의해야 합니다. 반면 DynamoDB 테이블은 스키마가 없습니다. 즉, 테이블을 생성할 때 기본 키 외에는 추가 속성이나 데이터 형식을 정의할 필요가 없습니다.

다음 섹션에서는 SQL을 사용하여 테이블을 생성하는 방법과 DynamoDB를 사용하여 테이블을 생성하는 방법을 비교합니다.

주제


- SQL에서 테이블 생성
- DynamoDB에서 테이블 생성


## SQL에서 테이블 생성

SQL에서는 다음 예에 나온 것처럼 `CREATE TABLE` 문을 사용하여 테이블을 생성합니다.

```
CREATE TABLE Music (
    Artist VARCHAR(20) NOT NULL,
    SongTitle VARCHAR(30) NOT NULL,
    AlbumTitle VARCHAR(25),
    Year INT,
    Price FLOAT,
    Genre VARCHAR(10),
    Tags TEXT,
    PRIMARY KEY(Artist, SongTitle)
);
```

이 테이블의 기본 키는 Artist 및 SongTitle로 구성됩니다.

테이블의 모든 열과 데이터 형식 및 테이블의 기본 키를 정의해야 합니다. (필요할 경우, `ALTER TABLE` 문을 사용하여 나중에 이 정의를 변경할 수 있습니다.)

많은 SQL 구현에서는 `CREATE TABLE` 문의 일부로 테이블의 스토리지 사양을 정의할 수 있습니다. 달리 지정하지 않는 한 테이블은 기본 스토리지 설정으로 생성됩니다. 프로덕션 환경에서 데이터베이스 관리자는 최적의 스토리지 파라미터 결정을 도울 수 있습니다.

## DynamoDB에서 테이블 생성

다음 예에 나온 것처럼 `CreateTable` 작업을 사용하여 프로비저닝된 모드 테이블을 생성하고 파라미터를 지정합니다.

```
{
    TableName : "Music",
    KeySchema: [
        {
            AttributeName: "Artist",
            KeyType: "HASH" //Partition key
        },
        {
            AttributeName: "SongTitle",
            KeyType: "RANGE" //Sort key
        }
    ],
    AttributeDefinitions: [
        {
            AttributeName: "Artist",
            AttributeType: "S"
        },
        {
            AttributeName: "SongTitle",
            AttributeType: "S"
        }
    ],
    ProvisionedThroughput: {       // Only specified if using provisioned mode
        ReadCapacityUnits: 1,
        WriteCapacityUnits: 1
    }
}
```

이 테이블의 기본 키는 Artist(파티션 키)와 SongTitle(정렬 키)로 구성됩니다.

다음 파라미터를 `CreateTable`에 입력해야 합니다.

- `TableName` - 테이블 이름
- `KeySchema` - 기본 키에 사용되는 속성. 자세한 내용은 테이블, 항목 및 속성 및 프라이머리 키 섹션을 참조하세요.
- `AttributeDefinitions` - 키 스키마 속성의 데이터 형식

- `ProvisionedThroughput (for provisioned tables)` - 이 테이블에 필요한 초당 읽기 및 쓰기 수. DynamoDB는 처리량 요구 사항이 항상 충족되도록 충분한 스토리지 및 시스템 리소스를 남겨 둡니다. 필요할 경우, `UpdateTable` 작업을 사용하여 나중에 이를 변경할 수 있습니다. 스토리지 할당이 전적으로 DynamoDB에 의해 관리되기 때문에 테이블의 스토리지 요구 사항을 지정할 필요가 없습니다.

## 관계형(SQL) 데이터베이스 및 DynamoDB에서 테이블 정보 가져오기 비교

테이블이 사양에 따라 생성되었음을 확인할 수 있습니다. 관계형 데이터베이스에서는 테이블 스키마가 모두 표시됩니다. Amazon DynamoDB 테이블은 스키마가 없으므로 기본 키 속성만 표시됩니다.

주제

- SQL에서 테이블에 대한 정보 가져오기
- DynamoDB에서 테이블에 대한 정보 가져오기

### SQL에서 테이블에 대한 정보 가져오기

대부분의 관계형 데이터베이스 관리 시스템(RDBMS)에서는 열, 데이터 형식, 기본 키 정의 등과 같은 테이블 구조를 설명할 수 있습니다. SQL에서는 이 작업을 하는 표준적 방법이 없습니다. 다만 많은 데이터베이스 시스템은 `DESCRIBE` 명령을 제공합니다. 다음은 MySQL 예제입니다.

```
DESCRIBE Music;
```

이는 모든 열 이름, 데이터 형식, 크기와 함께 테이블의 구조를 반환합니다.

```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Artist     | varchar(20) | NO   | PRI | NULL    |       |
| SongTitle  | varchar(30) | NO   | PRI | NULL    |       |
| AlbumTitle | varchar(25) | YES  |     | NULL    |       |
| Year       | int(11)     | YES  |     | NULL    |       |
| Price      | float       | YES  |     | NULL    |       |
| Genre      | varchar(10) | YES  |     | NULL    |       |
| Tags       | text        | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

이 테이블의 기본 키는 Artist 및 SongTitle로 구성됩니다.

## DynamoDB에서 테이블에 대한 정보 가져오기

DynamoDB에는 이와 비슷한 `DescribeTable` 작업이 있습니다. 이때는 테이블 이름 파라미터만 있으면 됩니다.

```
{
    TableName : "Music"
}
```

`DescribeTable`의 회신은 다음과 같습니다.

```
{
  "Table": {
    "AttributeDefinitions": [
      {
        "AttributeName": "Artist",
        "AttributeType": "S"
      },
      {
        "AttributeName": "SongTitle",
        "AttributeType": "S"
      }
    ],
    "TableName": "Music",
    "KeySchema": [
      {
        "AttributeName": "Artist",
        "KeyType": "HASH"  //Partition key
      },
      {
        "AttributeName": "SongTitle",
        "KeyType": "RANGE"  //Sort key
      }
    ],

    ...
```

`DescribeTable`은 또한 테이블의 인덱스, 할당된 처리량 설정, 대략적인 항목 카운트 및 기타 메타데이터에 관한 정보를 반환합니다.

# 테이블에 데이터를 쓸 때의 관계형(SQL) 데이터베이스와 DynamoDB 비교

관계형 데이터베이스 테이블에는 데이터의 행이 포함됩니다. 행은 열로 구성됩니다. Amazon DynamoDB 테이블에는 항목이 포함됩니다. 항목은 속성으로 구성됩니다.

이 섹션에서는 테이블에 하나의 행(또는 항목)을 쓰는 방법을 설명합니다.

주제


- SQL에서 테이블에 데이터 쓰기
- DynamoDB에서 테이블에 데이터 쓰기


## SQL에서 테이블에 데이터 쓰기

관계형 데이터베이스의 테이블은 행과 열로 이루어진 2차원 데이터 구조입니다. 일부 데이터베이스 관리 시스템은 보통 기본 JSON 또는 XML 데이터 형식을 사용하여 반정형 데이터 지원도 제공합니다. 하지만 구현의 세부적 내용은 공급업체마다 다릅니다.

SQL에서는 `INSERT` 문을 사용하여 테이블에 행을 추가합니다.

```
INSERT INTO Music
    (Artist, SongTitle, AlbumTitle,
    Year, Price, Genre,
    Tags)
VALUES(
    'No One You Know', 'Call Me Today', 'Somewhat Famous',
    2015, 2.14, 'Country',
    '{"Composers": ["Smith", "Jones", "Davis"],"LengthInSeconds": 214}'
);
```

이 테이블의 기본 키는 Artist 및 SongTitle로 구성됩니다. 이 열들의 값을 지정해야 합니다.

> **Note**
>
> 이 예제에서는 Tags 열을 사용하여 Music 테이블의 노래에 대한 반정형 데이터를 저장합니다. Tags 열은 TEXT 형식으로 정의되었으며 MySQL에 최대 65,535자를 저장할 수 있습니다.

# DynamoDB에서 테이블에 데이터 쓰기

Amazon DynamoDB에서는 DynamoDB API나 PartiQL(SQL 호환 쿼리 언어)을 사용하여 테이블에 항목을 추가할 수 있습니다.

## DynamoDB API

DynamoDB API에서는 `PutItem` 작업을 사용하여 테이블에 항목을 추가합니다.

```
{
    TableName: "Music",
    Item: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today",
        "AlbumTitle":"Somewhat Famous",
        "Year": 2015,
        "Price": 2.14,
        "Genre": "Country",
        "Tags": {
            "Composers": [
                  "Smith",
                  "Jones",
                  "Davis"
            ],
            "LengthInSeconds": 214
        }
    }
}
```

이 테이블의 기본 키는 Artist 및 SongTitle로 구성됩니다. 이 속성들의 값을 지정해야 합니다.

다음은 이 `PutItem` 예제에 관해 알아야 할 주요 사항입니다.

- DynamoDB는 JSON을 사용하여 문서에 대한 기본 지원을 제공합니다. 따라서 DynamoDB는 Tags 같은 반정형 데이터를 저장하는 데 적합합니다. JSON 문서 안에서 데이터를 가져오고 조작할 수도 있습니다.
- Music 테이블에는 기본 키(Artist 및 SongTitle) 외에는 미리 정의된 속성이 없습니다.
- 대부분의 SQL 데이터베이스는 트랜잭션 지향적입니다. `INSERT` 문을 발행하더라도 `COMMIT` 문을 발행하기 전까지는 데이터 수정이 영구적이지 않습니다. Amazon DynamoDB의 경우, DynamoDB가 HTTP 200 상태 코드(OK)로 응답하면 `PutItem` 작업의 결과가 영구적으로 적용됩니다.

다음은 몇 가지 다른 PutItem 예제입니다.

```
{
    TableName: "Music",
    Item: {
        "Artist": "No One You Know",
        "SongTitle": "My Dog Spot",
        "AlbumTitle":"Hey Now",
        "Price": 1.98,
        "Genre": "Country",
        "CriticRating": 8.4
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "No One You Know",
        "SongTitle": "Somewhere Down The Road",
        "AlbumTitle":"Somewhat Famous",
        "Genre": "Country",
        "CriticRating": 8.4,
        "Year": 1984
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "The Acme Band",
        "SongTitle": "Still In Love",
        "AlbumTitle":"The Buck Starts Here",
        "Price": 2.47,
        "Genre": "Rock",
        "PromotionInfo": {
            "RadioStationsPlaying":[
                 "KHCR", "KBQX", "WTNR", "WJJH"
            ],
            "TourDates": {
                "Seattle": "20150625",
                "Cleveland": "20150630"
            },
```

```
            "Rotation": "Heavy"
        }
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "The Acme Band",
        "SongTitle": "Look Out, World",
        "AlbumTitle":"The Buck Starts Here",
        "Price": 0.99,
        "Genre": "Rock"
    }
}
```

> **Note**
>
> DynamoDB는 `PutItem` 외에도 여러 항목을 동시에 쓰는 `BatchWriteItem` 작업을 지원합니다.

### PartiQL for DynamoDB

PartiQL에서는 PartiQL `Insert` 문을 사용하여 `ExecuteStatement` 작업을 통해 테이블에 항목을 추가합니다.

```
INSERT into Music value {
    'Artist': 'No One You Know',
    'SongTitle': 'Call Me Today',
    'AlbumTitle': 'Somewhat Famous',
    'Year' : '2015',
    'Genre' : 'Acme'
}
```

이 테이블의 기본 키는 Artist 및 SongTitle로 구성됩니다. 이 속성들의 값을 지정해야 합니다.

> **Note**
>
> `Insert` 및 `ExecuteStatement`를 사용하는 코드 예제는 DynamoDB의 PartiQL insert 문 섹션을 참조하세요.

## 테이블에서 데이터를 읽을 때의 관계형(SQL) 데이터베이스와 DynamoDB 비교

SQL에서는 `SELECT` 문을 사용하여 테이블에서 하나 이상의 항목을 가져올 수 있습니다. `WHERE` 절을 사용하여 반환되는 데이터를 결정합니다.

한편, Amazon DynamoDB를 사용하는 경우 데이터를 읽기 위해 다음과 같은 작업이 수행됩니다.

- `ExecuteStatement`는 테이블에서 단일 또는 여러 개의 항목을 가져옵니다. `BatchExecuteStatement`는 단일 작업으로 서로 다른 테이블에서 여러 항목을 가져옵니다. 이 두 작업 모두 SQL 호환 쿼리 언어 PartiQL을 사용합니다.
- `GetItem` - 테이블에서 단일 항목을 가져옵니다. 이는 항목의 물리적 위치에 대한 직접 액세스를 제공하기 때문에 단일 항목을 읽는 가장 효율적인 방법입니다. (DynamoDB도 단일 작업으로 최대 100개의 `GetItem` 호출을 수행할 수 있는 `BatchGetItem` 작업을 제공합니다.)
- `Query` - 특정 파티션 키가 있는 항목을 모두 가져옵니다. 이러한 항목 안에서 키를 정렬하고 데이터의 하위 집합만 가져오도록 조건을 적용할 수 있습니다. `Query`는 데이터가 저장된 파티션에 대한 빠르고 효율적인 액세스를 제공합니다. (자세한 내용은 DynamoDB의 파티션 및 데이터 배포 섹션을 참조하세요.)
- `Scan` - 지정한 테이블의 모든 항목을 가져옵니다. (이 작업은 시스템 리소스를 많이 사용할 수 있으므로 큰 테이블에는 사용해서는 안 됩니다.)

> **Note**
>
> 관계형 데이터베이스의 경우, `SELECT` 문을 사용하여 여러 테이블의 데이터를 조인하고 결과를 반환할 수 있습니다. 조인은 관계형 모델의 기초입니다. 조인의 효율적 실행을 위해서는 데이터베이스와 그 애플리케이션의 성능을 지속적으로 튜닝해야 합니다. DynamoDB는 비관계형 NoSQL 데이터베이스로서 테이블 조인을 지원하지 않습니다. 그 대신 애플리케이션은 한 번에 한 테이블의 데이터를 읽습니다.

다음 섹션에서는 다양한 데이터 읽기 사용 사례를 설명하고, 관계형 데이터베이스와 DynamoDB로 이런 작업을 수행하는 방법을 살펴봅니다.

주제

- 프라이머리 키를 사용하여 항목 읽기 비교
- 테이블 쿼리 비교
- 테이블 스캔 비교

## 프라이머리 키를 사용하여 항목 읽기 비교

데이터베이스에 대한 공통적인 액세스 패턴 중 하나는 한 테이블에서 단일 항목을 읽어오는 것입니다. 원하는 항목의 기본 키를 지정해야 합니다.

주제

- SQL에서 항목의 프라이머리 키를 사용하여 항목 읽기
- DynamoDB에서 항목의 프라이머리 키를 사용하여 항목 읽기

### SQL에서 항목의 프라이머리 키를 사용하여 항목 읽기

SQL에서는 SELECT 문을 사용하여 테이블에서 데이터를 가져옵니다. 결과에 하나 이상의 열(또는 * 연산자를 사용하는 경우 모든 열)이 포함되도록 요청할 수 있습니다. WHERE 절은 반환할 행을 결정합니다.

다음은 Music 테이블에서 단일 행을 가져오는 SELECT 문입니다. WHERE 절은 기본 키 값을 지정합니다.

```
SELECT *
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

이 쿼리를 수정하여 열의 하위 집합만 검색할 수 있습니다.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

이 테이블의 프라이머리 키는 Artist 및 SongTitle로 구성되어 있다는 점에 유의하세요.

## DynamoDB에서 항목의 프라이머리 키를 사용하여 항목 읽기

Amazon DynamoDB에서는 DynamoDB API나 PartiQL(SQL 호환 쿼리 언어)을 사용하여 테이블에서 항목을 읽을 수 있습니다.

### DynamoDB API

DynamoDB API에서는 `PutItem` 작업을 사용하여 테이블에 항목을 추가합니다.

DynamoDB는 항목을 해당 프라이머리 키를 기준으로 가져오기 위한 `GetItem` 작업을 제공합니다. `GetItem`은 항목의 물리적 위치에 대한 직접 액세스를 제공하므로 매우 효율적입니다. (자세한 내용은 DynamoDB의 파티션 및 데이터 배포 섹션을 참조하세요.)

기본적으로 `GetItem`은 모든 속성과 함께 전체 항목을 반환합니다.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    }
}
```

속성의 일부만 반환하는 `ProjectionExpression` 파라미터를 추가할 수 있습니다.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    },
    "ProjectionExpression": "AlbumTitle, Year, Price"
}
```

이 테이블의 프라이머리 키는 Artist 및 SongTitle로 구성되어 있다는 점에 유의하세요.

DynamoDB `GetItem` 작업은 매우 효율적입니다. 프라이머리 키 값을 사용하여 해당 항목의 정확한 스토리지 위치를 확인하고 그 위치에서 직접 항목을 가져옵니다. SQL `SELECT` 문도 기본 키 값을 기준으로 항목을 가져오는 경우에는 마찬가지로 효율적입니다.

SQL `SELECT` 문은 다양한 쿼리 및 테이블 스캔을 지원합니다. DynamoDB는 테이블 쿼리 비교 및 테이블 스캔 비교에 설명된 대로 `Query` 및 `Scan` 작업을 통해 이와 비슷한 기능을 제공합니다.

SQL SELECT 문은 테이블 조인을 수행하여 한 번에 여러 테이블에서 데이터를 가져올 수 있습니다. 조인은 데이터베이스 테이블들이 정규화되어 있고 테이블 간 관계가 명확한 경우에 가장 효과적입니다. 그러나 하나의 SELECT 문에서 지나치게 많은 테이블을 조인하는 경우, 애플리케이션 성능이 영향을 받을 수 있습니다. 이런 문제는 데이터베이스 복제, 구체화된 보기 또는 쿼리 다시 쓰기를 사용하여 해결할 수 있습니다.

DynamoDB는 비관계형 데이터베이스로, 테이블 조인을 지원하지 않습니다. 기존 애플리케이션을 관계형 데이터베이스에서 DynamoDB로 마이그레이션하는 경우, 조인의 필요성을 제거하기 위해 데이터 모델을 비정규화해야 합니다.

PartiQL for DynamoDB

PartiQL에서는 PartiQL Select 문을 사용하여 ExecuteStatement 작업을 통해 테이블에서 항목을 읽습니다.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

이 테이블의 프라이머리 키는 Artist 및 SongTitle로 구성되어 있다는 점에 유의하세요.

> **Note**
>
> Select PartiQL 문은 DynamoDB 테이블을 쿼리하거나 스캔하는 데에도 사용할 수 있습니다.

Select 및 ExecuteStatement를 사용하는 코드 예제는 DynamoDB의 PartiQL select 문 섹션을 참조하세요.

## 테이블 쿼리 비교

또 다른 공통적인 액세스 패턴은 쿼리 기준에 기반하여 한 테이블에서 복수의 항목을 읽어오는 것입니다.

주제

- SQL에서 테이블 쿼리
- DynamoDB에서 테이블 쿼리

### SQL에서 테이블 쿼리

SQL SELECT 문을 사용하여 키 열, 키 이외 열 또는 임의의 조합에서 쿼리를 수행할 수 있습니다. 다음 예제와 같이 WHERE 절은 반환되는 행을 결정합니다.

```
/* Return a single song, by primary key */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today';
```

```
/* Return all of the songs by an artist */

SELECT * FROM Music
WHERE Artist='No One You Know';
```

```
/* Return all of the songs by an artist, matching first part of title */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE 'Call%';
```

```
/* Return all of the songs by an artist, with a particular word in the title...
...but only if the price is less than 1.00 */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE '%Today%'
AND Price < 1.00;
```

이 테이블의 프라이머리 키는 Artist 및 SongTitle로 구성되어 있다는 점에 유의하세요.

### DynamoDB에서 테이블 쿼리

Amazon DynamoDB에서는 DynamoDB API나 PartiQL(SQL 호환 쿼리 언어)을 사용하여 테이블에서 항목을 쿼리할 수 있습니다.

#### DynamoDB API

Amazon DynamoDB의 Query 작업을 사용해도 비슷한 방법으로 데이터를 가져올 수 있습니다. Query 작업은 데이터가 저장된 물리적 위치에 대한 빠르고 효율적인 액세스를 제공합니다. 자세한 내용은 DynamoDB의 파티션 및 데이터 배포 섹션을 참조하세요.

테이블 또는 보조 인덱스와 함께 Query를 사용할 수 있습니다. 파티션 키 값에는 등식 조건을 지정해야 하며 지정된 경우 정렬 키 속성에 대해 다른 조건을 선택적으로 지정할 수 있습니다.

KeyConditionExpression 파라미터는 쿼리하려는 키 값을 지정합니다. 선택 사항인 FilterExpression을 사용하면 반환 전에 일정 항목을 결과에서 제거할 수 있습니다.

DynamoDB에서는 ExpressionAttributeValues를 표현식 파라미터(KeyConditionExpression, FilterExpression 등)에서 자리 표시자로 사용해야 합니다. 이는 런타임에 실제 값을 SELECT 문으로 대체하는 관계형 데이터베이스의 바인드 변수 사용과 유사합니다.

이 테이블의 프라이머리 키는 Artist 및 SongTitle로 구성되어 있다는 점에 유의하세요.

다음은 몇 가지 DynamoDB Query 예제입니다.

```
// Return a single song, by primary key

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and SongTitle = :t",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call Me Today"
    }
}
```

```
// Return all of the songs by an artist

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a",
    ExpressionAttributeValues: {
        ":a": "No One You Know"
    }
}
```

```
// Return all of the songs by an artist, matching first part of title

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and begins_with(SongTitle, :t)",
```

```
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call"
    }
}
```

PartiQL for DynamoDB

PartiQL에서는 `ExecuteStatement` 작업과 파티션 키에 `Select` 문을 사용하여 쿼리를 수행할 수 있습니다.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know'
```

이와 같이 `SELECT` 문을 사용하면 이 특정 `Artist`와 연결된 모든 곡이 반환됩니다.

`Select` 및 `ExecuteStatement`를 사용하는 코드 예제는 DynamoDB의 PartiQL select 문 섹션을 참조하세요.

## 테이블 스캔 비교

SQL에서 `WHERE` 절이 없는 `SELECT` 문은 테이블의 모든 행을 반환합니다. Amazon DynamoDB에서 `Scan` 작업도 동일한 작업을 수행합니다. 두 경우 모두 모든 항목 또는 일부 항목만 검색할 수 있습니다.

SQL 데이터베이스를 사용하건 NoSQL 데이터베이스를 사용하건 스캔은 시스템 리소스를 많이 사용할 수 있으므로 꼭 필요할 때만 사용해야 합니다. 스캔이 적절하거나(작은 테이블을 스캔할 때) 불가피한(데이터를 대량으로 내보낼 때) 경우도 있습니다. 하지만 일반적으로는 스캔 수행을 피할 수 있도록 애플리케이션을 설계해야 합니다. 자세한 내용은 DynamoDB에서 테이블 쿼리 섹션을 참조하세요.

> **Note**
>
> 대량 내보내기를 수행하면 파티션당 하나 이상의 파일이 만들어집니다. 각 파일의 모든 항목은 해당 파티션의 해시된 키스페이스에서 가져온 것입니다.

주제

- SQL에서 테이블 스캔

- DynamoDB에서 테이블 스캔

### SQL에서 테이블 스캔

SQL에서는 WHERE 절을 지정하지 않고 SELECT 문을 사용하여 테이블을 스캔하고 모든 데이터를 가져올 수 있습니다. 결과에 하나 이상의 열이 포함되도록 요청할 수 있습니다. 또는 와일드카드 문자(*)를 사용하는 경우 모든 열을 요청할 수 있습니다.

다음은 SELECT 문 사용의 예입니다.

```
/* Return all of the data in the table */
SELECT * FROM Music;
```

```
/* Return all of the values for Artist and Title */
SELECT Artist, Title FROM Music;
```

### DynamoDB에서 테이블 스캔

Amazon DynamoDB에서는 DynamoDB API나 SQL PartiQL(호환 쿼리 언어)을 사용하여 테이블에서 스캔을 수행할 수 있습니다.

#### DynamoDB API

DynamoDB API에서는 Scan 작업을 통해 테이블 또는 보조 인덱스의 모든 항목에 액세스하여 하나 이상의 항목 및 항목 속성을 반환합니다.

```
// Return all of the data in the table
{
    TableName:  "Music"
}
```

```
// Return all of the values for Artist and Title
{
    TableName:  "Music",
    ProjectionExpression: "Artist, Title"
}
```

Scan 작업도 결과에 표시되기를 원치 않는 항목을 삭제하는 데 사용할 수 있는 FilterExpression 파라미터를 제공합니다. FilterExpression은 스캔이 수행된 후 결과가

반환되기 전에 적용됩니다. (큰 테이블에는 사용하지 않는 것이 좋습니다. 일치하는 소수의 항목이 반환되더라도 여전히 전체 Scan에 대한 요금이 청구됩니다.)

PartiQL for DynamoDB

PartiQL에서는 ExecuteStatement 작업을 통해 Select 문을 사용하여 테이블의 모든 내용을 반환하여 스캔을 수행합니다.

```
SELECT AlbumTitle, Year, Price
FROM Music
```

이 문은 Music 테이블에서 모든 항목을 반환합니다.

Select 및 ExecuteStatement를 사용하는 코드 예제는 DynamoDB의 PartiQL select 문 섹션을 참조하세요.

## 인덱스 관리 시 관계형(SQL) 데이터베이스와 DynamoDB 비교

인덱스는 대체 쿼리 패턴에 대한 액세스를 제공하고, 쿼리 속도를 높일 수 있습니다. 이 섹션에서는 SQL에서의 인덱스 생성 및 사용을 Amazon DynamoDB의 경우와 비교 대조합니다.

관계형 데이터베이스를 사용하건 DynamoDB를 사용하건 인덱스 생성에는 신중을 기해야 합니다. 테이블에서 쓰기가 이루어질 때마다 테이블의 모든 인덱스가 업데이트되어야 합니다. 테이블이 크고 쓰기 작업이 많은 환경에서는 시스템 리소스가 많이 사용될 수 있습니다. 읽기 전용 또는 읽기가 주를 이루는 환경에서는 그 정도로 문제가 되지는 않지만 인덱스가 단순히 공간을 차지하는 것이 아니라 실제로 애플리케이션에 의해 사용되어야 합니다.

주제

- 인덱스 생성 시 관계형(SQL) 데이터베이스와 DynamoDB 비교
- 인덱스를 쿼리하고 스캔할 때 관계형(SQL) 데이터베이스와 DynamoDB 비교

### 인덱스 생성 시 관계형(SQL) 데이터베이스와 DynamoDB 비교

SQL의 CREATE INDEX 문을 Amazon DynamoDB의 UpdateTable 작업과 비교합니다.

주제

- SQL에서 인덱스 생성
- DynamoDB에서 인덱스 생성

### SQL에서 인덱스 생성

관계형 데이터베이스에서 인덱스는 한 테이블의 여러 열에서 빠른 쿼리를 수행할 수 있는 데이터 구조입니다. `CREATE INDEX` SQL 문을 사용하면 기존 테이블에 인덱스를 추가하여 열을 인덱싱하도록 지정할 수 있습니다. 인덱스가 생성된 후 테이블에서 평소처럼 데이터를 쿼리할 수 있지만 데이터베이스는 이제 전체 테이블을 스캔하는 대신 인덱스를 사용하여 지정된 행을 테이블에서 빠르게 찾을 수 있습니다.

인덱스 생성 후 데이터베이스는 인덱스를 유지합니다. 테이블의 데이터를 수정할 때마다 테이블의 변경 사항을 반영하여 인덱스도 자동으로 수정됩니다.

MySQL에서 인덱스를 생성하는 방법은 다음과 같습니다.

```
CREATE INDEX GenreAndPriceIndex
ON Music (genre, price);
```

### DynamoDB에서 인덱스 생성

DynamoDB에서는 비슷한 용도의 보조 인덱스를 생성하고 사용할 수 있습니다.

DynamoDB의 인덱스는 관계형 데이터베이스의 인덱스와 다릅니다. 보조 인덱스를 생성할 때는 해당 키 속성, 즉 파티션 키와 정렬 키를 지정해야 합니다. 보조 인덱스를 생성한 후 테이블과 마찬가지로 보조 인덱스를 `Query` 또는 `Scan`할 수 있습니다. DynamoDB에는 쿼리 최적화 프로그램이 없으므로 보조 인덱스는 `Query` 또는 `Scan`할 때만 사용됩니다.

DynamoDB는 다음과 같이 두 종류의 인덱스를 지원합니다.

- 글로벌 보조 인덱스 - 이 인덱스의 기본 키는 해당 테이블의 두 가지 속성 중 어느 것이나 될 수 있습니다.
- 로컬 보조 인덱스 - 이 인덱스의 파티션 키는 해당 테이블의 파티션 키와 동일해야 합니다. 하지만 정렬 키는 다른 속성이어도 됩니다.

DynamoDB에서는 보조 인덱스의 데이터가 해당 테이블과 최종적으로 일관되어야 합니다. 테이블이나 로컬 보조 인덱스에서는 강력히 일관된 `Query` 또는 `Scan` 작업을 요청할 수 있습니다. 그러나 글로벌 보조 인덱스는 최종 일관성만 지원합니다.

`UpdateTable` 작업을 사용하고 `GlobalSecondaryIndexUpdates`를 지정하여 글로벌 보조 인덱스를 기존 테이블에 추가할 수 있습니다.

```
{
    TableName: "Music",
    AttributeDefinitions:[
        {AttributeName: "Genre", AttributeType: "S"},
        {AttributeName: "Price", AttributeType: "N"}
    ],
    GlobalSecondaryIndexUpdates: [
        {
            Create: {
                IndexName: "GenreAndPriceIndex",
                KeySchema: [
                    {AttributeName: "Genre", KeyType: "HASH"}, //Partition key
                    {AttributeName: "Price", KeyType: "RANGE"}, //Sort key
                ],
                Projection: {
                    "ProjectionType": "ALL"
                },
                ProvisionedThroughput: {                                // Only
 specified if using provisioned mode
                    "ReadCapacityUnits": 1,"WriteCapacityUnits": 1
                }
            }
        }
    ]
}
```

다음 파라미터를 UpdateTable에 입력해야 합니다.

- TableName - 인덱스가 연동될 테이블
- AttributeDefinitions - 인덱스의 키 스키마 속성에 대한 데이터 형식
- GlobalSecondaryIndexUpdates - 생성하려는 인덱스에 관한 세부 정보
  - IndexName - 인덱스의 이름
  - KeySchema - 인덱스 기본 키에 사용되는 속성
  - Projection - 테이블에서 인덱스로 복사되는 속성. 이 경우, ALL은 테이블에서 인덱스로 복사되는 모든 속성을 뜻합니다.
  - ProvisionedThroughput (for provisioned tables) - 이 인덱스에 필요한 초당 읽기 및 쓰기 수. (이것은 테이블의 할당 처리량 설정과 별개입니다.)

이 작업에는 테이블의 데이터를 새 인덱스에 채우기가 포함됩니다. 채우기 중에 테이블은 사용 가능한 상태를 유지합니다. 하지만 인덱스의 `Backfilling` 속성이 true에서 false로 바뀔 때까지는 인덱스를 사용할 수 없습니다. `DescribeTable` 작업을 사용하여 이 속성을 볼 수 있습니다.

## 인덱스를 쿼리하고 스캔할 때 관계형(SQL) 데이터베이스와 DynamoDB 비교

Amazon DynamoDB의 `Query` 및 `Scan` 작업과 SQL의 SELECT 문을 사용한 인덱스 쿼리 및 스캔을 비교합니다.

주제

- SQL에서 인덱스 쿼리 및 스캔
- DynamoDB에서 인덱스 쿼리 및 스캔

### SQL에서 인덱스 쿼리 및 스캔

관계형 데이터베이스에서는 직접 인덱스를 다루지 않습니다. 그 대신 `SELECT` 문을 실행하여 테이블을 쿼리하며, 쿼리 옵티마이저가 인덱스를 사용할 수 있습니다.

쿼리 옵티마이저는 사용 가능한 인덱스를 평가하고 그 인덱스로 쿼리 속도를 높일 수 있는지 파악하는 RDBMS(관계형 데이터베이스 관리 시스템)의 구성 요소입니다. 인덱스를 사용하여 쿼리 속도를 높일 수 있는 경우, RDBMS는 먼저 인덱스에 액세스한 다음 인덱스를 사용해 테이블에서 데이터를 찾습니다.

다음은 GenreAndPriceIndex를 사용해 성능을 개선할 수 있는 몇 가지 SQL 문입니다. Music 테이블에 충분한 데이터가 있어서 쿼리 옵티마이저가 전체 테이블을 단순히 스캔하기보다는 이 인덱스를 사용하기로 결정한다고 가정합니다.

```
/* All of the rock songs */

SELECT * FROM Music
WHERE Genre = 'Rock';
```

```
/* All of the cheap country songs */

SELECT Artist, SongTitle, Price FROM Music
WHERE Genre = 'Country' AND Price < 0.50;
```

### DynamoDB에서 인덱스 쿼리 및 스캔

DynamoDB에서는 테이블에서와 동일하게 인덱스에서 직접 `Query` 및 `Scan` 작업을 수행합니다. DynamoDB API 또는 PartiQL(SQL 호환 쿼리 언어)을 사용하여 인덱스를 쿼리하거나 스캔할 수 있습니다. `TableName`과 `IndexName`을 모두 지정해야 합니다.

다음은 DynamoDB의 GenreAndPriceIndex에 대한 몇 가지 쿼리입니다. (이 인덱스의 키 스키마는 Genre와 Price로 구성됩니다.)

DynamoDB API

```
// All of the rock songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre",
    ExpressionAttributeValues: {
        ":genre": "Rock"
    },
};
```

이 예제에서는 속성 전부가 아니라 일부만 결과에 표시하려 한다는 것을 나타내기 위해 `ProjectionExpression`을 사용합니다.

```
// All of the cheap country songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre and Price < :price",
    ExpressionAttributeValues: {
        ":genre": "Country",
        ":price": 0.50
    },
    ProjectionExpression: "Artist, SongTitle, Price"
};
```

다음은 GenreAndPriceIndex에서의 스캔입니다.

```
// Return all of the data in the index
```

```
{
    TableName:  "Music",
    IndexName: "GenreAndPriceIndex"
}
```

PartiQL for DynamoDB

PartiQL에서는 PartiQL `Select` 문을 사용하여 인덱스에 대한 쿼리 및 스캔을 수행합니다.

```
// All of the rock songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock'
```

```
// All of the cheap country songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock' AND Price < 0.50
```

다음은 GenreAndPriceIndex에서의 스캔입니다.

```
// Return all of the data in the index

SELECT *
FROM Music.GenreAndPriceIndex
```

> **Note**
>
> `Select`를 사용하는 코드 예제는 DynamoDB의 PartiQL select 문 섹션을 참조하세요.

## 테이블에서 데이터를 수정할 때 관계형(SQL) 데이터베이스와 DynamoDB 비교

SQL 언어는 데이터 수정을 위한 `UPDATE` 문을 제공합니다. Amazon DynamoDB는 비슷한 태스크를 위해 `UpdateItem` 작업을 사용합니다.

주제


- SQL에서 테이블의 데이터 수정
- DynamoDB에서 테이블의 데이터 수정


## SQL에서 테이블의 데이터 수정

SQL에서는 UPDATE 문을 사용하여 하나 이상의 행을 수정합니다. SET 절은 하나 이상의 열의 새로운 값을 지정하고, WHERE 절은 어떤 행이 수정되는지 결정합니다. 다음은 예입니다.

```
UPDATE Music
SET RecordLabel = 'Global Records'
WHERE Artist = 'No One You Know' AND SongTitle = 'Call Me Today';
```

WHERE 절과 일치하는 행이 없으면 UPDATE 문은 영향을 미치지 않습니다.

## DynamoDB에서 테이블의 데이터 수정

DynamoDB에서는 클래식 API나 PartiQL(SQL 호환 쿼리 언어)을 사용하여 단일 항목을 수정할 수 있습니다. 복수의 항목을 수정하려면 복수의 작업을 사용해야 합니다.

DynamoDB API

DynamoDB API에서는 UpdateItem 작업을 사용하여 단일 항목을 수정합니다.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ExpressionAttributeValues: {
        ":label": "Global Records"
    }
}
```

수정할 항목의 Key 속성과 속성 값을 지정하는 UpdateExpression을 지정해야 합니다. UpdateItem은 'upsert' 작업처럼 동작합니다. 즉 항목이 테이블에 존재하면 업데이트되지만 그렇지 않으면 새 항목이 추가(삽입)됩니다.

UpdateItem은 조건부 쓰기를 지원하는데, 특정 ConditionExpression이 true로 평가되는 경우에 한해 작업이 성공합니다. 예를 들어, 다음 UpdateItem 작업은 노래의 가격이 2.00 이상이 아니면 업데이트를 수행하지 않습니다.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ConditionExpression: "Price >= :p",
    ExpressionAttributeValues: {
        ":label": "Global Records",
        ":p": 2.00
    }
}
```

UpdateItem은 원자성 카운터 즉, 증가하거나 감소할 수 있는 Number 형식의 속성도 지원합니다. 원자성 카운터는 여러 가지 면에서 SQL 데이터베이스의 시퀀스 발생기, 자격 증명 열 또는 자동 증분 필드와 유사합니다.

다음은 UpdateItem 작업을 통해 새 속성(Plays)을 초기화하여 노래가 재생된 횟수를 추적하는 예제입니다.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = :val",
    ExpressionAttributeValues: {
        ":val": 0
    },
    ReturnValues: "UPDATED_NEW"
}
```

ReturnValues 파라미터는 업데이트된 속성의 새 값을 반환하는 UPDATED_NEW로 설정됩니다. 이 경우에는 0을 반환합니다.

누군가가 이 노래를 재생할 때마다 다음 UpdateItem 작업을 사용하여 Plays를 1씩 증가시킬 수 있습니다.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = Plays + :incr",
    ExpressionAttributeValues: {
        ":incr": 1
    },
    ReturnValues: "UPDATED_NEW"
}
```

PartiQL for DynamoDB

PartiQL에서는 PartiQL Update 문을 사용하여 ExecuteStatement 작업을 통해 테이블의 항목을 수정합니다.

이 테이블의 기본 키는 Artist 및 SongTitle로 구성됩니다. 이 속성들의 값을 지정해야 합니다.

```
UPDATE Music
SET RecordLabel ='Global Records'
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

다음 예와 같이 여러 필드를 한 번에 수정할 수도 있습니다.

```
UPDATE Music
SET RecordLabel = 'Global Records'
SET AwardsWon = 10
WHERE Artist ='No One You Know' AND SongTitle='Call Me Today'
```

Update은 원자성 카운터 즉, 증가하거나 감소할 수 있는 Number 형식의 속성도 지원합니다. 원자성 카운터는 여러 가지 면에서 SQL 데이터베이스의 시퀀스 발생기, 자격 증명 열 또는 자동 증분 필드와 유사합니다.

다음은 Update 문을 통해 새 속성(Plays)을 초기화하여 노래가 재생된 횟수를 추적하는 예제입니다.

```
UPDATE Music
```

```
SET Plays = 0
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

누군가가 이 노래를 재생할 때마다 다음 Update 문을 사용하여 Plays를 1씩 증가시킬 수 있습니다.

```
UPDATE Music
SET Plays = Plays + 1
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

> **Note**
>
> Update 및 ExecuteStatement를 사용하는 코드 예제는 DynamoDB의 PartiQL update 문 섹션을 참조하세요.

## 테이블에서 데이터를 삭제할 때 관계형(SQL) 데이터베이스와 DynamoDB 비교

SQL에서 DELETE 문은 테이블에서 하나 이상의 행을 제거합니다. Amazon DynamoDB에서는 DeleteItem 작업을 사용하여 항목을 한 번에 하나씩 삭제합니다.

주제

- SQL에서 테이블의 데이터 삭제
- DynamoDB에서 테이블의 데이터 삭제

### SQL에서 테이블의 데이터 삭제

SQL에서는 DELETE 문을 사용하여 하나 이상의 행을 삭제합니다. WHERE 절은 수정하려는 행을 결정합니다. 다음은 예입니다.

```
DELETE FROM Music
WHERE Artist = 'The Acme Band' AND SongTitle = 'Look Out, World';
```

여러 열을 삭제하도록 WHERE 절을 수정할 수 있습니다. 예를 들어, 다음 예제와 같이 특정 아티스트의 모든 노래를 삭제할 수 있습니다.

```
DELETE FROM Music WHERE Artist = 'The Acme Band'
```

# DynamoDB에서 테이블의 데이터 삭제

DynamoDB에서는 클래식 API나 PartiQL(SQL 호환 쿼리 언어)을 사용하여 단일 항목을 삭제할 수 있습니다. 복수의 항목을 수정하려면 복수의 작업을 사용해야 합니다.

## DynamoDB API

DynamoDB API에서는 `DeleteItem` 작업을 사용하여 테이블에서 데이터를 한 번에 한 항목씩 삭제합니다. 항목의 기본 키 값을 지정해야 합니다.

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    }
}
```

> **Note**
>
> Amazon DynamoDB는 `DeleteItem` 외에도 여러 항목을 동시에 삭제하는 `BatchWriteItem` 작업을 지원합니다.

`DeleteItem`은 조건부 쓰기를 지원하는데, 특정 `ConditionExpression`이 true로 평가되는 경우에 한해 작업이 성공합니다. 예를 들어, 다음 `DeleteItem` 작업은 RecordLabel 속성이 있는 항목만 삭제합니다.

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    },
   ConditionExpression: "attribute_exists(RecordLabel)"
}
```

## PartiQL for DynamoDB

PartiQL에서는 `ExecuteStatement` 작업을 통해 `Delete` 문을 사용하여 테이블에서 데이터를 한 번에 한 항목씩 삭제합니다. 항목의 기본 키 값을 지정해야 합니다.

이 테이블의 기본 키는 Artist 및 SongTitle로 구성됩니다. 이 속성들의 값을 지정해야 합니다.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks'
```

작업에 대한 추가 조건을 지정할 수 있습니다. 다음 DELETE 작업은 Awards가 11 이상인 경우에만 항목을 삭제합니다.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks' AND Awards > 11
```

> **Note**
>
> DELETE 및 ExecuteStatement를 사용하는 코드 예제는 DynamoDB의 PartiQL delete 문 섹션을 참조하세요.

## 테이블 제거 시 관계형(SQL) 데이터베이스와 DynamoDB 비교

SQL에서는 DROP TABLE 문을 사용하여 테이블을 제거합니다. Amazon DynamoDB에서는 DeleteTable 작업을 사용합니다.

주제

- SQL에서 테이블 제거
- DynamoDB에서 테이블 제거

### SQL에서 테이블 제거

테이블이 더 이상 필요하지 않아 영구적으로 삭제하려는 경우 SQL에서 DROP TABLE 문을 사용합니다.

```
DROP TABLE Music;
```

테이블이 삭제되고 나면 복구할 수 없습니다. (일부 관계형 데이터베이스에서 DROP TABLE 작업 실행을 취소할 수는 있지만 일부 공급업체에 한정된 기능으로 널리 구현되지는 않습니다.)

### DynamoDB에서 테이블 제거

DynamoDB에는 이와 비슷한 `DeleteTable` 작업이 있습니다. 다음 예에서는 테이블이 영구적으로 삭제됩니다.

```
{
    TableName: "Music"
}
```

# Amazon DynamoDB 학습 리소스 및 도구

DynamoDB를 이해하고 작업하는 데 도움이 되는 다음과 같은 추가 리소스를 사용할 수 있습니다.

주제

- 코딩 및 시각화를 위한 도구
- 권장 가이드 문서
- 지식 센터 문서
- 블로그 게시물, 리포지토리 및 가이드
- 데이터 모델링 및 디자인 패턴 프레젠테이션
- 교육 과정

## 코딩 및 시각화를 위한 도구

DynamoDB 작업에 다음과 같은 코딩 및 시각화 도구를 사용할 수 있습니다.

- Amazon DynamoDB용 NoSQL Workbench - DynamoDB 테이블을 디자인, 생성, 쿼리, 관리하는 데 도움이 되는 통합된 시각적 도구이며, 데이터 모델링, 데이터 시각화 및 쿼리 개발 기능을 제공합니다.
- Dynobase - 간단하게 DynamoDB 테이블을 보고 작업하고, 앱 코드를 작성하고, 실시간 확인을 통해 레코드를 편집할 수 있는 데스크톱 도구입니다.
- DynamoDB Toolbox - 데이터 모델링 작업과 JavaScript 및 Node.js에서의 작업에 유용한 유틸리티를 제공하는 Jeremy Daly의 프로젝트입니다.
- DynamoDB Streams 프로세서 - DynamoDB 스트림 작업에 사용할 수 있는 간단한 도구입니다.

# 권장 가이드 문서

AWS 권장 가이드에서는 프로젝트를 가속화하는 데 도움이 되는 오랜 시간 동안 검증된 전략, 가이드 및 패턴을 제공합니다. 이러한 리소스는 AWS 기술 전문가와 글로벌 AWS 파트너 커뮤니티가 고객의 비즈니스 목표 달성을 지원한 다년간의 경험을 바탕으로 개발했습니다.

데이터 모델링 및 마이그레이션

- DynamoDB의 계층적 데이터 모델
- DynamoDB를 사용한 데이터 모델링
- AWS DMS를 사용하여 DynamoDB로 Oracle 데이터베이스 마이그레이션

글로벌 테이블

- Amazon DynamoDB 글로벌 테이블 사용

서버리스

- AWS Step Functions를 사용하여 서버리스 사가 패턴 구현

SaaS 아키텍처

- 단일 컨트롤 플레인에서 여러 SaaS 제품의 테넌트 관리
- C# 및 AWS CDK를 사용한 사일로 모델을 위한 SaaS 아키텍처의 테넌트 온보딩

데이터 보호 및 데이터 이동

- Amazon DynamoDB에 대한 크로스 계정 액세스 구성
- DynamoDB의 전체 테이블 복사 옵션
- AWS의 데이터베이스에 대한 재해 복구 전략

기타사항

- DynamoDB에서 태깅 적용 지원

권장 가이드 비디오 둘러보기

- 서버리스 아키텍처를 사용한 데이터 파이프라인 생성
- Novartis - 구매 엔진: AI 기반 조달 포털
- Veritiv: AWS 데이터 레이크에서 인사이트를 활용하여 판매 수요 예측
- mimik: AWS를 활용하는 하이브리드 엣지 클라우드로 엣지 마이크로서비스 메시 지원
- Amazon DynamoDB를 사용하여 변경 데이터 캡처

DynamoDB에 대한 추가 권장 가이드 문서 및 비디오는 권장 가이드를 참조하세요.

## 지식 센터 문서

AWS 지식 센터 문서 및 비디오에는 AWS 고객으로부터 가장 자주 받는 질문과 요청이 수록되어 있습니다. 다음은 DynamoDB와 관련된 특정 작업에 대한 최신 지식 센터 문서입니다.

### 비용 최적화

- Amazon DynamoDB로 비용을 최적화하려면 어떻게 해야 하나요?

### 제한 및 지연 시간

- Amazon DynamoDB 테이블에서 지연 시간이 긴 문제를 해결하려면 어떻게 해야 하나요?
- DynamoDB 테이블이 제한되는 이유는 무엇인가요?
- 온디맨드 DynamoDB 테이블이 제한되는 이유는 무엇인가요?

### 페이지 매김

- DynamoDB에서 페이지 매김을 구현하려면 어떻게 해야 하나요?

### 트랜잭션

- DynamoDB에서 `TransactWriteItems` API 호출이 실패하는 이유는 무엇인가요?

### 문제 해결

- DynamoDB Auto Scaling 문제를 해결하려면 어떻게 해야 하나요?
- DynamoDB에서 HTTP 4XX 오류를 해결하려면 어떻게 해야 하나요?

DynamoDB에 대한 추가 문서 및 비디오는 지식 센터 문서를 참조하세요.

## 블로그 게시물, 리포지토리 및 가이드

DynamoDB 개발자 안내서 외에도 DynamoDB를 사용하는 데 유용한 리소스가 많이 있습니다. 다음은 DynamoDB 작업에 필요한 몇 가지 엄선된 블로그 게시물, 리포지토리 및 가이드입니다.

- 다양한 AWS SDK 언어(Node.js, Java, Python .NET, Go,, Rust)로 지원되는 AWS의 DynamoDB 코드 예제 리포지토리입니다.
- DynamoDB 북 - Alex DeBrie의 종합 가이드로, DynamoDB를 사용한 데이터 모델링에 대한 전략 중심의 접근 방식을 소개합니다.
- DynamoDB 가이드 - Alex DeBrie의 오픈 가이드로, DynamoDB NoSQL 데이터베이스의 기본 개념과 고급 기능을 살펴봅니다.
- 20가지 간단한 단계를 통해 RDBMS에서 DynamoDB로 전환하는 방법 - Jeremy Daly가 제공하는 데이터 모델링 학습에 유용한 단계 목록입니다.
- DynamoDB JavaScript DocumentClient 치트 시트 - Node.js 또는 JavaScript 환경에서 DynamoDB를 사용하여 애플리케이션 빌드를 시작하는 데 도움이 되는 치트 시트입니다.
- DynamoDB 핵심 개념 비디오 - 이 재생 목록은 DynamoDB의 여러 핵심 개념을 다룹니다.

## 데이터 모델링 및 디자인 패턴 프레젠테이션

DynamoDB를 최대한 효과적으로 활용하는 데 도움이 되는 데이터 모델링 및 디자인 패턴에 대한 다음 리소스를 사용할 수 있습니다.

- AWS re:Invent 2019: DynamoDB를 활용한 데이터 모델링
  - DynamoDB 데이터 모델링의 원칙을 안내하는 Alex DeBrie의 강연입니다.
- AWS re:Invent 2020: DynamoDB를 활용한 데이터 모델링 - 1부
- AWS re:Invent 2020: DynamoDB를 활용한 데이터 모델링 - 2부
- AWS re:Invent 2017: 고급 디자인 패턴
- AWS re:Invent 2018: 고급 디자인 패턴
- AWS re:Invent 2019: 고급 디자인 패턴
  - 이 세션에서 Jeremy Daly가 자신의 12가지 핵심 고려 사항을 공유합니다.
- AWS re:Invent 2020: DynamoDB 고급 디자인 패턴 - 1부
- AWS re:Invent 2020: DynamoDB 고급 디자인 패턴 - 2부

- Twitch의 DynamoDB 오피스 아워

> **Note**
>
> 각 세션에서는 다양한 사용 사례와 예제를 다룹니다.

## 교육 과정

DynamoDB에 대해 자세히 알아볼 수 있는 다양한 교육 과정과 교육 옵션이 있습니다. 다음은 몇 가지 최신 예입니다.

- Amazon DynamoDB를 활용한 개발 - AWS에서 설계한 과정으로, Amazon DynamoDB용 데이터 모델링을 통해 실제 애플리케이션을 개발하는 초보자에서 전문가까지 안내해드립니다.
- DynamoDB 심층 분석 과정 - Pluralsight의 교육 과정입니다.
- Amazon DynamoDB: NoSQL 데이터베이스 중심 애플리케이션 빌드 - edX에서 호스팅되는 AWS 교육 및 자격증 팀의 교육 과정입니다.

# DynamoDB 읽기 및 쓰기

DynamoDB 읽기 및 쓰기는 테이블에서 데이터를 검색(읽기)하고 테이블에서 데이터를 삽입, 업데이트 또는 삭제(쓰기)하는 작업을 말합니다. DynamoDB를 사용할 때는 읽기 및 쓰기의 개념을 이해하는 것이 중요합니다. 읽기와 쓰기는 애플리케이션의 성능과 비용에 직접적인 영향을 미치기 때문입니다.

이 주제에서는 DynamoDB에 적용되는 다양한 유형의 읽기 일관성에 대한 자세한 정보를 제공합니다. 또한 이 주제에서는 수행할 수 있는 다양한 읽기 및 쓰기 작업의 단위 소비량에 대해서도 설명합니다.

주제


- DynamoDB 읽기 일관성
- DynamoDB 읽기 및 쓰기 작업


# DynamoDB 읽기 일관성

Amazon DynamoDB는 테이블, 로컬 보조 인덱스(LSI), 글로벌 보조 인덱스(GSI), 스트림에서 데이터를 읽습니다. 자세한 내용은 Amazon DynamoDB의 핵심 구성 요소 섹션을 참조하세요. 테이블과 LSI 모두 최종 읽기 일관성(기본값)과 강력히 일관된 읽기라는 두 가지 읽기 일관성 옵션을 제공합니다. GSI와 스트림에서의 모든 읽기는 최종적으로 일관됩니다.

애플리케이션이 DynamoDB 테이블에 데이터를 쓰고 HTTP 200 응답(OK)을 받으면 쓰기가 성공적으로 완료되고 지속적으로 유지된 것입니다. DynamoDB는 읽기 커밋됨 격리를 제공하고 읽기 작업이 항상 항목에 대해 커밋된 값을 반환하도록 합니다. 읽기는 결국 성공하지 못한 쓰기의 항목을 절대 보여주지 않습니다. 읽기 커밋됨 격리는 읽기 작업 직후에 항목에 대한 수정을 막지 않습니다.

## 최종적으로 일관된 읽기

최종적 일관성은 모든 읽기 작업의 기본 읽기 일관성 모델입니다. DynamoDB 테이블이나 인덱스에 최종 읽기 일관성을 실행할 때 최근에 완료된 쓰기 작업의 결과가 응답에 반영되지 않을 수 있습니다. 잠시 후 읽기 요청을 반복하면 응답이 결국 보다 최근 항목을 반환합니다. 최종 읽기 일관성은 테이블, 로컬 보조 인덱스, 글로벌 보조 인덱스에서 지원됩니다. 또한 DynamoDB 스트림에서의 모든 읽기도 최종적으로 일관됩니다.

최종 읽기 일관성의 비용은 강력히 일관된 읽기 비용의 절반입니다. 자세한 내용은 Amazon DynamoDB 요금을 참조하세요.

## 강력하게 일관된 읽기

`GetItem`, `Query`, `Scan` 같은 읽기 작업은 선택적 `ConsistentRead` 파라미터를 제공합니다. `ConsistentRead`를 true로 설정하면 DynamoDB는 성공한 모든 이전 쓰기 작업의 업데이트가 반영된 최신 데이터가 포함된 응답을 반환합니다. 강력히 일관된 읽기는 테이블과 로컬 보조 인덱스에서만 지원됩니다. 글로벌 보조 인덱스 또는 DynamoDB 스트림에서의 강력히 일관된 읽기는 지원되지 않습니다.

## 글로벌 테이블 읽기 일관성

DynamoDB는 다중 활성 및 다중 리전 복제를 위한 글로벌 테이블도 지원합니다. 글로벌 테이블은 서로 다른 AWS 리전에 있는 여러 복제본 테이블로 구성됩니다. 복제본 테이블의 항목이 변경되면 동일한 글로벌 테이블에 있는 다른 모든 복제본에 변경 내용이 복제됩니다. 복제는 일반적으로 1초 내에 이루어지며 최종 일관성을 갖습니다. 자세한 내용은 일관성 모델 섹션을 참조하세요.

# DynamoDB 읽기 및 쓰기 작업

DynamoDB 읽기 작업을 사용하면 파티션 키 값 및 선택적으로 정렬 키 값을 지정하여 테이블에서 하나 이상의 항목을 검색할 수 있습니다. DynamoDB 쓰기 작업을 사용하면 테이블에서 항목을 삽입, 업데이트 또는 삭제할 수 있습니다. 이 주제에서는 이 두 작업의 용량 단위 소비량에 대해 설명합니다.

주제

- 읽기 작업의 용량 단위 소비량
- 쓰기 작업의 용량 단위 소비량

## 읽기 작업의 용량 단위 소비량

DynamoDB 읽기 요청은 강력히 일관된 읽기, 최종 읽기 일관성 또는 트랜잭션 읽기 요청일 수 있습니다.

- 최대 4KB 항목의 강력히 일관된 읽기 요청에는 하나의 읽기 단위가 필요합니다.
- 최대 4KB 항목의 최종적으로 일관된 읽기 요청에는 절반의 읽기 단위가 필요합니다.
- 최대 4KB 항목의 트랜잭션 읽기 요청에는 2개의 읽기 단위가 필요합니다.

DynamoDB 읽기 일관성 모델에 대한 자세한 내용은 DynamoDB 읽기 일관성 단원을 참조하세요.

읽기 항목 크기는 다음 4KB 배수로 반올림됩니다. 예를 들어, 3,500바이트의 항목을 읽으려면 4KB 항목을 읽을 때와 동일한 처리량이 소비됩니다.

4KB보다 큰 항목을 읽어야 하는 경우, DynamoDB에 추가 읽기 단위가 필요합니다. 필요한 총 읽기 단위 수는 항목 크기 및 최종적으로 일관된 읽기와 강력히 일관된 읽기 중 어느 것을 원하는지에 따라 달라집니다. 예를 들어 항목 크기가 8KB인 경우 강력히 일관된 읽기 1회를 유지하려면 읽기 단위 2개가 필요합니다. 최종적으로 일관된 읽기를 선택하는 경우 읽기 단위 1개가 필요하고 트랜잭션 읽기 요청의 경우 읽기 단위 4개가 필요합니다.

다음 목록은 DynamoDB 읽기 작업이 어떻게 읽기 단위를 소비하는지에 대한 설명입니다.

- GetItem: 테이블에서 단일 항목을 읽습니다. `GetItem`에 사용될 읽기 단위 수를 파악하려면 항목 크기를 가져와서 다음 4KB 배수로 반올림합니다. 강력히 일관된 읽기를 지정한 경우, 이 값이 필요한 읽기 단위 수가 됩니다. 최종적으로 일관된 읽기(기본값)의 경우, 이 숫자를 2로 나눕니다.

  예를 들어, 크기가 3.5KB인 항목 하나를 읽는 경우 DynamoDB가 항목 크기를 4KB로 올림합니다. 그리고 크기가 10KB인 항목 하나를 읽는다면 DynamoDB가 항목 크기를 12KB로 올림합니다.
- BatchGetItem: 하나 이상의 테이블에서 최대 100개의 항목을 읽습니다. DynamoDB는 배치의 각 항목을 개별 `GetItem` 요청으로 처리합니다. DynamoDB는 먼저 각 항목의 크기를 다음 4KB 배수로 반올림한 후 전체 크기를 계산합니다. 계산 결과는 모든 항목의 전체 크기와 반드시 동일하지는 않습니다. 예를 들어 `BatchGetItem`이 1.5KB와 6.5KB 크기의 두 항목을 읽는 경우 DynamoDB는 크기를 12KB(4KB + 8KB)로 계산합니다. DynamoDB는 크기를 8KB(1.5KB+6.5KB)로 계산하지 않습니다.
- Query: 파티션 키 값이 동일한 여러 항목을 읽습니다. 반환되는 모든 항목이 단일 읽기 작업으로 처리됩니다. 이 경우 DynamoDB는 모든 항목의 총 크기를 계산합니다. 그런 다음 DynamoDB는 해당 크기를 다음 4KB 배수로 반올림합니다. 예를 들어, 쿼리가 10개 항목을 반환하여 전체 크기가 40.8KB라고 가정하겠습니다. 이 경우 DynamoDB는 작업 항목 크기를 44KB로 올림합니다. 그리고, 쿼리가 각각 64byte인 항목 1,500개를 반환한다면 누적 크기는 96KB가 됩니다.
- Scan: 테이블의 모든 항목을 읽습니다. DynamoDB는 스캔으로 반환되는 항목 크기가 아닌 평가할 항목 크기를 계산합니다. Scan 작업에 대한 자세한 설명은 DynamoDB에서 테이블 스캔 섹션을 참조하세요.

> **Important**
>
> 존재하지 않는 항목에 대해 읽기 작업을 수행하는 경우 DynamoDB는 위에 설명된 대로 읽기 처리량을 계속 사용합니다. `Query`/`Scan` 작업의 경우 데이터가 없더라도 읽기 일관성 및 요청을 처리하기 위해 검색된 파티션 수에 따라 추가 읽기 처리량에 대한 요금이 계속 부과됩니다.

항목을 반환하는 모든 작업에서 검색할 속성의 하위 집합을 요청할 수 있습니다. 그러나 이렇게 해도 항목 크기 계산에 영향이 미치지 않습니다. 또한 `Query`나 `Scan` 작업은 속성 값이 아닌 항목 수를 반환하기도 합니다. 항목 수를 가져올 때도 사용되는 읽기 단위의 수는 동일하며, 항목 크기 계산도 똑같이 적용됩니다. 이는 DynamoDB가 수를 늘리기 위해 각 항목을 읽어야 하기 때문입니다.

## 쓰기 작업의 용량 단위 소비량

쓰기 단위 1은 최대 1KB 크기의 항목에 대해 1회 쓰기를 나타냅니다. 1KB보다 큰 항목을 써야 하는 경우, DynamoDB가 추가 쓰기 단위를 사용해야 합니다. 트랜잭션 쓰기 요청은 최대 1KB 크기 항목의 1회 쓰기를 수행하는 데 2개의 쓰기 단위가 필요합니다. 필요한 총 쓰기 요청 단위 수는 항목 크기에 따라 결정됩니다. 예를 들어 항목 크기가 2KB인 경우 쓰기 요청 1회를 지속하는 데 쓰기 단위 2개가 필요하고, 트랜잭션 쓰기 요청의 경우 쓰기 단위 4개가 필요합니다.

쓰기 항목 크기는 다음 1KB 배수로 반올림됩니다. 예를 들어, 500바이트의 항목을 쓰려면 1KB 항목을 쓸 때와 동일한 처리량이 소비됩니다.

다음 목록은 DynamoDB 쓰기 작업이 어떻게 쓰기 단위를 소비하는지에 대한 설명입니다.

- PutItem: 테이블에 단일 항목을 씁니다. 이때 기본 키가 동일한 항목이 테이블에 존재하면 이 항목까지 바뀝니다. 프로비저닝된 처리량 소비를 계산할 경우 둘 중 항목 크기가 더 큰 것이 적용됩니다.
- UpdateItem: 테이블에서 단일 항목을 수정합니다. DynamoDB는 업데이트 전후에 표시되는 항목의 크기를 고려합니다. 이때 할당 처리량을 소비하려면 크기가 더 큰 항목이 반영됩니다. 항목 속성의 하위 집합을 업데이트하더라도 `UpdateItem`은 할당 처리량('사전' 항목 크기와 '사후' 항목 크기 중 더 큰 것)을 모두 소비합니다.
- DeleteItem: 테이블에서 단일 항목을 제거합니다. 할당된 처리량 소비는 삭제된 항목의 크기를 기준으로 합니다.
- BatchWriteItem: 하나 이상의 테이블에 최대 25개의 항목을 씁니다. DynamoDB는 배치의 각 항목을 개별 `PutItem` 또는 `DeleteItem` 요청으로 처리합니다(업데이트는 지원되지 않음). DynamoDB가 먼저 각 항목의 크기를 다음 1KB 배수로 반올림한 후 전체 크기를 계산합니다. 계산 결과는 모든 항목의 전체 크기와 반드시 동일하지는 않습니다. 예를 들어 `BatchWriteItem`이 500바이트와 3.5KB

크기의 두 항목을 쓰는 경우 DynamoDB는 크기를 5KB(1KB + 4KB)로 계산합니다. DynamoDB는 크기를 4KB(500바이트 + 3.5KB)로 계산하지 않습니다.

`PutItem`, `UpdateItem` 및 `DeleteItem` 작업의 경우 DynamoDB는 항목 크기를 다음 1KB로 올림합니다. 예를 들어 크기가 1.6KB인 항목 하나를 업로드하거나 삭제한다면 DynamoDB가 항목 크기를 2KB로 반올림합니다.

`PutItem`, `UpdateItem` 및 `DeleteItem` 작업은 대부분 조건부 쓰기가 가능합니다. 이 경우 사용자가 지정한 표현식이 true로 평가되어야만 작업이 성공적으로 완료됩니다. 표현식이 false로 평가되더라도 DynamoDB는 계속 테이블에서 쓰기 용량 단위를 소비합니다. 사용된 쓰기 용량 단위 수는 항목 크기에 따라 결정됩니다. 이 항목은 테이블의 기존 항목이거나 생성 또는 업데이트하려는 새 항목일 수 있습니다. 예를 들어 기존 항목의 크기가 300KB입니다. 생성 또는 업데이트하려는 새 항목은 310KB입니다. 이 경우 새 항목에 소비되는 쓰기 용량 단위는 310KB입니다.

# DynamoDB 처리량 용량

이 섹션에서는 DynamoDB 테이블에 사용할 수 있는 두 가지 처리량 모드를 간략히 살펴보고 애플리케이션에 적절한 용량 모드를 선택하기 위해 고려할 사항을 알아봅니다. 테이블의 처리량 모드에 따라 테이블 용량이 관리되는 방식이 결정됩니다. 처리량 모드는 테이블의 읽기 및 쓰기 작업에 대한 요금이 부과되는 방식도 결정합니다. Amazon DynamoDB에서는 다양한 워크로드 요구 사항에 맞춰 테이블에 대해 온디맨드 모드 또는 프로비저닝된 모드를 선택할 수 있습니다.

주제


- 온디맨드 모드
- 프로비저닝된 모드
- DynamoDB 온디맨드 용량 모드
- DynamoDB 프로비저닝된 용량 모드
- DynamoDB 웜 처리량 이해
- DynamoDB 버스트 및 조정 용량
- DynamoDB에서 용량 모드 전환 시 고려 사항


## 온디맨드 모드

Amazon DynamoDB 온디맨드 모드는 데이터베이스 관리를 간소화하고 고객의 가장 까다로운 애플리케이션을 지원하도록 자동으로 규모를 조정하는 서버리스 처리량 옵션입니다. DynamoDB 온디맨드를 사용하면 용량 계획, 사용량 모니터링 및 규모 조정 정책 구성에 대한 걱정 없이 테이블을 만들 수 있습니다. DynamoDB 온디맨드는 읽기 및 쓰기 요청에 대해 요청당 지불 요금이 적용되므로 사용하는 만큼에 대해서만 비용을 지불하면 됩니다. 온디맨드 모드 테이블의 경우 애플리케이션에서 수행할 것으로 예상되는 읽기 및 쓰기 처리량을 지정할 필요가 없습니다.

온디맨드 모드는 대부분의 DynamoDB 워크로드에 권장되는 기본 처리량 옵션입니다. DynamoDB는 처리량 관리 및 규모 조정의 모든 측면을 처리하여 소규모로 시작하고 초당 수백만 개의 요청으로 규모 조정할 수 있는 워크로드를 지원합니다. 테이블의 처리량 용량을 관리하지 않고도 필요에 따라 DynamoDB 테이블을 읽고 쓸 수 있습니다. 자세한 내용은 DynamoDB 온디맨드 용량 모드 섹션을 참조하세요.

# 프로비저닝된 모드

프로비저닝된 모드에서는 애플리케이션에 필요한 초당 읽기 및 쓰기 횟수를 지정해야 합니다. 실제로 사용한 프로비저닝 용량이 아닌 프로비저닝한 시간당 읽기 및 쓰기 용량을 기준으로 요금이 청구됩니다. 그러면 DynamoDB 사용을 정의된 요청 속도 이하로 유지하도록 관리하여 비용을 예측하는 데 도움이 됩니다.

예측 가능한 증가를 보이는 안정적인 워크로드가 있고 애플리케이션의 용량 요구 사항을 안정적으로 예측할 수 있는 경우 프로비저닝된 용량을 사용하도록 선택할 수 있습니다. 자세한 내용은 DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요.

# DynamoDB 온디맨드 용량 모드

Amazon DynamoDB 온디맨드는 용량 계획 없이 가장 까다로운 워크로드를 수용하도록 자동으로 규모 조정되는 진정한 서버리스 데이터베이스 환경을 제공합니다. 온디맨드는 설정 프로세스를 간소화하고 용량 관리 및 모니터링이 필요 없게 만들며 빠른 자동 규모 조정을 제공합니다. 요청당 지불 요금을 사용하면 실제로 사용하는 처리량에 대해서만 비용을 지불하기 때문에 유휴 용량에 대해 걱정할 필요가 없습니다. 읽기 또는 쓰기 요청당 요금이 청구되므로 실제 사용량이 비용에 직접 반영됩니다.

온디맨드 모드를 선택하면 DynamoDB는 이전에 도달한 트래픽 수준까지 확장 또는 축소할 때 즉시 워크로드를 수용합니다. 워크로드의 트래픽 수준이 새 피크에 도달하면 DynamoDB는 증가된 처리량 요구 사항에 맞게 자동으로 규모 조정됩니다. 온디맨드 모드는 소규모로 시작하여 초당 수백만 개의 요청으로 규모 조정할 수 있는 현대적 서버리스 애플리케이션 구축을 간소화하기 때문에 권장되는 기본 처리량 옵션입니다. 온디맨드 테이블이 스케일 아웃되면 스로틀링 없이 향후 동일한 처리량을 즉시 달성할 수 있습니다. 테이블에서 트래픽이 발생하지 않는 경우 온디맨드를 사용하면 처리량에 대한 요금이 부과되지 않습니다. 온디맨드 모드의 규모 조정 속성에 대한 자세한 내용은 초기 처리량 및 규모 조정 속성 섹션을 참조하세요.

온디맨드 모드를 사용하는 테이블은 DynamoDB 프로비저닝 모드가 제공하는 것과 동일한 한 자릿수 밀리초 지연 시간, 서비스 수준 계약(SLA) 및 보안을 제공합니다.

> **Note**
>
> 기본적으로 DynamoDB는 의도하지 않은 런어웨이 사용으로부터 사용자를 보호합니다. 계정의 모든 테이블에 대해 40,000개의 테이블 수준 읽기 및 쓰기 처리량 제한을 초과하여 규모를 조정하려면 이 할당량 증가를 요청할 수 있습니다. 기본 테이블 처리량 할당량을 초과하는 처리량 요청은 스로틀링됩니다. 자세한 내용은 처리량 기본 할당량 섹션을 참조하세요.

선택적으로, 개별 온디맨드 테이블 및 글로벌 보조 인덱스의 초당 최대 읽기 또는 쓰기(또는 둘 다) 처리량을 구성할 수도 있습니다. 처리량을 구성하면 예측 가능한 비용 관리를 위해 테이블 수준의 사용량과 비용을 제한하고, 리소스 소비가 의도치 않게 급증하는 것을 방지하며, 과도한 사용을 예방할 수 있습니다. 최대 테이블 처리량을 초과하는 처리량 요청은 제한됩니다. 애플리케이션 요구 사항에 따라 언제든지 테이블별 최대 처리량을 수정할 수 있습니다. 자세한 내용은 온디맨드 테이블의 최대 처리량 섹션을 참조하세요.

시작하려면 온디맨드 모드를 사용하도록 테이블을 생성하거나 업데이트합니다. 자세한 내용은 DynamoDB 테이블에 대한 기본 작업 섹션을 참조하세요.

테이블은 언제든지 온디맨드 모드에서 프로비저닝된 용량 모드로 전환할 수 있습니다. 용량 모드 간에 여러 번 전환하는 경우 다음 조건이 적용됩니다.

- 온디맨드 모드에서 새로 생성된 테이블은 언제든지 프로비저닝된 용량 모드로 전환할 수 있습니다. 하지만 테이블 생성 타임스탬프 이후 24시간이 지난 뒤에야 온디맨드 모드로 다시 전환할 수 있습니다.
- 온디맨드 모드의 기존 테이블은 언제든지 프로비저닝된 용량 모드로 전환할 수 있습니다. 하지만 온디맨드로의 전환을 나타내는 마지막 타임스탬프가 발생한 지 24시간이 지난 후에야 다시 온디맨드 모드로 전환할 수 있습니다.

읽기 및 쓰기 용량 모드 간 전환에 대한 자세한 내용은 DynamoDB에서 용량 모드 전환 시 고려 사항 섹션을 참조하세요. 온디맨드 테이블 할당량에 대해서는 읽기/쓰기 처리량 섹션을 참조하세요.

주제

- 읽기 요청 단위 및 쓰기 요청 단위
- 초기 처리량 및 규모 조정 속성
- 온디맨드 테이블의 최대 처리량

## 읽기 요청 단위 및 쓰기 요청 단위

DynamoDB에서는 읽기 요청 단위 및 쓰기 요청 단위의 측면에서 애플리케이션이 테이블에서 수행하는 읽기 및 쓰기에 대해 요금이 부과됩니다.

읽기 요청 단위 1은 초당 강력히 일관된 읽기 1 또는 초당 최종적으로 일관된 읽기 2(최대 4KB 크기 항목의 경우)를 나타냅니다. DynamoDB 읽기 일관성 모델에 대한 자세한 내용은 DynamoDB 읽기 일관성 섹션을 참조하세요.

쓰기 요청 단위 1은 최대 1KB 크기의 항목에 대해 초당 1회 쓰기 작업을 나타냅니다.

읽기 및 쓰기 단위가 소비되는 방식에 대한 자세한 내용은 DynamoDB 읽기 및 쓰기 작업 섹션을 참조하세요.

## 초기 처리량 및 규모 조정 속성

온디맨드 용량 모드를 사용하는 DynamoDB 테이블은 애플리케이션의 트래픽 볼륨에 따라 자동으로 조정됩니다. 새로운 온디맨드 테이블은 초당 최대 4,000회 쓰기 및 초당 최대 12,000회 읽기를 지속할 수 있습니다. 온디맨드 용량 모드의 테이블은 이전 피크 트래픽의 최대 2배 용량을 즉시 수용합니다. 예를 들어 애플리케이션의 트래픽 패턴이 초당 25,000~50,000회의 강력히 일관된 읽기 사이에서 다양하다고 가정합니다. 이때 초당 50,000회 읽기는 이전에 도달한 트래픽 피크입니다. 온디맨드 용량 모드는 초당 최대 100,000회 읽기의 지속적인 트래픽을 즉시 수용할 수 있습니다. 애플리케이션이 초당 100,000회 읽기 트래픽을 지속하는 경우 해당 피크가 새로운 이전 피크가 되어 후속 트래픽은 초당 최대 200,000회 읽기에 도달할 수 있습니다.

워크로드가 테이블에서 이전 피크의 2배 이상을 생성하는 경우 DynamoDB가 트래픽 볼륨 증가에 따라 자동으로 추가 용량을 할당합니다. 이러한 용량 할당은 워크로드에 제한이 발생하지 않도록 하는 데 도움이 됩니다. 하지만 30분 이내에 이전 피크의 2배 용량을 초과할 경우 조절이 발생할 수 있습니다. 예를 들어 애플리케이션의 트래픽 패턴이 초당 25,000~50,000회의 강력히 일관된 읽기 사이에서 다양하다고 가정합니다. 이때 초당 50,000회 읽기는 이전에 도달한 트래픽 피크입니다. 초당 100,000회 이상의 읽기가 발생하기 전에 테이블을 사전 워밍하거나 최소 30분 이상의 트래픽 증가 간격을 두는 것이 좋습니다. 사전 워밍에 대한 자세한 내용은 DynamoDB 웜 처리량 이해 섹션을 참조하세요.

DynamoDB는 워크로드의 최대 트래픽이 이전 최고치의 2배 이내로 유지되는 경우 30분 조절 제한을 두지 않습니다. 최대 트래픽이 최고치의 2배를 초과하는 경우, 마지막 최고치에 도달한 지 30분 후에 이러한 증가가 발생하는지 확인하세요.

## 온디맨드 테이블의 최대 처리량

온디맨드 테이블의 경우, 선택적으로 개별 테이블 및 연결된 글로벌 보조 인덱스(GSI)의 초당 최대 읽기 또는 쓰기(또는 둘 다) 처리량을 지정할 수도 있습니다. 최대 온디맨드 처리량을 지정하면 테이블 수준의 사용량과 비용을 제한하는 데 도움이 됩니다. 기본적으로 최대 처리량 설정은 적용되지 않으며 온디맨드 처리량은 계정 내 모든 테이블의 40,000 테이블 수준 읽기 및 쓰기 처리량 AWS 서비스 할당량에 따라 제한됩니다. 필요한 경우 서비스 할당량 증가를 요청할 수 있습니다.

온디맨드 테이블의 최대 처리량을 구성하면 지정된 최대 처리량을 초과하는 처리량 요청은 제한됩니다. 애플리케이션 요구 사항에 따라 언제든지 테이블 수준의 처리량 설정을 수정할 수 있습니다.

다음은 온디맨드 테이블의 최대 처리량을 사용할 때 도움이 될 수 있는 몇 가지 일반적인 사용 사례입니다.

- 처리량 비용 최적화 - 온디맨드 테이블에 최대 처리량을 사용하면 비용 예측성과 관리 용이성이 향상됩니다. 또한 온디맨드 모드로 트래픽 패턴과 예산이 서로 다른 워크로드를 지원할 수 있는 뛰어난 유연성을 누리게 됩니다.
- 과도한 사용 방지 - 최대 처리량을 설정하면 온디맨드 테이블에서 최적화되지 않은 코드나 악성 프로세스로 인해 발생할 수 있는 우발적인 읽기 또는 쓰기 사용량 급증을 방지할 수 있습니다. 이 테이블 수준의 설정은 조직이 특정 기간 내에 리소스를 과도하게 소비하지 않도록 보호할 수 있습니다.
- 다운스트림 서비스 보호 - 고객 애플리케이션에는 서버리스 및 비서버리스 기술이 포함될 수 있습니다. 서버리스 아키텍처 부분은 수요에 맞춰 빠르게 규모를 조정할 수 있습니다. 하지만 용량이 고정된 다운스트림 구성 요소에는 과부하가 걸릴 수 있습니다. 온디맨드 테이블에 최대 처리량 설정을 구현하면 대규모 이벤트가 여러 다운스트림 구성 요소로 전파되어 예상치 못한 부작용을 초래하는 것을 방지할 수 있습니다.

신규 및 기존 단일 리전 테이블, 글로벌 테이블 및 GSI에 대해 온디맨드 모드의 최대 처리량을 구성할 수 있습니다. Amazon S3 워크플로에서 테이블 복원 및 데이터 가져오기 중에 최대 처리량을 구성할 수도 있습니다.

DynamoDB 콘솔, AWS CLI, AWS CloudFormation 또는 DynamoDB API를 사용하여 온디맨드 테이블의 최대 처리량 설정을 지정할 수 있습니다.

> **Note**
>
> 온디맨드 테이블의 최대 처리량은 최선의 방식으로 적용되며 보장된 요청 최대치가 아닌 목표로 생각해야 합니다. 버스트 용량 때문에 워크로드가 일시적으로 지정된 최대 처리량을 초과할 수 있습니다. DynamoDB는 테이블의 최대 처리량 설정을 초과하는 읽기 또는 쓰기를 수용하기 위해 경우에 따라 버스트 용량을 사용하기도 합니다. 원래는 조절되어야 하는 예상치 못한 읽기 또는 쓰기 요청도 버스트 용량으로 해결할 수 있습니다.

주제

- 온디맨드 모드에서 최대 처리량을 사용할 때 고려할 사항
- 요청 제한 및 CloudWatch 지표

## 온디맨드 모드에서 최대 처리량을 사용할 때 고려할 사항

온디맨드 모드에서 테이블의 최대 처리량을 사용하는 경우 다음 고려 사항이 적용됩니다.

- 온디맨드 테이블 또는 해당 테이블 내의 개별 글로벌 보조 인덱스에 대한 최대 읽기 및 쓰기 처리량을 독립적으로 설정하여 특정 요구 사항에 따라 접근 방식을 세밀하게 조정할 수 있습니다.
- Amazon CloudWatch를 사용하여 DynamoDB 테이블 수준 사용 지표를 모니터링 및 파악하고 온디맨드 모드에 적합한 최대 처리량 설정을 결정할 수 있습니다. 자세한 내용은 DynamoDB 지표 및 차원 섹션을 참조하세요.
- 하나의 글로벌 테이블 복제본에서 최대 읽기 또는 쓰기(또는 둘 다) 처리량 설정을 지정하면 모든 복제본 테이블에 동일한 최대 처리량 설정이 자동으로 적용됩니다. 글로벌 테이블의 복제본 테이블과 보조 인덱스의 쓰기 처리량을 동일하게 설정해 데이터를 적절히 복제하는 것이 중요합니다. 자세한 내용은 글로벌 테이블에 대한 모범 사례 섹션을 참조하세요.
- 지정할 수 있는 가장 작은 최대 읽기 또는 쓰기 처리량은 초당 요청 단위 1개입니다.
- 지정하는 최대 처리량은 온디맨드 테이블 또는 해당 테이블 내 개별 글로벌 보조 인덱스에 사용할 수 있는 기본 처리량 할당량보다 낮아야 합니다.

### 요청 제한 및 CloudWatch 지표

애플리케이션이 온디맨드 테이블에 설정한 최대 읽기 또는 쓰기 처리량을 초과하는 경우 DynamoDB는 해당 요청을 제한하기 시작합니다. DynamoDB는 읽기 또는 쓰기를 제한할 때 호출자에게 `ThrottlingException`을 반환합니다. 그런 다음 필요한 경우 적절한 조치를 취할 수 있습니다. 예를 들어, 최대 테이블 처리량 설정을 늘리거나 비활성화하거나 잠시 기다린 후 요청을 다시 시도할 수 있습니다.

테이블 또는 글로벌 보조 인덱스에 구성된 최대 처리량 모니터링을 단순화하기 위해 CloudWatch는 다음과 같은 지표를 제공합니다. OnDemandMaxReadRequestUnits 및 OnDemandMaxWriteRequestUnits

## DynamoDB 프로비저닝된 용량 모드

DynamoDB에서 프로비저닝된 테이블을 새로 생성할 때는 프로비저닝된 처리량 용량을 지정해야 합니다. 이는 테이블에서 지원할 수 있는 읽기 및 쓰기 처리량의 양입니다. 실제로 사용한 프로비저닝 용량이 아닌 프로비저닝한 시간당 읽기 및 쓰기 용량을 기준으로 요금이 청구됩니다.

애플리케이션의 데이터 및 액세스 요구 사항이 변화함에 따라 테이블의 처리량 설정을 조정해야 합니다. Auto Scaling을 사용하여 트래픽 변경에 따라 테이블의 프로비저닝된 용량을 자동으로 조정할 수 있습니다. DynamoDB 오토 스케일링은 애플리케이션 오토 스케일링의 규모 조정 정책을 사용합니다. DynamoDB에서 오토 스케일링을 구성하려면 대상 사용률 외에도 읽기 및 쓰기 용량의 최소 및 최대 수준을 설정합니다. Application Auto Scaling은 지표가 대상에서 벗어날 경우 조정 이벤트를 트리거하는

CloudWatch 경보를 생성하고 관리합니다. 오토 스케일링은 테이블의 활동을 모니터링하고 사전 구성된 임곗값에 따라 용량 설정을 위 또는 아래로 조정합니다. Auto Scaling은 소비된 용량이 구성된 목표 사용률을 2분 연속으로 초과할 때 트리거됩니다. CloudWatch 경보가 Auto Scaling을 트리거하기 전에 최대 몇 분의 짧은 지연이 있을 수 있습니다. 자세한 내용은 DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리 섹션을 참조하세요.

DynamoDB Auto Scaling을 사용하는 경우 실제 워크로드에 맞게 처리량 설정이 자동으로 조정됩니다. 또한 UpdateTable 작업을 사용하여 테이블의 처리량 용량을 수동으로 조정할 수도 있습니다. 예를 들어 기존 데이터 스토어에서 새 DynamoDB 테이블로 데이터를 대량 로드해야 할 경우 이러한 방식을 선택할 수 있습니다. 쓰기 처리량 설정을 크게 하여 테이블을 생성한 다음, 데이터 대량 로드가 완료되면 이 설정을 줄여도 됩니다.

> **Note**
>
> 기본적으로 DynamoDB는 의도하지 않은 런어웨이 사용으로부터 사용자를 보호합니다. 계정의 모든 테이블에 대해 40,000개의 테이블 수준 읽기 및 쓰기 처리량 제한을 초과하여 규모를 조정하려면 이 할당량 증가를 요청할 수 있습니다. 기본 테이블 처리량 할당량을 초과하는 처리량 요청은 스로틀링됩니다. 자세한 내용은 처리량 기본 할당량 섹션을 참조하세요.

테이블은 언제든지 온디맨드 모드에서 프로비저닝된 용량 모드로 전환할 수 있습니다. 용량 모드 간에 여러 번 전환하는 경우 다음 조건이 적용됩니다.

- 온디맨드 모드에서 새로 생성된 테이블은 언제든지 프로비저닝된 용량 모드로 전환할 수 있습니다. 하지만 테이블 생성 타임스탬프 이후 24시간이 지난 뒤에야 온디맨드 모드로 다시 전환할 수 있습니다.
- 온디맨드 모드의 기존 테이블은 언제든지 프로비저닝된 용량 모드로 전환할 수 있습니다. 하지만 온디맨드로의 전환을 나타내는 마지막 타임스탬프가 발생한 지 24시간이 지난 후에야 다시 온디맨드 모드로 전환할 수 있습니다.

읽기 및 쓰기 용량 모드 간 전환에 대한 자세한 내용은 DynamoDB에서 용량 모드 전환 시 고려 사항 섹션을 참조하세요.

주제

- 읽기 용량 단위 및 쓰기 용량 단위
- 초기 처리량 설정 선택
- DynamoDB Auto Scaling

- DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리
- DynamoDB 예약 용량

## 읽기 용량 단위 및 쓰기 용량 단위

프로비저닝된 모드 테이블의 경우 용량 단위로 처리량 요구 사항을 지정합니다. 이 단위는 애플리케이션이 초당 읽거나 써야 하는 데이터 양을 나타냅니다. 필요하면 나중에 이러한 설정을 수정하거나 DynamoDB Auto Scaling을 활성화하여 자동으로 수정할 수 있습니다.

최대 4KB 크기 항목의 경우 읽기 용량 단위(RCU) 1은 초당 강력히 일관된 읽기 1회 또는 초당 최종적으로 일관된 읽기 2회를 나타냅니다. DynamoDB 읽기 일관성 모델에 대한 자세한 내용은 DynamoDB 읽기 일관성 섹션을 참조하세요.

쓰기 용량 단위(WCU) 1은 최대 1KB 크기의 항목에 대해 초당 1회 쓴다는 의미입니다. 다양한 읽기 및 쓰기 작업에 대한 자세한 내용은 DynamoDB 읽기 및 쓰기 작업 섹션을 참조하세요.

## 초기 처리량 설정 선택

데이터베이스에서 읽기 및 쓰기를 위한 요구 사항은 애플리케이션마다 다릅니다. DynamoDB 테이블의 초기 처리량 설정을 결정할 때는 다음 사항을 고려해야 합니다.

- 예상 읽기 및 쓰기 요청 속도 - 초당 수행해야 하는 읽기 및 쓰기 수를 추정해야 합니다.
- 항목 크기 - 크기가 작아서 용량 단위 하나를 사용하여 읽거나 쓸 수 있는 항목도 있습니다. 그보다 큰 항목에는 용량 단위가 여러 개 필요합니다. 테이블에 들어갈 항목의 평균 크기를 예측하여 해당 테이블의 할당된 처리량을 정확히 설정할 수 있습니다.
- 읽기 일관성 요구 사항 - 읽기 용량 단위의 기준이 되는 강력히 일관된 읽기 작업은 최종적으로 일관된 읽기보다 데이터베이스 리소스를 두 배나 더 많이 소비합니다. 애플리케이션에 필요한 것이 강력한 일관된 읽기인지, 아니면 이러한 요구 사항을 완화하여 그 대신 최종적 일관된 읽기를 수행할 수 있는지 여부를 결정해야 합니다. 기본적으로 DynamoDB의 읽기 작업은 최종적으로 일관된 읽기입니다. 필요한 경우 이러한 작업에 대해 강력히 일관된 읽기를 요청할 수 있습니다.

예를 들어, 테이블에서 초당 80개의 항목을 읽으려고 하며, 항목 크기는 3KB이고, 강력히 일관된 읽기를 수행하려 합니다. 이 시나리오에서, 각 읽기는 프로비저닝된 읽기 용량 단위 1을 요구합니다. 이 수를 구하려면 작업의 항목 크기를 4KB로 나눕니다. 그리고 다음 예와 같이 가장 가까운 정수로 반올림합니다.

- 3KB/4KB = 0.75 또는 읽기 용량 단위 1

따라서 테이블에서 초당 80개 항목을 읽으려면 다음 예와 같이 테이블의 프로비저닝된 읽기 처리량을 읽기 용량 단위 80으로 설정합니다.

- 항목당 읽기 용량 단위 1 × 초당 읽기 80회 = 읽기 용량 단위 80

이번에는 초당 100개의 항목을 테이블에 쓰려 하고 각 항목의 크기는 512바이트인 경우를 예로 들어 보겠습니다. 이 경우, 각 쓰기에는 프로비저닝된 쓰기 용량 단위 1이 필요합니다. 이 수를 구하려면 작업의 항목 크기를 1KB로 나눕니다. 그리고 다음 예와 같이 가장 가까운 정수로 반올림합니다.

- 512바이트/1KB = 0.5 또는 쓰기 용량 단위 1

테이블에 초당 100개 항목을 쓰려면 테이블의 프로비저닝된 쓰기 처리량을 쓰기 용량 단위 100으로 설정합니다.

- 항목당 쓰기 용량 단위 1 × 초당 쓰기 100회 = 쓰기 용량 단위 100

## DynamoDB Auto Scaling

DynamoDB Auto Scaling은 테이블 및 글로벌 보조 인덱스의 프로비저닝된 처리량 용량을 적극적으로 관리합니다. Auto Scaling을 사용하면 읽기 및 쓰기 용량 단위에 범위(상한 및 하한)를 지정할 수 있습니다. 또한 해당 범위에서의 목표 사용률을 정의할 수 있습니다. DynamoDB Auto Scaling은 애플리케이션 워크로드가 증가하거나 감소하는 경우에도 목표 사용률을 유지하려고 시도합니다.

DynamoDB Auto Scaling을 사용하면 테이블 또는 글로벌 보조 인덱스가 프로비저닝된 읽기 및 쓰기 용량을 늘려 요청 제한 없이 갑작스러운 트래픽 증가를 처리할 수 있습니다. 워크로드가 감소할 경우 DynamoDB Auto Scaling은 사용하지 않는 프로비저닝된 용량에 대한 요금을 지불하지 않도록 처리량을 줄일 수 있습니다.

> **Note**
>
> AWS Management Console을 사용하여 테이블이나 글로벌 보조 인덱스를 생성한 경우 DynamoDB Auto Scaling이 기본적으로 활성화됩니다.
> Auto Scaling 설정은 콘솔, AWS CLI 또는 AWS SDK 중 하나를 사용하여 언제든지 관리할 수 있습니다. 자세한 내용은 DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리 섹션을 참조하세요.

### 사용률

사용률은 프로비저닝 용량이 초과되었는지 파악하는 데 도움이 됩니다. 초과된 경우 비용을 절감하도록 테이블 용량을 줄여야 합니다. 반대로 프로비저닝 용량이 부족한지 파악하는 데도 도움이 될 수 있습니다. 이 경우 예기치 않게 높은 트래픽 인스턴스 중에 요청이 제한될 수 있는 가능성을 방지하기 위해 테이블 용량을 늘려야 합니다. 자세한 내용은 Amazon DynamoDB auto scaling: Performance and cost optimization at any scale을 참조하세요.

DynamoDB Auto Scaling을 사용하는 경우 목표 사용률도 설정해야 합니다. Auto Scaling에서는 이 비율을 목표로 삼아 용량을 늘리거나 줄입니다. 목표 사용률을 70%로 설정하는 것이 좋습니다. 자세한 내용은 DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리 섹션을 참조하세요.

## DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리

많은 데이터베이스 워크로드는 본질적으로 주기적인 반면 다른 워크로드는 미리 예측하기 어렵습니다. 낮 시간 동안에는 대부분의 사용자가 활성 상태인 소셜 네트워킹 앱을 한 가지 예로 들어 보겠습니다. 이러한 데이터베이스에서는 주간 활동을 처리할 수 있어야 하지만, 밤에는 동일한 수준의 처리량이 필요 없습니다. 예상치 못한 빠른 속도로 도입 중인 새로운 모바일 게임 앱을 또 다른 예로 들 수 있습니다. 이 게임의 인기가 너무 높아지면 사용 가능한 데이터베이스 리소스 양을 초과하여 성능이 느려지고 고객 불만이 발생할 것입니다. 이러한 종류의 워크로드는 대개 사용량 변화에 따라 수동 개입을 통해 데이터베이스 리소스의 규모를 늘리거나 줄여야 합니다.

Amazon DynamoDB Auto Scaling은 AWS Application Auto Scaling 서비스를 사용하여 프로비저닝된 처리 능력을 실제 트래픽 패턴에 따라 사용자 대신 동적으로 조정합니다. 따라서 테이블 또는 글로벌 보조 인덱스(GSI)에 따라 할당된 읽기 및 쓰기 용량을 늘려 제한 없이 갑작스러운 트래픽 증가를 처리할 수 있습니다. 워크로드가 감소할 경우 Application Auto Scaling은 사용하지 않는 프로비저닝된 용량에 대한 요금을 지불하지 않도록 처리량을 줄일 수 있습니다.

> **Note**
>
> AWS Management Console을 사용하여 테이블이나 글로벌 보조 인덱스를 생성한 경우 DynamoDB Auto Scaling이 기본적으로 활성화됩니다. 언제든지 Auto Scaling 설정을 변경할 수 있습니다. 자세한 내용은 DynamoDB Auto Scaling에서 AWS Management Console 사용 섹션을 참조하세요.
>
> 테이블 또는 글로벌 테이블 복제본을 삭제하면 연결된 확장 가능한 대상, 확장 정책 또는 CloudWatch 경보가 자동으로 삭제되지 않습니다.

Application Auto Scaling을 사용하여 테이블 또는 글로벌 보조 인덱스의 크기 조정 정책을 생성합니다. 이 규모 조정 정책을 통해 읽기 용량이나 쓰기 용량(또는 둘 다)을 조정할 것인지 여부와 테이블 또는 인덱스에 대해 할당된 용량 단위의 최댓값 및 최솟값 설정을 지정할 수 있습니다.

크기 조정 정책에는 특정 시점에 소비된 프로비저닝된 처리량의 비율인 목표 사용률도 포함됩니다. Application Auto Scaling은 목표 추적 알고리즘을 사용하여 실제 워크로드에 따라 테이블(또는 인덱스)의 프로비저닝된 처리량을 위나 아래로 조정하여 실제 용량 사용률이 목표 사용률 값이나 그에 가까운 수준으로 유지되도록 합니다.

DynamoDB 출력은 1분 동안 프로비저닝된 처리량을 소비했습니다. Auto Scaling은 소비된 용량이 구성된 목표 사용률을 2분 연속으로 초과할 때 트리거됩니다. CloudWatch 경보가 Auto Scaling을 트리거하기 전에 최대 몇 분의 짧은 지연이 있을 수 있습니다. 이러한 지연을 통해 정확한 CloudWatch 지표 평가가 이루어집니다. 하지만 소비된 처리량의 급증 간격이 1분보다 크면 Auto Scaling이 트리거되지 않을 수 있습니다. 마찬가지로 15개의 연속 데이터 포인트가 목표 사용률보다 낮을 때 스케일 다운 이벤트가 발생할 수 있습니다. 두 경우 모두 Auto Scaling이 트리거된 후 UpdateTable가 간접적으로 호출됩니다. 그런 다음 테이블 또는 인덱스의 프로비저닝된 용량을 업데이트하는 데 몇 분 정도 걸립니다. 이 기간 동안 테이블의 이전 프로비저닝된 용량을 초과하는 모든 요청은 제한됩니다.

> **Important**
>
> 위반할 데이터 포인트 수를 조정하여 기본 경보를 트리거할 수는 없습니다(현재 숫자는 향후 변경될 수 있음).

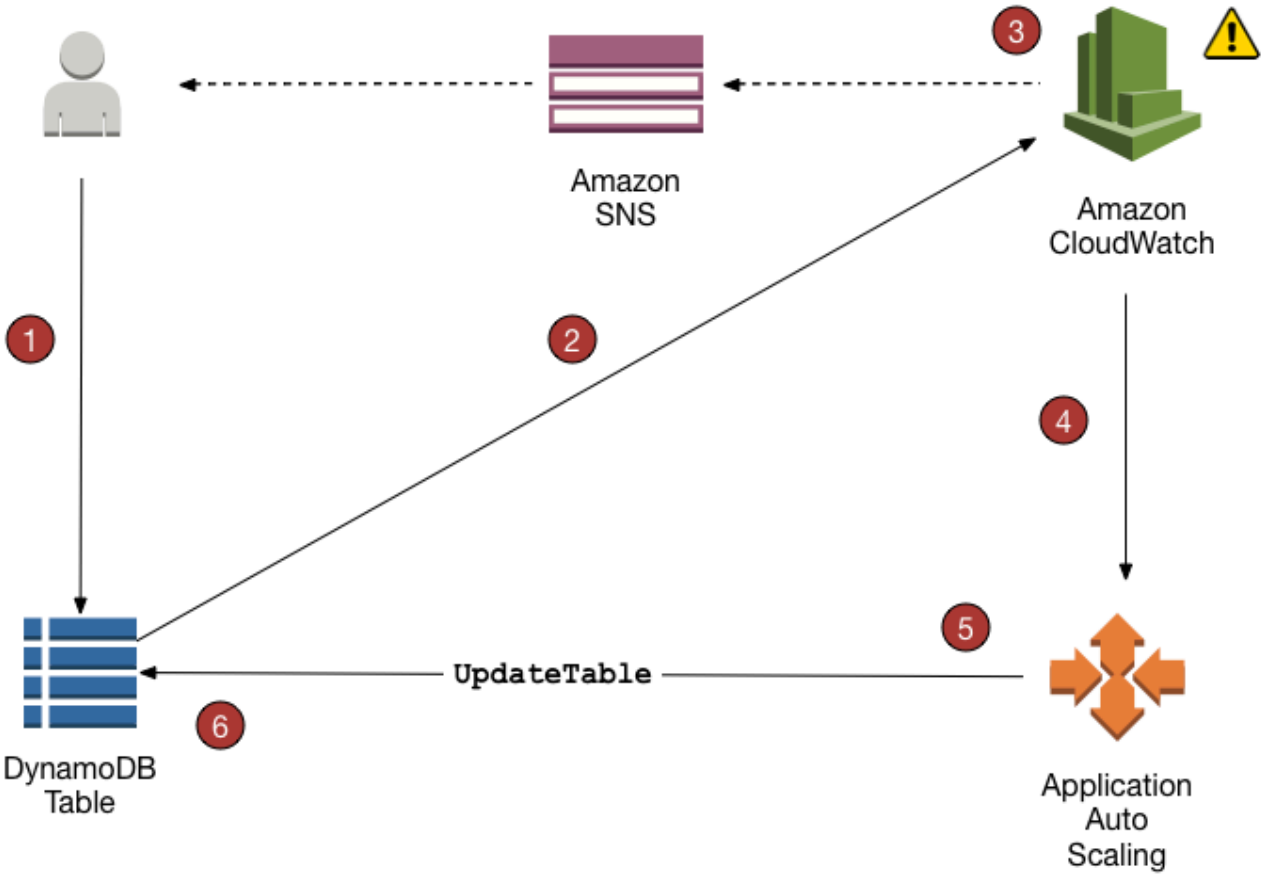
읽기 및 쓰기 용량에 대해 20%와 90% 사이에서 Auto Scaling 목표 사용률 값을 설정할 수 있습니다.

> **Note**
>
> DynamoDB Auto Scaling은 테이블 외에 글로벌 보조 인덱스도 지원합니다. 모든 글로벌 보조 인덱스에는 기본 테이블과 별도로 자체 프로비저닝된 처리 능력이 있습니다. 글로벌 보조 인덱스에 대한 크기 조정 정책을 생성하면 Application Auto Scaling은 해당 인덱스의 프로비저닝된 처리량 설정을 조절하여 실제 사용률이 원하는 사용률이나 그에 가깝게 유지되도록 합니다.

# DynamoDB Auto Scaling 작동 방식

> **Note**
>
> DynamoDB Auto Scaling을 빠르게 시작하려면 DynamoDB Auto Scaling에서 AWS Management Console 사용 섹션을 참조하세요.

아래 그림에는 DynamoDB Auto Scaling이 테이블의 처리 용량을 관리하는 방법이 간단히 소개되어 있습니다.




다음은 앞의 그림에 소개된 Auto Scaling 프로세스를 요약한 단계입니다.

1. DynamoDB 테이블에 대한 Application Auto Scaling 정책을 생성합니다.
2. DynamoDB는 Amazon CloudWatch에 소비된 용량 지표를 게시합니다.

3. 테이블에서 사용한 용량이 특정 기간의 목표 사용률을 초과하는 경우(또는 목표에 미달하는 경우), Amazon CloudWatch는 경보를 트리거합니다. 콘솔에서 이 경보를 확인하고 Amazon Simple Notification Service(Amazon SNS)를 사용하여 알림을 받을 수 있습니다.
4. CloudWatch 경보를 받으면 크기 조정 정책을 평가하기 위해 Application Auto Scaling이 호출됩니다.
5. Application Auto Scaling은 `UpdateTable` 요청을 실행하여 테이블의 프로비저닝된 처리량을 조정합니다.
6. DynamoDB는 `UpdateTable` 요청을 처리하고 해당 테이블의 프로비저닝된 처리 용량을 동적으로 늘리거나 줄임으로써 목표 사용률에 근접하게 합니다.

DynamoDB Auto Scaling의 작동 방식을 알아보기 위해 `ProductCatalog`라는 테이블이 있다고 가정합니다. 이 테이블에는 부정기적으로 데이터가 대량 로드되며, 따라서 쓰기 활동이 많지는 않습니다. 그러나 읽기 활동은 높은 수준으로 이루어지며 그 양은 시간에 따라 달라집니다. `ProductCatalog`에 대한 Amazon CloudWatch 지표를 모니터링함으로써 이 테이블에 읽기 용량 단위 1,200개가 필요하다는 사실을 알 수 있습니다(활동이 최대 수준일 때 DynamoDB에서 읽기 요청을 제한하지 않아도 되는 정도). 이와 함께 `ProductCatalog`에는 읽기 트래픽이 최소 수준일 때 읽기 용량 단위가 최소한 150개 필요하다고 판단됩니다. 제한 방지에 대한 자세한 내용은 DynamoDB의 제한 문제 섹션을 참조하세요.

그러므로 읽기 용량 단위 150개 ~ 1,200개 범위에서 목표 사용률 70%면 `ProductCatalog` 테이블에 적당하다고 결정합니다. 목표 사용률이란 프로비저닝된 용량 단위에 대한 소비된 용량 단위의 비율을 백분율로 나타낸 값입니다. Application Auto Scaling은 목표 추적 알고리즘을 사용하여 `ProductCatalog`의 프로비저닝된 읽기 용량을 필요에 따라 조절함으로써 사용률이 70% 안팎으로 유지되도록 합니다.

> **Note**
>
> DynamoDB Auto Scaling은 실제 워크로드가 일정 시간(분) 동안 높게 또는 낮게 유지되는 경우에 한해 프로비저닝된 처리량(throughput) 설정을 수정합니다. Application Auto Scaling의 목표 추적 알고리즘은 목표 사용률을 장기적으로 사용자가 선택한 값 안팎으로 유지되도록 합니다. 짧은 기간 동안 갑자기 급증하는 활동은 테이블에 기본 제공되는 버스트 용량으로 처리합니다. 자세한 내용은 버스트 용량 섹션을 참조하세요.

`ProductCatalog` 테이블에 대해 DynamoDB Auto Scaling을 활성화하기 위해 크기 조정 정책을 만듭니다. 이 정책은 다음을 지정합니다.

- 관리하려는 테이블 또는 글로벌 보조 인덱스
- 관리하려는 용량 유형(읽기 용량 또는 쓰기 용량)
- 프로비저닝된 처리량 설정의 상한값과 하한값
- 목표 사용률

크기 조정 정책이 생성되면 Application Auto Scaling은 사용자 대신 Amazon CloudWatch 경보 쌍을 생성합니다. 각 쌍은 할당된 처리량(throughput) 설정의 상한값과 하한값을 나타냅니다. 테이블의 실제 사용률이 일정한 시간 동안 목표 사용률을 벗어나면 이러한 CloudWatch 경보가 트리거됩니다.

CloudWatch 경보 중 하나가 트리거되면 Amazon SNS에서 알림을 보냅니다(알림을 활성화한 경우). 그러면 CloudWatch 경보가 Application Auto Scaling을 호출하고, 여기서 다시 DynamoDB에 알려 `ProductCatalog` 테이블의 프로비저닝된 용량을 적절히 상향 또는 하향 조정하도록 합니다.

규모 조정 이벤트 중에 AWS Config에는 기록된 구성 항목당 요금이 부과됩니다. 규모 조정 이벤트가 발생하면 각 읽기 및 쓰기 Auto Scaling 이벤트에 대해 4개의 CloudWatch 경보가 생성됩니다. 즉, ProvisionedCapacity 경보인 ProvisionedCapacityLow, ProvisionedCapacityHigh와 ConsumedCapacity 경보인 AlarmHigh, AlarmLow입니다. 그 결과 총 8개의 경보가 발생합니다. 따라서 AWS Config는 모든 규모 조정 이벤트에 대해 8개의 구성 항목을 기록합니다.

> **Note**
>
> 또한 DynamoDB 규모 조정이 특정 시간에 이루어지도록 예약할 수 있습니다. 여기에서 기본 단계를 알아보세요.

## 사용 노트

DynamoDB Auto Scaling을 사용하려면 먼저 다음 내용을 이해해야 합니다.

- DynamoDB Auto Scaling은 Auto Scaling 정책에 따라 필요한 만큼 자주 읽기 용량이나 쓰기 용량을 늘릴 수 있습니다. 모든 DynamoDB 할당량은 Amazon DynamoDB의 할당량에서 설명한 것처럼 계속 유효합니다.
- DynamoDB Auto Scaling은 프로비저닝된 처리량 설정을 사용자가 수동으로 수정하는 것도 허용합니다. 이러한 수동 조정은 DynamoDB Auto Scaling과 관련된 기존의 CloudWatch 경보에는 영향을 주지 않습니다.
- 글로벌 보조 인덱스가 하나 이상인 테이블에 대해 DynamoDB Auto Scaling을 활성화하는 경우, 해당 인덱스에도 Auto Scaling을 균일하게 적용하는 것이 좋습니다. 이를 통해 테이블 쓰기 및 읽기 성

능이 향상되고 제한을 방지할 수 있습니다. AWS Management Console에서 Apply same settings to global secondary indexes(글로벌 보조 인덱스에 동일한 설정 적용)를 선택하여 Auto Scaling을 사용하도록 설정할 수 있습니다. 자세한 내용은 기존 테이블에서 DynamoDB Auto Scaling 활성화 섹션을 참조하세요.

- 테이블 또는 전역 테이블 복제본을 삭제하면 연결된 확장 가능한 대상, 확장 정책 또는 CloudWatch 경보가 자동으로 삭제되지 않습니다.
- 기존 테이블에 대한 GSI를 생성할 때 GSI에 대해 Auto Scaling이 활성화되지 않습니다. GSI 빌드 중에는 용량을 수동으로 관리해야 합니다. GSI의 백필이 완료되고 활성 상태에 도달하면 Auto Scaling이 정상적으로 작동합니다.

## DynamoDB Auto Scaling에서 AWS Management Console 사용

AWS Management Console을 사용하여 새 테이블을 생성할 경우, 해당 테이블에 대해 Amazon DynamoDB Auto Scaling이 기본적으로 활성화됩니다. 또한 콘솔을 사용하여 기존 테이블에 대해 Auto Scaling을 활성화하거나, Auto Scaling을 수정하거나, Auto Scaling을 비활성화할 수 있습니다.

> **Note**
>
> 휴지 시간 확장 및 축소 설정과 같은 추가 고급 기능의 경우, AWS Command Line Interface(AWS CLI)를 사용하여 DynamoDB Auto Scaling을 관리합니다. 자세한 내용은 AWS CLI를 사용하여 DynamoDB Auto Scaling 관리 섹션을 참조하세요.

주제

- 시작하기 전에: 사용자에게 DynamoDB Auto Scaling에 대한 권한 부여
- Auto Scaling을 활성화하여 새 테이블 만들기
- 기존 테이블에서 DynamoDB Auto Scaling 활성화
- 콘솔에서 Auto Scaling 활동 보기
- DynamoDB Auto Scaling 설정 수정 또는 비활성화

### 시작하기 전에: 사용자에게 DynamoDB Auto Scaling에 대한 권한 부여

AWS Identity and Access Management(IAM)에서 AWS 관리형 정책 `DynamoDBFullAccess`는 DynamoDB 콘솔을 사용하는 데 필요한 권한을 제공합니다. 하지만 DynamoDB Auto Scaling의 경우에는 사용자에게 몇 가지 권한이 추가로 필요합니다.

> **Important**
>
> Auto scaling이 활성화된 테이블을 삭제하려면 `application-autoscaling:*` 권한이 필요합니다. AWS 관리형 정책인 `DynamoDBFullAccess`에 그러한 권한이 포함되어 있습니다.

DynamoDB 콘솔 액세스 및 DynamoDB Auto Scaling의 사용자를 설정하려면 역할을 생성하고 AmazonDynamoDBFullAccess 정책을 해당 역할에 추가합니다. 그런 다음 사용자에게 역할을 할당합니다.

Auto Scaling을 활성화하여 새 테이블 만들기

> **Note**
>
> DynamoDB Auto Scaling을 사용하려면 사용자 대신 Auto Scaling 작업을 수행하는 서비스 연결 역할(`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`)이 있어야 합니다. 이 역할은 자동으로 생성됩니다. 자세한 정보는 Application Auto Scaling 사용 설명서의 Application Auto Scaling 서비스 연결 역할을 참조하세요.

Auto Scaling을 활성화하여 테이블을 새로 만들려면

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. [Create table]을 선택합니다.
3. 테이블 생성 페이지에서 테이블 이름과 프라이머리 키 정보를 입력합니다.
4. 기본 설정을 선택하면 새 테이블에서 Auto Scaling이 활성화됩니다.

   또는 설정 사용자 지정을 선택한 후 다음을 수행하여 테이블에 대한 사용자 지정 설정을 지정합니다.

   a. 테이블 클래스에서 DynamoDB Standard의 기본 선택을 유지합니다.

   b. 읽기/쓰기 용량 설정에서 기본 선택인 프로비저닝됨을 유지하고 다음을 수행합니다.

      i. 읽기 용량에서 Auto Scaling을 켜기로 설정해야 합니다.

      ii. 읽기 용량에서 Auto Scaling을 켜기로 설정해야 합니다.

      iii. 읽기 용량 및 쓰기 용량에 대해 테이블에 원하는 크기 조정 정책을 설정하고 선택적으로 테이블의 모든 글로벌 보조 인덱스를 설정합니다.

- 최소 용량 단위 - Auto Scaling 범위의 하한값을 입력합니다.
- 최대 용량 단위 - Auto Scaling 범위의 상한값을 입력합니다.
- 목표 사용률 - 테이블의 목표 사용률을 입력합니다.

> **Note**
>
> 새 테이블에 대한 글로벌 보조 인덱스를 생성하는 경우 생성 시 인덱스의 용량은 기본 테이블의 용량과 동일합니다. 테이블을 만든 후 테이블 설정에서 인덱스의 용량을 변경할 수 있습니다.

5. 테이블 생성을 선택합니다. 지정한 Auto Scaling 파라미터로 테이블이 생성됩니다.

기존 테이블에서 DynamoDB Auto Scaling 활성화

> **Note**
>
> DynamoDB Auto Scaling을 사용하려면 사용자 대신 Auto Scaling 작업을 수행하는 서비스 연결 역할(`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`)이 있어야 합니다. 이 역할은 자동으로 생성됩니다. 자세한 정보는 Application Auto Scaling에 대한 서비스 연결 역할을 참조하세요.

기존 테이블에 대해 DynamoDB Auto Scaling을 활성화하려면

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 테이블을 선택합니다.
3. Auto Scaling을 활성화하려는 테이블을 선택한 후 다음을 수행합니다.
   a. 추가 설정 탭으로 이동합니다.
   b. 읽기/쓰기 용량 섹션에서 편집을 선택합니다.
   c. 용량 모드 섹션에서 프로비저닝됨을 선택합니다.
   d. Table capacity(테이블 용량) 섹션에서 Read capacity(읽기 용량), Write capacity(쓰기 용량) 또는 양쪽 모두에 대해 Auto Scaling(자동 크기 조정)을 On(켜기)으로 설정합니다. 이러한 각 항목에 대해 테이블에 원하는 크기 조정 정책을 설정하고 선택적으로 테이블의 모든 글로벌 보조 인덱스를 설정합니다.

- 최소 용량 단위 - Auto Scaling 범위의 하한값을 입력합니다.
- 최대 용량 단위 - Auto Scaling 범위의 상한값을 입력합니다.
- 목표 사용률 - 테이블의 목표 사용률을 입력합니다.
- 모든 글로벌 보조 인덱스에 대해 동일한 용량 읽기/쓰기 용량 설정 사용 - 글로벌 보조 인덱스에서 기본 테이블과 동일한 Auto Scaling 정책을 사용할지 여부를 선택합니다.

> **Note**
>
> 최상의 성능을 내려면 글로벌 보조 인덱스에 동일한 설정 적용(Use the same read/write capacity settings for all global secondary indexes)을 활성화하는 것이 좋습니다. 이 옵션을 사용하면 DynamoDB Auto Scaling에서 기본 테이블의 모든 글로벌 보조 인덱스를 균일하게 조정할 수 있습니다. 기존 글로벌 보조 인덱스와 향후 이 테이블에 대해 생성하는 모든 인덱스에 적용됩니다.
> 이 옵션을 활성화하면 개별 글로벌 보조 인덱스에 대해 크기 조정 정책을 설정할 수 없습니다.

4. 원하는 대로 설정되었으면 [Save]를 선택합니다.

콘솔에서 Auto Scaling 활동 보기

애플리케이션에서 해당 테이블에 대해 읽기 및 쓰기 트래픽을 구동할 경우 DynamoDB Auto Scaling은 테이블의 처리량 설정을 동적으로 수정합니다. Amazon CloudWatch는 모든 DynamoDB 테이블 및 보조 인덱스에 대한 프로비저닝되고 사용된 용량, 제한 이벤트, 지연 시간 및 기타 지표를 추적합니다.

DynamoDB 콘솔에서 이러한 지표를 보려면 작업할 테이블을 선택하고 모니터링 탭을 선택합니다. 테이블 지표의 사용자 지정 가능한 보기를 만들려면 CloudWatch에서 모두 보기(View all in CloudWatch)를 선택합니다.

DynamoDB Auto Scaling 설정 수정 또는 비활성화

AWS Management Console을DynamoDB 사용하여 Auto Scaling 설정을 수정할 수 있습니다. 이렇게 하려면 테이블의 추가 설정 탭으로 이동하여 읽기/쓰기 용량 섹션에서 편집을 선택합니다. 이러한 설정에 대한 자세한 내용은 기존 테이블에서 DynamoDB Auto Scaling 활성화 섹션을 참조하세요.

## AWS CLI를 사용하여 DynamoDB Auto Scaling 관리

AWS Management Console을 사용하는 대신 AWS Command Line Interface(AWS CLI)를 사용하여 Amazon DynamoDB Auto Scaling을 관리할 수 있습니다. 이 섹션의 자습서에서는 DynamoDB Auto

Scaling을 관리할 수 있도록 AWS CLI를 설치 및 구성하는 방법을 보여 줍니다. 이 자습서에서는 다음을 수행합니다.

- `TestTable`이라는 DynamoDB 테이블을 생성합니다. 이 테이블의 초기 처리량 설정은 읽기 용량 단위 5, 쓰기 용량 단위 5입니다.
- `TestTable`에 대한 Application Auto Scaling 정책을 생성합니다. 정책은 사용된 쓰기 용량과 할당된 쓰기 용량 사이의 비율이 50퍼센트 목표 비율이 되도록 유지하려 합니다. 이 수치의 범위는 쓰기 용량 단위 5-10입니다. (Application Auto Scaling은 이 범위를 초과하여 처리량을 조정할 수 없습니다.)
- Python 프로그램을 실행하여 `TestTable`에 대해 쓰기 트래픽을 실행합니다. 목표 비율이 장기적으로 50퍼센트를 초과할 경우 Application Auto Scaling은 목표 사용률의 50퍼센트가 유지될 수 있도록 DynamoDB에 `TestTable` 처리량을 상향 조정하도록 알립니다.
- DynamoDB가 `TestTable`에 대한 프로비저닝된 쓰기 용량을 조정했는지 확인합니다.

> **Note**
>
> 또한 DynamoDB 규모 조정이 특정 시간에 이루어지도록 예약할 수 있습니다. 여기에서 기본 단계를 알아보세요.

주제


- 시작하기 전 준비 사항
- 1단계: DynamoDB 테이블 생성
- 2단계: 크기 조정 가능 대상 등록
- 3단계: 크기 조정 정책 생성
- 4단계: TestTable로 쓰기 트래픽 유도
- 5단계: Application Auto Scaling 작업 보기
- (선택 사항) 6단계: 정리


## 시작하기 전 준비 사항

이 자습서를 시작하기 전에 다음 작업을 완료해야 합니다.

### AWS CLI 설치

AWS CLI를 아직 설치하지 않았다면 이를 설치하고 구성해야 합니다. 이를 위해 AWS Command Line Interface 사용 설명서에서 다음 지침을 따르세요.

- AWS CLI 설치
- AWS CLI 구성

Python 을 설치합니다.

이 자습서에서는 Python 프로그램을 실행해야 합니다.(4단계: TestTable로 쓰기 트래픽 유도 참조). 아직 설치하지 않은 경우 Python을 다운로드 할 수 있습니다 .

1단계: DynamoDB 테이블 생성

이 단계에서는 AWS CLI를 사용하여 `TestTable`를 생성합니다. 기본 키는 `pk`(파티션 키)와 `sk`(정렬 키)로 구성됩니다. 이 두 가지 속성 모두 `Number` 유형입니다. 이 테이블의 초기 처리량 설정은 읽기 용량 단위 5, 쓰기 용량 단위 5입니다.

1. 다음 AWS CLI 명령을 사용하여 테이블을 생성합니다.

```
aws dynamodb create-table \
    --table-name TestTable \
    --attribute-definitions \
        AttributeName=pk,AttributeType=N \
        AttributeName=sk,AttributeType=N \
    --key-schema \
        AttributeName=pk,KeyType=HASH \
        AttributeName=sk,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

2. 테이블 상태를 확인하려면 다음 명령을 사용하세요.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

   테이블 상태가 `ACTIVE`이면 사용할 수 있습니다.

2단계: 크기 조정 가능 대상 등록

이제 Application Auto Scaling을 사용하여 테이블의 쓰기 용량을 확장 가능 목표로 등록할 수 있습니다. 이렇게 하면 Application Auto Scaling에서 TestTable에 대한 프로비저닝된 쓰기 용량을 조정할 수 있지만, 조정 범위는 용량 단위 5~10으로 제한됩니다.

> **Note**
>
> DynamoDB Auto Scaling을 사용하려면 사용자 대신 Auto Scaling 작업을 수행하는 서비스 연결 역할(`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`)이 있어야 합니다. 이 역할은 자동으로 생성됩니다. 자세한 정보는 Application Auto Scaling 사용 설명서의 Application Auto Scaling 서비스 연결 역할을 참조하세요.

1. 다음 명령을 입력하여 확장 가능 목표를 등록합니다.

```
aws application-autoscaling register-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --min-capacity 5 \
    --max-capacity 10
```

2. 등록을 확인하려면 다음 명령을 사용합니다.

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace dynamodb \
    --resource-id "table/TestTable"
```

> **Note**
>
> 글로벌 보조 인덱스에 대한 확장 가능 목표를 등록할 수도 있습니다. 예를 들어 글로벌 보조 인덱스("test-index")의 경우 리소스 ID와 확장형 차원 인수는 적절하게 업데이트됩니다.
>
> ```
> aws application-autoscaling register-scalable-target \
>     --service-namespace dynamodb \
>     --resource-id "table/TestTable/index/test-index" \
>     --scalable-dimension "dynamodb:index:WriteCapacityUnits" \
>     --min-capacity 5 \
>     --max-capacity 10
> ```

### 3단계: 크기 조정 정책 생성

이 단계에서는 TestTable의 확장 정책을 만듭니다. 이 정책은 Application Auto Scaling이 테이블의 프로비저닝된 처리량을 조정할 수 있는 위치와 조정 시 수행하는 작업에 대한 정보를 정의합니다. 이 정책을 전 단계에서 정의한 확장 가능 목표(TestTable 테이블에 대한 쓰기 용량 단위)와 연결합니다.

정책에는 다음 요소가 포함됩니다.

- PredefinedMetricSpecification - Application Auto Scaling이 조정할 수 있는 지표입니다. DynamoDB의 경우 PredefinedMetricType에 다음 값을 사용할 수 있습니다.
  - DynamoDBReadCapacityUtilization
  - DynamoDBWriteCapacityUtilization
- ScaleOutCooldown - 프로비저닝된 처리량을 증가시키는 각 Application Auto Scaling 이벤트 간의 최소 시간(초). 이 파라미터를 사용하면 Application Auto Scaling이 실제 워크로드에 대응하여 지속적으로 과도하지 않게 처리량을 올릴 수 있습니다. ScaleOutCooldown의 기본 설정은 0입니다.
- ScaleInCooldown - 프로비저닝된 처리량을 줄이는 각 Application Auto Scaling 이벤트 간의 최소 시간(초). 이 파라미터를 사용하면 Application Auto Scaling이 처리량을 점진적으로 예측 가능하게 줄일 수 있습니다. ScaleInCooldown의 기본 설정은 0입니다.
- TargetValue - Application Auto Scaling이 사용되는 용량과 프로비저닝된 용량의 비율을 이 값 수준으로 유지하도록 합니다. TargetValue를 백분율로 지정합니다.

> **Note**
>
> TargetValue의 용도를 자세히 이해하기 위해 쓰기 용량 단위 200으로 할당 처리량을 설정한 테이블을 예로 들어 보겠습니다. 이 테이블에 대해 TargetValue가 70퍼센트인 확장 정책을 만들려고 합니다.
> 테이블에 대한 쓰기 트래픽을 시작했더니 실제 쓰기 처리량이 150 용량 단위였다고 가정해 보겠습니다. 사용 용량 대 할당 용량의 비율은 150/200으로 75퍼센트입니다. 이 비율은 목표를 초과하므로 Application Auto Scaling은 프로비저닝된 쓰기 용량을 215로 늘려 비율이 (150/215) 또는 69.77%(TargetValue를 초과하지 않지만 최대한 가깝게)가 되게 합니다.

TestTable의 경우 TargetValue를 50%로 설정합니다. Application Auto Scaling는 테이블의 프로비저닝된 처리량을 용량 단위 5~10의 범위 내에서 조정(2단계: 크기 조정 가능 대상 등록 참조)하여 사용 용량과 프로비저닝된 용량 간의 비율이 50% 수준으로 유지되도록 합니다. ScaleOutCooldown와 ScaleInCooldown의 값을 60초로 설정합니다.

1. 다음 콘텐츠를 가진 `scaling-policy.json`이라는 파일을 생성합니다:

```
{
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "TargetValue": 50.0
}
```

2. 다음 AWS CLI 명령을 사용하여 정책을 생성합니다.

```
aws application-autoscaling put-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy" \
    --policy-type "TargetTrackingScaling" \
    --target-tracking-scaling-policy-configuration file://scaling-policy.json
```

3. 출력에서 Application Auto Scaling이 2개의 Amazon CloudWatch 경보를 생성했음을 확인합니다. 하나는 크기 조정 목표 범위의 상한에 대한 경보이고 하나는 하한에 대한 경보입니다.
4. 다음 AWS CLI 명령을 사용하여 조정 정책에 대한 세부 정보를 확인합니다.

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --policy-name "MyScalingPolicy"
```

5. 출력에서 정책 설정이 2단계: 크기 조정 가능 대상 등록 및 3단계: 크기 조정 정책 생성의 사양과 일치함을 확인합니다.

### 4단계: TestTable로 쓰기 트래픽 유도

이제 `TestTable`에 데이터를 써서 확장 정책을 테스트할 수 있습니다. 이렇게 하려면 Python 프로그램을 실행합니다.

1. 다음 콘텐츠를 가진 `bulk-load-test-table.py`이라는 파일을 생성합니다:

```
import boto3
```

```
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table("TestTable")

filler = "x" * 100000

i = 0
while (i < 10):
    j = 0
    while (j < 10):
        print (i, j)

        table.put_item(
            Item={
                'pk':i,
                'sk':j,
                'filler':{"S":filler}
            }
        )
        j += 1
    i += 1
```

2. 다음 명령을 입력하여 프로그램을 실행합니다.

   ```
   python bulk-load-test-table.py
   ```

   TestTable의 프로비저닝된 쓰기 용량은 쓰기 용량 단위 5로 매우 낮아서 쓰기 조절(throttling)로 인해 프로그램이 때때로 중단됩니다. 이는 예상된 동작입니다.

   다음 단계로 이동하여 프로그램이 지속적으로 실행될 수 있도록 해 보겠습니다.

### 5단계: Application Auto Scaling 작업 보기

이 단계에서는 시작된 Application Auto Scaling 작업을 확인합니다. 또한 Application Auto Scaling이 TestTable의 프로비저닝된 쓰기 용량을 업데이트했는지 확인합니다.

1. 다음 명령을 입력하여 Application Auto Scaling 작업을 확인합니다.

   ```
   aws application-autoscaling describe-scaling-activities \
       --service-namespace dynamodb
   ```

Python 프로그램이 실행되는 동안 이 명령을 때때로 다시 실행하세요. (조정 정책이 호출되려면 몇 분 정도 걸립니다.) 다음과 같은 출력이 표시됩니다.

```
...
{
    "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
    "Description": "Setting write capacity units to 10.",
    "ResourceId": "table/TestTable",
    "ActivityId": "0cc6fb03-2a7c-4b51-b67f-217224c6b656",
    "StartTime": 1489088210.175,
    "ServiceNamespace": "dynamodb",
    "EndTime": 1489088246.85,
    "Cause": "monitor alarm AutoScaling-table/TestTable-
AlarmHigh-1bb3c8db-1b97-4353-baf1-4def76f4e1b9 in state ALARM triggered policy
 MyScalingPolicy",
    "StatusMessage": "Successfully set write capacity units to 10. Change
 successfully fulfilled by dynamodb.",
    "StatusCode": "Successful"
},
...
```

이는 Application Auto Scaling이 DynamoDB에 `UpdateTable` 요청을 보냈음을 나타냅니다.

2. 다음 명령을 입력하여 DynamoDB가 테이블의 쓰기 용량을 늘렸음을 확인합니다.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

`WriteCapacityUnits`가 `5`에서 `10`으로 확장되었습니다.

## (선택 사항) 6단계: 정리

이 자습서에서 몇 가지 리소스를 만들어 보았습니다. 이러한 리소스가 더 이상 필요하지 않으면 삭제하면 됩니다.

1. `TestTable`에 대한 조정 정책을 삭제합니다.

```
aws application-autoscaling delete-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
```

```
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy"
```

2. 확장 가능 목표의 등록을 취소합니다.

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits"
```

3. `TestTable` 테이블을 삭제합니다.

```
aws dynamodb delete-table --table-name TestTable
```

## AWS SDK를 사용하여 Amazon DynamoDB 테이블에서 Auto Scaling 구성

AWS Management Console 및 AWS Command Line Interface(AWS CLI)를 사용하는 방법 외에 Amazon DynamoDB Auto Scaling과 상호 작용하는 애플리케이션을 작성할 수도 있습니다. 이 섹션에서는 이 기능을 테스트하는 데 사용할 수 있는 2가지 Java 프로그램이 나와 있습니다.

- `EnableDynamoDBAutoscaling.java`
- `DisableDynamoDBAutoscaling.java`

### 테이블에 대해 Application Auto Scaling 활성화

다음 프로그램은 DynamoDB 테이블(`TestTable`)에 대해 Auto Scaling 정책을 설정하는 예제를 보여줍니다. 작동 방식은 다음과 같습니다.

- 프로그램에서 `TestTable`의 확장 가능 목표로 쓰기 용량 단위를 등록합니다. 이 수치의 범위는 쓰기 용량 단위 5-10입니다.
- 확장 가능 목표가 생성되면 프로그램에서 목표 추적 구성을 설정합니다. 정책은 사용된 쓰기 용량과 할당된 쓰기 용량 사이의 비율이 50퍼센트 목표 비율이 되도록 유지하려 합니다.
- 그런 다음 프로그램은 대상 추적 구성에 따라 확장 정책을 생성합니다.

> **Note**
>
> 테이블 또는 전역 테이블 복제본을 수동으로 제거하면 연결된 확장 가능한 대상, 확장 정책 또는 CloudWatch 경보가 자동으로 제거되지 않습니다.

Java v2

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExceptio
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRespons
import software.amazon.awssdk.services.applicationautoscaling.model.PolicyType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.PredefinedMetricSpecification;
import
 software.amazon.awssdk.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 software.amazon.awssdk.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import software.amazon.awssdk.services.applicationautoscaling.model.ScalingPolicy;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import software.amazon.awssdk.services.applicationautoscaling.model.MetricType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.TargetTrackingScalingPolicyCon
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
```

```
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class EnableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <roleARN> <policyName>\s

            Where:
               tableId - The table Id value (for example, table/Music).
               roleARN - The ARN of the role that has ApplicationAutoScaling
 permissions.
               policyName - The name of the policy to create.

            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        System.out.println("This example registers an Amazon DynamoDB table, which
 is the resource to scale.");
        String tableId = args[0];
        String roleARN = args[1];
        String policyName = args[2];
        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
 ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        ApplicationAutoScalingClient appAutoScalingClient =
 ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
            .build();

        registerScalableTarget(appAutoScalingClient, tableId, roleARN, ns,
 tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        configureScalingPolicy(appAutoScalingClient, tableId, ns, tableWCUs,
 policyName);
    }
```

```
   public static void registerScalableTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, String roleARN, ServiceNamespace ns,
ScalableDimension tableWCUs) {
       try {
           RegisterScalableTargetRequest targetRequest =
RegisterScalableTargetRequest.builder()
               .serviceNamespace(ns)
               .scalableDimension(tableWCUs)
               .resourceId(tableId)
               .roleARN(roleARN)
               .minCapacity(5)
               .maxCapacity(10)
               .build();

           appAutoScalingClient.registerScalableTarget(targetRequest);
           System.out.println("You have registered " + tableId);

       } catch (ApplicationAutoScalingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
       }
   }

   // Verify that the target was created.
   public static void verifyTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
       DescribeScalableTargetsRequest dscRequest =
DescribeScalableTargetsRequest.builder()
           .scalableDimension(tableWCUs)
           .serviceNamespace(ns)
           .resourceIds(tableId)
           .build();

       DescribeScalableTargetsResponse response =
appAutoScalingClient.describeScalableTargets(dscRequest);
       System.out.println("DescribeScalableTargets result: ");
       System.out.println(response);
   }

   // Configure a scaling policy.
   public static void configureScalingPolicy(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs, String policyName) {
       // Check if the policy exists before creating a new one.
```

```
        DescribeScalingPoliciesResponse describeScalingPoliciesResponse =
appAutoScalingClient.describeScalingPolicies(DescribeScalingPoliciesRequest.builder()
            .serviceNamespace(ns)
            .resourceId(tableId)
            .scalableDimension(tableWCUs)
            .build());

        if (!describeScalingPoliciesResponse.scalingPolicies().isEmpty()) {
            // If policies exist, consider updating an existing policy instead of
creating a new one.
            System.out.println("Policy already exists. Consider updating it
instead.");
            List<ScalingPolicy> polList =
describeScalingPoliciesResponse.scalingPolicies();
            for (ScalingPolicy pol : polList) {
                System.out.println("Policy name:" +pol.policyName());
            }
        } else {
            // If no policies exist, proceed with creating a new policy.
            PredefinedMetricSpecification specification =
PredefinedMetricSpecification.builder()

.predefinedMetricType(MetricType.DYNAMO_DB_WRITE_CAPACITY_UTILIZATION)
                .build();

            TargetTrackingScalingPolicyConfiguration policyConfiguration =
TargetTrackingScalingPolicyConfiguration.builder()
                .predefinedMetricSpecification(specification)
                .targetValue(50.0)
                .scaleInCooldown(60)
                .scaleOutCooldown(60)
                .build();

            PutScalingPolicyRequest putScalingPolicyRequest =
PutScalingPolicyRequest.builder()
                .targetTrackingScalingPolicyConfiguration(policyConfiguration)
                .serviceNamespace(ns)
                .scalableDimension(tableWCUs)
                .resourceId(tableId)
                .policyName(policyName)
                .policyType(PolicyType.TARGET_TRACKING_SCALING)
                .build();

            try {
```

```
                appAutoScalingClient.putScalingPolicy(putScalingPolicyRequest);
                System.out.println("You have successfully created a scaling policy
 for an Application Auto Scaling scalable target");
            } catch (ApplicationAutoScalingException e) {
                System.err.println("Error: " + e.awsErrorDetails().errorMessage());
            }
        }
    }
}
```

Java v1

이 프로그램을 사용하려면 유효한 Application Auto Scaling 서비스 연결 역할의 Armazon 리소스 이름(ARN)을 제공해야 합니다. 예: `arn:aws:iam::122517410325:role/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`. 다음 프로그램에서 `SERVICE_ROLE_ARN_GOES_HERE`를 실제 ARN으로 변경합니다.

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClientBuilder;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.MetricType;
import com.amazonaws.services.applicationautoscaling.model.PolicyType;
import
 com.amazonaws.services.applicationautoscaling.model.PredefinedMetricSpecification;
import com.amazonaws.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;
import
 com.amazonaws.services.applicationautoscaling.model.TargetTrackingScalingPolicyConfiguratio
```

```
public class EnableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = (AWSApplicationAutoScalingClient)
 AWSApplicationAutoScalingClientBuilder
   .standard().build();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
  ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
  String resourceID = "table/TestTable";

  // Define the scalable target
  RegisterScalableTargetRequest rstRequest = new RegisterScalableTargetRequest()
    .withServiceNamespace(ns)
    .withResourceId(resourceID)
    .withScalableDimension(tableWCUs)
    .withMinCapacity(5)
    .withMaxCapacity(10)
    .withRoleARN("SERVICE_ROLE_ARN_GOES_HERE");

  try {
   aaClient.registerScalableTarget(rstRequest);
  } catch (Exception e) {
   System.err.println("Unable to register scalable target: ");
   System.err.println(e.getMessage());
  }

  // Verify that the target was created
  DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceIds(resourceID);
  try {
   DescribeScalableTargetsResult dsaResult =
aaClient.describeScalableTargets(dscRequest);
   System.out.println("DescribeScalableTargets result: ");
   System.out.println(dsaResult);
   System.out.println();
  } catch (Exception e) {
   System.err.println("Unable to describe scalable target: ");
   System.err.println(e.getMessage());
  }
```

```
 System.out.println();

 // Configure a scaling policy
 TargetTrackingScalingPolicyConfiguration targetTrackingScalingPolicyConfiguration
= new TargetTrackingScalingPolicyConfiguration()
   .withPredefinedMetricSpecification(
     new PredefinedMetricSpecification()
       .withPredefinedMetricType(MetricType.DynamoDBWriteCapacityUtilization))
   .withTargetValue(50.0)
   .withScaleInCooldown(60)
   .withScaleOutCooldown(60);

 // Create the scaling policy, based on your configuration
 PutScalingPolicyRequest pspRequest = new PutScalingPolicyRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID)
   .withPolicyName("MyScalingPolicy")
   .withPolicyType(PolicyType.TargetTrackingScaling)

.withTargetTrackingScalingPolicyConfiguration(targetTrackingScalingPolicyConfiguration);

 try {
  aaClient.putScalingPolicy(pspRequest);
 } catch (Exception e) {
  System.err.println("Unable to put scaling policy: ");
  System.err.println(e.getMessage());
 }

 // Verify that the scaling policy was created
 DescribeScalingPoliciesRequest dspRequest = new DescribeScalingPoliciesRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID);

 try {
  DescribeScalingPoliciesResult dspResult =
aaClient.describeScalingPolicies(dspRequest);
  System.out.println("DescribeScalingPolicies result: ");
  System.out.println(dspResult);
 } catch (Exception e) {
  e.printStackTrace();
  System.err.println("Unable to describe scaling policy: ");
  System.err.println(e.getMessage());
```

```
  }

 }

}
```

테이블에 대해 Application Auto Scaling 비활성화

다음 프로그램은 이전 프로세스를 반대로 수행합니다. Auto Scaling 정책을 제거한 후 확장 가능 목표의 등록을 해제합니다.

Java v2

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExceptio
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeregisterScalableTargetReques
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class DisableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <policyName>\s

            Where:
               tableId - The table Id value (for example, table/Music).\s
               policyName - The name of the policy (for example, $Music5-scaling-
policy).

            """;
        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        ApplicationAutoScalingClient appAutoScalingClient =
 ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
            .build();

        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
 ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        String tableId = args[0];
        String policyName = args[1];

        deletePolicy(appAutoScalingClient, policyName, tableWCUs, ns, tableId);
        verifyScalingPolicies(appAutoScalingClient, tableId, ns, tableWCUs);
        deregisterScalableTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
    }

    public static void deletePolicy(ApplicationAutoScalingClient
 appAutoScalingClient, String policyName, ScalableDimension tableWCUs,
 ServiceNamespace ns, String tableId) {
        try {
            DeleteScalingPolicyRequest delSPRequest =
 DeleteScalingPolicyRequest.builder()
```

```
                .policyName(policyName)
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deleteScalingPolicy(delSPRequest);
            System.out.println(policyName +" was deleted successfully.");

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the scaling policy was deleted
    public static void verifyScalingPolicies(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        DescribeScalingPoliciesRequest dscRequest =
DescribeScalingPoliciesRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceId(tableId)
            .build();

        DescribeScalingPoliciesResponse response =
appAutoScalingClient.describeScalingPolicies(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    public static void deregisterScalableTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        try {
            DeregisterScalableTargetRequest targetRequest =
DeregisterScalableTargetRequest.builder()
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deregisterScalableTarget(targetRequest);
            System.out.println("The scalable target was deregistered.");
```

```
        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    public static void verifyTarget(ApplicationAutoScalingClient
 appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
 tableWCUs) {
        DescribeScalableTargetsRequest dscRequest =
 DescribeScalableTargetsRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
 appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }
}
```

Java v1

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DeregisterScalableTargetRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;
```

```
public class DisableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = new
 AWSApplicationAutoScalingClient();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
  ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
  String resourceID = "table/TestTable";

  // Delete the scaling policy
  DeleteScalingPolicyRequest delSPRequest = new DeleteScalingPolicyRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID)
    .withPolicyName("MyScalingPolicy");

  try {
   aaClient.deleteScalingPolicy(delSPRequest);
  } catch (Exception e) {
   System.err.println("Unable to delete scaling policy: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scaling policy was deleted
  DescribeScalingPoliciesRequest descSPRequest = new
 DescribeScalingPoliciesRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID);

  try {
   DescribeScalingPoliciesResult dspResult =
 aaClient.describeScalingPolicies(descSPRequest);
   System.out.println("DescribeScalingPolicies result: ");
   System.out.println(dspResult);
  } catch (Exception e) {
   e.printStackTrace();
   System.err.println("Unable to describe scaling policy: ");
   System.err.println(e.getMessage());
  }
```

```
  System.out.println();

  // Remove the scalable target
  DeregisterScalableTargetRequest delSTRequest = new
 DeregisterScalableTargetRequest()
     .withServiceNamespace(ns)
     .withScalableDimension(tableWCUs)
     .withResourceId(resourceID);

  try {
   aaClient.deregisterScalableTarget(delSTRequest);
  } catch (Exception e) {
   System.err.println("Unable to deregister scalable target: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scalable target was removed
  DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
     .withServiceNamespace(ns)
     .withScalableDimension(tableWCUs)
     .withResourceIds(resourceID);

  try {
   DescribeScalableTargetsResult dsaResult =
 aaClient.describeScalableTargets(dscRequest);
   System.out.println("DescribeScalableTargets result: ");
   System.out.println(dsaResult);
   System.out.println();
  } catch (Exception e) {
   System.err.println("Unable to describe scalable target: ");
   System.err.println(e.getMessage());
  }

 }

}
```

## DynamoDB 예약 용량

Standard 테이블 클래스를 사용하는 프로비저닝된 용량 테이블의 경우 DynamoDB는 읽기 및 쓰기 용량에 대한 예약 용량을 구매할 수 있는 기능을 제공합니다. 예약 용량 구매는 계약 기간 동안 할인 요금으로 프로비저닝된 처리량 용량의 최소 금액을 지불하는 계약입니다.

> **Note**
>
> 복제한 쓰기 요청 단위(rWRU)에 대한 예약 용량은 구매할 수 없습니다. 예약 용량은 구매한 리전에만 적용됩니다. DynamoDB Standard-IA 테이블 클래스 또는 온디맨드 용량 모드를 사용하는 테이블에는 예약 용량을 사용할 수 없습니다.

예약 용량은 WCU 100개 또는 RCU 100개 할당으로 구매할 수 있습니다. 제공되는 최소 예약 용량은 용량 단위(읽기 또는 쓰기) 100입니다. DynamoDB 예약 용량은 1년 약정으로 제공되거나 일부 리전에서 3년 약정으로 제공됩니다. 1년 약정의 경우 표준 요금에서 최대 54%, 3년 약정의 경우 표준 요금에서 최대 77% 할인을 받을 수 있습니다. 구매 방법 및 시기에 대한 자세한 내용은 Amazon DynamoDB 예약 용량을 참조하세요.

DynamoDB 예약 용량을 구매하는 경우 일회성 부분 선납금을 지불하고 약정한 프로비저닝된 사용량에 대해 할인된 시간당 요금을 적용받습니다. 실제 사용량과 관계없이 약정된 프로비저닝된 전체 사용량에 대해 비용을 지불하므로 비용 절감 효과는 사용량과 밀접하게 연관됩니다. 구매한 예약 용량을 초과하여 프로비저닝하는 용량에는 프로비저닝된 용량 표준 요금이 청구됩니다. 사전에 읽기 및 쓰기 용량 단위를 예약하면 프로비저닝된 용량 비용을 크게 절감할 수 있습니다.

예약 용량을 판매 또는 취소하거나 다른 리전 또는 계정에 양도할 수 없습니다.

## DynamoDB 웜 처리량 이해

웜 처리량은 DynamoDB 테이블이 즉시 지원할 수 있는 읽기 및 쓰기 작업 수를 나타냅니다. 이러한 값은 기본적으로 모든 테이블 및 글로벌 보조 인덱스(GSI)에 사용할 수 있으며 과거 사용량에 따라 규모 조정된 양을 나타냅니다. 온디맨드 모드를 사용하는 경우 또는 프로비저닝된 처리량을 이러한 값으로 업데이트하는 경우 애플리케이션은 해당 값까지 요청을 즉시 실행할 수 있습니다.

DynamoDB는 사용량이 증가함에 따라 웜 처리량 값을 자동으로 조정합니다. 그러나 필요할 때 이러한 값을 사전에 늘릴 수도 있습니다. 이는 제품 출시 또는 판매와 같은 예정된 피크 이벤트에 특히 유용합니다. DynamoDB 테이블에 대한 요청 속도가 10배, 100배 이상 증가할 수 있는 계획된 피크 이벤트의 경우 현재 웜 처리량이 예상 트래픽을 처리하기에 충분한지 평가할 수 있습니다. 그렇지 않은 경우 처리량 설정 또는 청구 모드를 변경하지 않고도 웜 처리량 값을 늘릴 수 있습니다. 이 프로세스를 테이블

사전 워밍이라고 하며, 이를 통해 테이블이 즉시 지원할 수 있는 기준을 설정할 수 있습니다. 이렇게 하면 트래픽이 발생하는 순간부터 애플리케이션이 더 높은 요청 속도를 처리할 수 있습니다.

읽기 작업, 쓰기 작업 또는 둘 다에 대한 웜 처리량 값을 늘릴 수 있습니다. 신규 및 기존 단일 리전 테이블, 글로벌 테이블 및 GSI에 대해 이 값을 늘릴 수 있습니다. 글로벌 테이블의 경우 이 기능은 버전 2019.11.21(현재)에서 사용할 수 있으며 설정한 웜 처리량은 글로벌 테이블의 모든 복제본 테이블에 자동으로 적용됩니다. 사전 워밍할 수 있는 DynamoDB 테이블 수에는 제한이 없습니다. 사전 워밍을 완료하는 시간은 설정한 값과 테이블 또는 인덱스의 크기에 따라 달라집니다. 동시에 사전 워밍 요청을 제출할 수 있으며 이러한 요청은 테이블 작업을 방해하지 않습니다. 해당 리전의 계정에 대한 테이블 또는 인덱스 할당량 한도까지 테이블을 사전 워밍할 수 있습니다. Service Quotas 콘솔을 사용하여 현재 한도를 확인하고 필요한 경우 늘리세요.

웜 처리량 값은 기본적으로 모든 테이블과 보조 인덱스에 무료로 사용할 수 있습니다. 그러나 이러한 기본 웜 처리량 값을 사전에 늘려 테이블을 사전 워밍하는 경우 해당 요청에 대한 요금이 청구됩니다. 자세한 내용은 Amazon DynamoDB 요금을 참조하세요.

웜 처리량에 대한 자세한 내용은 아래 항목을 참조하세요.

주제

- DynamoDB 테이블의 현재 웜 처리량 확인
- 기존 DynamoDB 테이블의 웜 처리량 증가
- 웜 처리량이 더 높은 새 DynamoDB 테이블 만들기
- 다양한 시나리오에서 DynamoDB 웜 처리량 이해

## DynamoDB 테이블의 현재 웜 처리량 확인

다음 AWS CLI 및 AWS 콘솔 지침을 사용하여 테이블 또는 인덱스의 현재 웜 처리량 값을 확인합니다.

AWS Management Console

DynamoDB 콘솔을 사용하여 DynamoDB 테이블의 웜 처리량을 확인하려면:

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 테이블을 선택합니다.
3. 테이블 페이지에서 원하는 테이블을 선택합니다.
4. 현재 웜 처리량 값을 보려면 추가 설정을 선택합니다. 이 값은 초당 읽기 단위 및 초당 쓰기 단위로 표시됩니다.

AWS CLI

다음 AWS CLI 예제에서는 DynamoDB 테이블의 웜 처리량을 확인하는 방법을 보여줍니다.

1. DynamoDB 테이블에서 `describe-table` 작업을 실행합니다.

```
aws dynamodb describe-table --table-name GameScores
```

2. 아래와 비슷한 응답을 받게 됩니다. `WarmThroughput` 설정은 `ReadUnitsPerSecond` 및 `WriteUnitsPerSecond`로 표시됩니다. `Status`는 웜 처리량 값이 업데이트될 때는 `UPDATING`, 새 웜 처리량 값이 설정되었을 때는 `ACTIVE`가 됩니다.

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": 1726128388.729,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
```

```
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST",
            "LastUpdateToPayPerRequestDateTime": 1726128388.729
        },
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 0,
                    "WriteCapacityUnits": 0
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 12000,
                    "WriteUnitsPerSecond": 4000,
                    "Status": "ACTIVE"
```

```
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12000,
            "WriteUnitsPerSecond": 4000,
            "Status": "ACTIVE"
        }
    }
}
```

## 기존 DynamoDB 테이블의 웜 처리량 증가

DynamoDB 테이블의 현재 웜 처리량 값을 확인한 후 다음 단계를 통해 업데이트할 수 있습니다.

AWS Management Console

DynamoDB 콘솔을 사용하여 DynamoDB 테이블의 웜 처리량 값을 확인하려면:

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 테이블을 선택합니다.
3. 테이블 페이지에서 원하는 테이블을 선택합니다.
4. 웜 처리량 필드에서 편집을 선택합니다.
5. 웜 처리량 편집 페이지에서 웜 처리량 증가를 선택합니다.
6. 초당 읽기 단위와 초당 쓰기 단위를 조정합니다. 이 두 설정은 테이블이 즉시 처리할 수 있는 처리량을 정의합니다.
7. 저장을 선택합니다.
8. 요청이 처리를 완료하면 초당 읽기 단위와 초당 쓰기 단위가 웜 처리량 필드에 업데이트됩니다.

> **Note**
>
> 웜 처리량 값을 업데이트하는 것은 비동기 작업입니다. 업데이트가 완료되면 `Status`가 `ACTIVE`에서 `UPDATING`으로 변경됩니다.

AWS CLI

다음 AWS CLI 예제에서는 DynamoDB 테이블의 웜 처리량 값을 업데이트하는 방법을 보여줍니다.

1. DynamoDB 테이블에서 `update-table` 작업을 실행합니다.

```
aws dynamodb update-table \
    --table-name GameScores \
    --warm-throughput ReadUnitsPerSecond=12345,WriteUnitsPerSecond=4567 \
    --global-secondary-index-updates \
        "[
            {
                \"Update\": {
                    \"IndexName\": \"GameTitleIndex\",
                    \"WarmThroughput\": {
                        \"ReadUnitsPerSecond\": 88,
                        \"WriteUnitsPerSecond\": 77
                    }
                }
            }
        ]" \
    --region us-east-1
```

2. 아래와 비슷한 응답을 받게 됩니다. `WarmThroughput` 설정은 `ReadUnitsPerSecond` 및 `WriteUnitsPerSecond`로 표시됩니다. `Status`는 웜 처리량 값이 업데이트될 때는 `UPDATING`, 새 웜 처리량 값이 설정되었을 때는 `ACTIVE`가 됩니다.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
```

```
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": 1730242189.965,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 20,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
```

```
                    "ReadCapacityUnits": 50,
                    "WriteCapacityUnits": 25
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 50,
                    "WriteUnitsPerSecond": 25,
                    "Status": "UPDATING"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12300,
            "WriteUnitsPerSecond": 4500,
            "Status": "UPDATING"
        }
    }
}
```

### AWS SDK

다음 SDK 예제에서는 DynamoDB 테이블의 웜 처리량 값을 업데이트하는 방법을 보여줍니다.

#### Java

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndexUpdate;
import
 software.amazon.awssdk.services.dynamodb.model.UpdateGlobalSecondaryIndexAction;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

...
public static WarmThroughput buildWarmThroughput(final Long readUnitsPerSecond,
                                                 final Long writeUnitsPerSecond) {
    return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
```

```
            .build();
}

public static void updateDynamoDBTable(DynamoDbClient ddb,
                                       String tableName,
                                       Long tableReadUnitsPerSecond,
                                       Long tableWriteUnitsPerSecond,
                                       String globalSecondaryIndexName,
                                       Long globalSecondaryIndexReadUnitsPerSecond,
                                       Long globalSecondaryIndexWriteUnitsPerSecond)
 {

    final WarmThroughput tableWarmThroughput =
 buildWarmThroughput(tableReadUnitsPerSecond, tableWriteUnitsPerSecond);
    final WarmThroughput gsiWarmThroughput =
 buildWarmThroughput(globalSecondaryIndexReadUnitsPerSecond,
 globalSecondaryIndexWriteUnitsPerSecond);

    final GlobalSecondaryIndexUpdate globalSecondaryIndexUpdate =
 GlobalSecondaryIndexUpdate.builder()
            .update(UpdateGlobalSecondaryIndexAction.builder()
                    .indexName(globalSecondaryIndexName)
                    .warmThroughput(gsiWarmThroughput)
                    .build()
            ).build();

    final UpdateTableRequest request = UpdateTableRequest.builder()
            .tableName(tableName)
            .globalSecondaryIndexUpdates(globalSecondaryIndexUpdate)
            .warmThroughput(tableWarmThroughput)
            .build();

    try {
        ddb.updateTable(request);
    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }

    System.out.println("Done!");
}
```

Python

```
from boto3 import resource
from botocore.exceptions import ClientError

def update_dynamodb_table_warm_throughput(table_name, table_read_units,
 table_write_units, gsi_name, gsi_read_units, gsi_write_units, region_name="us-
east-1"):
    """
    Updates the warm throughput of a DynamoDB table and a global secondary index.

    :param table_name: The name of the table to update.
    :param table_read_units: The new read units per second for the table's warm
 throughput.
    :param table_write_units: The new write units per second for the table's warm
 throughput.
    :param gsi_name: The name of the global secondary index to update.
    :param gsi_read_units: The new read units per second for the GSI's warm
 throughput.
    :param gsi_write_units: The new write units per second for the GSI's warm
 throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    """
    try:
        ddb = resource('dynamodb', region_name)

        # Update the table's warm throughput
        table_warm_throughput = {
            "ReadUnitsPerSecond": table_read_units,
            "WriteUnitsPerSecond": table_write_units
        }

        # Update the global secondary index's warm throughput
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_read_units,
            "WriteUnitsPerSecond": gsi_write_units
        }

        # Construct the global secondary index update
        global_secondary_index_update = [
            {
                "Update": {
                    "IndexName": gsi_name,
                    "WarmThroughput": gsi_warm_throughput
```

```
                }
            }
        ]

        # Construct the update table request
        update_table_request = {
            "TableName": table_name,
            "GlobalSecondaryIndexUpdates": global_secondary_index_update,
            "WarmThroughput": table_warm_throughput
        }

        # Update the table
        ddb.update_table(**update_table_request)
        print("Table updated successfully!")
    except ClientError as e:
        print(f"Error updating table: {e}")
        raise e
```

Javascript

```
import { DynamoDBClient, UpdateTableCommand } from "@aws-sdk/client-dynamodb";

async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
            Update: {
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
```

```
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response: ${response}`);
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}
```

## 웜 처리량이 더 높은 새 DynamoDB 테이블 만들기

아래 단계에 따라 DynamoDB 테이블을 만들 때 웜 처리량 값을 조정할 수 있습니다. 이러한 단계는 글로벌 테이블 또는 보조 인덱스를 만들 때도 적용됩니다.

AWS Management Console

DynamoDB 테이블을 만들고 콘솔을 통해 웜 처리량 값을 조정하려면:

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 규칙 생성을 선택합니다.
3. 테이블 이름, 파티션 키 및 정렬 키(선택 사항)를 선택합니다.
4. 테이블 설정에서 설정 사용자 지정을 선택합니다.
5. 웜 처리량  필드에서 웜 처리량 증가를 선택합니다.
6. 초당 읽기 단위와 초당 쓰기 단위를 조정합니다. 이 두 설정은 테이블이 즉시 처리할 수 있는 최대 처리량을 정의합니다.
7. 나머지 테이블 세부 정보를 계속 추가한 다음 테이블 생성을 선택합니다.

AWS CLI

다음 AWS CLI 예제에서는 사용자 지정 웜 처리량 값을 사용하여 DynamoDB 테이블을 만드는 방법을 보여줍니다.

1. `create-table` 작업을 실행하여 다음 DynamoDB 테이블을 만듭니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S \
                            AttributeName=GameTitle,AttributeType=S \
                            AttributeName=TopScore,AttributeType=N  \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH
\"},
                                {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                \"Projection\":{
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 50,
                    \"WriteCapacityUnits\": 25
                },\"WarmThroughput\": {
                    \"ReadUnitsPerSecond\": 1987,
                    \"WriteUnitsPerSecond\": 543
                }
            }
        ]" \
    --warm-throughput ReadUnitsPerSecond=12345,WriteUnitsPerSecond=4567 \
    --region us-east-1
```

2. 아래와 비슷한 응답을 받게 됩니다. `WarmThroughput` 설정은 `ReadUnitsPerSecond` 및 `WriteUnitsPerSecond`로 표시됩니다. `Status`는 웜 처리량 값이 업데이트될 때는 `UPDATING`, 새 웜 처리량 값이 설정되었을 때는 `ACTIVE`가 됩니다.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": 1730241788.779,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 20,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
```

```
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 50,
                    "WriteCapacityUnits": 25
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 1987,
                    "WriteUnitsPerSecond": 543,
                    "Status": "UPDATING"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12345,
            "WriteUnitsPerSecond": 4567,
            "Status": "UPDATING"
        }
    }
}
```

AWS SDK

다음 SDK 예제에서는 사용자 지정 웜 처리량 값을 사용하여 DynamoDB 테이블을 생성하는 방법을 보여줍니다.

Java

```
import software.amazon.awscdk.services.dynamodb.ProjectionType;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;
...

public static WarmThroughput buildWarmThroughput(final Long readUnitsPerSecond,
                                                 final Long writeUnitsPerSecond) {
    return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
}
private static AttributeDefinition buildAttributeDefinition(final String
 attributeName,
                                                             final
 ScalarAttributeType scalarAttributeType) {
    return AttributeDefinition.builder()
            .attributeName(attributeName)
            .attributeType(scalarAttributeType)
            .build();
}
private static KeySchemaElement buildKeySchemaElement(final String attributeName,
                                                      final KeyType keyType) {
    return KeySchemaElement.builder()
            .attributeName(attributeName)
            .keyType(keyType)
            .build();
}
```

```
public static void createDynamoDBTable(DynamoDbClient ddb,
                                        String tableName,
                                        String partitionKey,
                                        String sortKey,
                                        String miscellaneousKeyAttribute,
                                        String nonKeyAttribute,
                                        Long tableReadCapacityUnits,
                                        Long tableWriteCapacityUnits,
                                        Long tableWarmReadUnitsPerSecond,
                                        Long tableWarmWriteUnitsPerSecond,
                                        String globalSecondaryIndexName,
                                        Long globalSecondaryIndexReadCapacityUnits,
                                        Long globalSecondaryIndexWriteCapacityUnits,
                                        Long
 globalSecondaryIndexWarmReadUnitsPerSecond,
                                        Long
 globalSecondaryIndexWarmWriteUnitsPerSecond) {

   // Define the table attributes
   final AttributeDefinition partitionKeyAttribute =
 buildAttributeDefinition(partitionKey, ScalarAttributeType.S);
   final AttributeDefinition sortKeyAttribute = buildAttributeDefinition(sortKey,
 ScalarAttributeType.S);
   final AttributeDefinition miscellaneousKeyAttributeDefinition =
 buildAttributeDefinition(miscellaneousKeyAttribute, ScalarAttributeType.N);
   final AttributeDefinition[] attributeDefinitions = {partitionKeyAttribute,
 sortKeyAttribute, miscellaneousKeyAttributeDefinition};

   // Define the table key schema
   final KeySchemaElement partitionKeyElement = buildKeySchemaElement(partitionKey,
 KeyType.HASH);
   final KeySchemaElement sortKeyElement = buildKeySchemaElement(sortKey,
 KeyType.RANGE);
   final KeySchemaElement[] keySchema = {partitionKeyElement, sortKeyElement};

   // Define the provisioned throughput for the table
   final ProvisionedThroughput provisionedThroughput =
 ProvisionedThroughput.builder()
           .readCapacityUnits(tableReadCapacityUnits)
           .writeCapacityUnits(tableWriteCapacityUnits)
           .build();

   // Define the Global Secondary Index (GSI)
```

```
   final KeySchemaElement globalSecondaryIndexPartitionKeyElement =
buildKeySchemaElement(sortKey, KeyType.HASH);
   final KeySchemaElement globalSecondaryIndexSortKeyElement =
buildKeySchemaElement(miscellaneousKeyAttribute, KeyType.RANGE);
   final KeySchemaElement[] gsiKeySchema =
{globalSecondaryIndexPartitionKeyElement, globalSecondaryIndexSortKeyElement};

   final Projection gsiProjection = Projection.builder()
           .projectionType(String.valueOf(ProjectionType.INCLUDE))
           .nonKeyAttributes(nonKeyAttribute)
           .build();
   final ProvisionedThroughput gsiProvisionedThroughput =
ProvisionedThroughput.builder()
           .readCapacityUnits(globalSecondaryIndexReadCapacityUnits)
           .writeCapacityUnits(globalSecondaryIndexWriteCapacityUnits)
           .build();
   // Define the warm throughput for the Global Secondary Index (GSI)
   final WarmThroughput gsiWarmThroughput =
buildWarmThroughput(globalSecondaryIndexWarmReadUnitsPerSecond,
globalSecondaryIndexWarmWriteUnitsPerSecond);
   final GlobalSecondaryIndex globalSecondaryIndex = GlobalSecondaryIndex.builder()
           .indexName(globalSecondaryIndexName)
           .keySchema(gsiKeySchema)
           .projection(gsiProjection)
           .provisionedThroughput(gsiProvisionedThroughput)
           .warmThroughput(gsiWarmThroughput)
           .build();

   // Define the warm throughput for the table
   final WarmThroughput tableWarmThroughput =
buildWarmThroughput(tableWarmReadUnitsPerSecond, tableWarmWriteUnitsPerSecond);

   final CreateTableRequest request = CreateTableRequest.builder()
           .tableName(tableName)
           .attributeDefinitions(attributeDefinitions)
           .keySchema(keySchema)
           .provisionedThroughput(provisionedThroughput)
           .globalSecondaryIndexes(globalSecondaryIndex)
           .warmThroughput(tableWarmThroughput)
           .build();

   CreateTableResponse response = ddb.createTable(request);
   System.out.println(response);
```

```
}
```

## Python

```
from boto3 import resource
from botocore.exceptions import ClientError

def create_dynamodb_table_warm_throughput(table_name, partition_key,
 sort_key, misc_key_attr, non_key_attr, table_provisioned_read_units,
 table_provisioned_write_units, table_warm_reads, table_warm_writes, gsi_name,
 gsi_provisioned_read_units, gsi_provisioned_write_units, gsi_warm_reads,
 gsi_warm_writes, region_name="us-east-1"):
    """
    Creates a DynamoDB table with a warm throughput setting configured.

    :param table_name: The name of the table to be created.
    :param partition_key: The partition key for the table being created.
    :param sort_key: The sort key for the table being created.
    :param misc_key_attr: A miscellaneous key attribute for the table being created.
    :param non_key_attr: A non-key attribute for the table being created.
    :param table_provisioned_read_units: The newly created table's provisioned read
 capacity units.
    :param table_provisioned_write_units: The newly created table's provisioned
 write capacity units.
    :param table_warm_reads: The read units per second setting for the table's warm
 throughput.
    :param table_warm_writes: The write units per second setting for the table's
 warm throughput.
    :param gsi_name: The name of the Global Secondary Index (GSI) to be created on
 the table.
    :param gsi_provisioned_read_units: The configured Global Secondary Index (GSI)
 provisioned read capacity units.
    :param gsi_provisioned_write_units: The configured Global Secondary Index (GSI)
 provisioned write capacity units.
    :param gsi_warm_reads: The read units per second setting for the Global
 Secondary Index (GSI)'s warm throughput.
    :param gsi_warm_writes: The write units per second setting for the Global
 Secondary Index (GSI)'s warm throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    """
    try:
        ddb = resource('dynamodb', region_name)
```

```
        # Define the table attributes
        attribute_definitions = [
            { "AttributeName": partition_key, "AttributeType": "S" },
            { "AttributeName": sort_key, "AttributeType": "S" },
            { "AttributeName": misc_key_attr, "AttributeType": "N" }
        ]

        # Define the table key schema
        key_schema = [
            { "AttributeName": partition_key, "KeyType": "HASH" },
            { "AttributeName": sort_key, "KeyType": "RANGE" }
        ]

        # Define the provisioned throughput for the table
        provisioned_throughput = {
            "ReadCapacityUnits": table_provisioned_read_units,
            "WriteCapacityUnits": table_provisioned_write_units
        }

        # Define the global secondary index
        gsi_key_schema = [
            { "AttributeName": sort_key, "KeyType": "HASH" },
            { "AttributeName": misc_key_attr, "KeyType": "RANGE" }
        ]
        gsi_projection = {
            "ProjectionType": "INCLUDE",
            "NonKeyAttributes": [non_key_attr]
        }
        gsi_provisioned_throughput = {
            "ReadCapacityUnits": gsi_provisioned_read_units,
            "WriteCapacityUnits": gsi_provisioned_write_units
        }
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_warm_reads,
            "WriteUnitsPerSecond": gsi_warm_writes
        }
        global_secondary_indexes = [
            {
                "IndexName": gsi_name,
                "KeySchema": gsi_key_schema,
                "Projection": gsi_projection,
                "ProvisionedThroughput": gsi_provisioned_throughput,
                "WarmThroughput": gsi_warm_throughput
            }
```

```
        ]

        # Define the warm throughput for the table
        warm_throughput = {
            "ReadUnitsPerSecond": table_warm_reads,
            "WriteUnitsPerSecond": table_warm_writes
        }

        # Create the DynamoDB client and create the table
        response = ddb.create_table(
            TableName=table_name,
            AttributeDefinitions=attribute_definitions,
            KeySchema=key_schema,
            ProvisionedThroughput=provisioned_throughput,
            GlobalSecondaryIndexes=global_secondary_indexes,
            WarmThroughput=warm_throughput
        )

        print(response)
    except ClientError as e:
        print(f"Error creating table: {e}")
        raise e
```

### Javascript

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";

async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
```

```
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
          ReadCapacityUnits: tableProvisionedReadUnits,
          WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
          ReadUnitsPerSecond: tableWarmReads,
          WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
          {
            IndexName: indexName,
            KeySchema: [
                { AttributeName: sortKey, KeyType: "HASH" },
                { AttributeName: miscKeyAttr, KeyType: "RANGE" },
            ],
            Projection: {
                ProjectionType: "INCLUDE",
                NonKeyAttributes: [nonKeyAttr],
            },
            ProvisionedThroughput: {
                ReadCapacityUnits: indexProvisionedReadUnits,
                WriteCapacityUnits: indexProvisionedWriteUnits,
            },
            WarmThroughput: {
                ReadUnitsPerSecond: indexWarmReads,
                WriteUnitsPerSecond: indexWarmWrites,
            },
          },
      ],
    });
```

```
    const response = await ddbClient.send(command);
    console.log(response);
  } catch (error) {
    console.error(`Error creating table: ${error}`);
    throw error;
  }
}
```

## 다양한 시나리오에서 DynamoDB 웜 처리량 이해

다음은 DynamoDB 웜 처리량으로 작업할 때 발생할 수 있는 몇 가지 시나리오입니다.

주제

- 웜 처리량 및 고르지 않은 액세스 패턴
- 프로비저닝된 테이블의 웜 처리량
- 온디맨드 테이블의 웜 처리량
- 최대 처리량이 구성된 온디맨드 테이블의 웜 처리량

### 웜 처리량 및 고르지 않은 액세스 패턴

테이블의 웜 처리량이 초당 30,000개의 읽기 단위와 초당 10,000개의 쓰기 단위인 경우에도 이러한 값에 도달하기 전에 읽기 또는 쓰기에 대한 스로틀링이 여전히 발생할 수 있습니다. 이는 핫 파티션 때문일 수 있습니다. DynamoDB는 사실상 무제한 처리량을 지원하도록 규모 조정을 계속할 수 있지만 각 개별 파티션은 초당 1,000개의 쓰기 단위와 초당 3,000개의 읽기 단위로 제한됩니다. 애플리케이션이 테이블 파티션의 작은 부분에 너무 많은 트래픽을 발생시키는 경우 테이블의 웜 처리량 값에 도달하기 전에도 스로틀링이 발생할 수 있습니다. 원활한 확장성을 보장하고 핫 파티션을 방지하려면 DynamoDB 모범 사례를 따르는 것이 좋습니다.

### 프로비저닝된 테이블의 웜 처리량

웜 처리량이 초당 30,000개의 읽기 단위와 초당 10,000개의 쓰기 단위이지만 현재 프로비저닝된 처리량이 4,000개의 RCU와 8,000개의 WCU인 프로비저닝된 테이블이 있다고 가정합니다. 프로비저닝된 처리량 설정을 업데이트하여 테이블의 프로비저닝된 처리량을 최대 RCU 30,000개 또는 WCU 10,000개까지 즉시 규모 조정할 수 있습니다. 프로비저닝된 처리량을 이러한 값 이상으로 늘리면 새로운 피크 처리량을 설정했기 때문에 웜 처리량이 자동으로 더 높은 새 값으로 조정됩니다. 예를 들어 프로비저닝된 처리량을 RCU 50,000개로 설정하면 웜 처리량이 초당 읽기 단위 50,000개로 증가합니다.

```
"ProvisionedThroughput":
```

```
    {
        "ReadCapacityUnits": 4000,
        "WriteCapacityUnits": 8000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

### 온디맨드 테이블의 웜 처리량

새로운 온디맨드 테이블은 초당 12,000개의 읽기 단위와 초당 4,000개의 쓰기 단위의 웜 처리량으로 시작됩니다. 테이블은 이러한 수준까지 지속적인 트래픽을 즉시 수용할 수 있습니다. 요청이 초당 12,000개의 읽기 단위 또는 초당 4,000개의 쓰기 단위를 초과하면 웜 처리량이 자동으로 더 높은 값으로 조정됩니다.

```
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 12000,
        "WriteUnitsPerSecond": 4000
    }
```

### 최대 처리량이 구성된 온디맨드 테이블의 웜 처리량

웜 처리량이 초당 30,000개의 읽기 단위이지만 최대 처리량이 읽기 요청 단위(RRU) 5,000개로 구성된 온디맨드 테이블이 있다고 가정해 보겠습니다. 이 시나리오에서 테이블의 처리량은 설정한 RRU 최대 5,000개로 제한됩니다. 이 최대값을 초과하는 모든 처리량 요청은 스로틀링됩니다. 그러나 애플리케이션 요구 사항에 따라 언제든지 테이블별 최대 처리량을 수정할 수 있습니다.

```
"OnDemandThroughput":
    {
        "MaxReadRequestUnits": 5000,
        "MaxWriteRequestUnits": 4000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

# DynamoDB 버스트 및 조정 용량

처리량 예외로 인한 제한을 최소화하기 위해 DynamoDB는 버스트 용량을 사용하여 사용량 급증을 처리합니다. DynamoDB는 조정 용량을 사용하여 고르지 않은 액세스 패턴을 지원합니다.

## 버스트 용량

DynamoDB는 버스트 용량을 통해 처리량 프로비저닝에서 어느 정도 유연성을 제공합니다. 사용 가능한 처리량을 완전히 사용하지 않을 때마다 DynamoDB는 나중에 사용량 급증을 처리하기 위해 처리량 버스트에 사용하지 않은 용량 일부를 예약해 둡니다. 원래는 조절되어야 하는 예상치 못한 읽기 또는 쓰기 요청도 버스트 용량으로 해결할 수 있습니다.

DynamoDB는 현재 최대 5분(300초)의 사용되지 않은 읽기 및 쓰기 용량을 유지합니다. 가끔 발생하는 읽기 또는 쓰기 작업의 버스트 중 이러한 추가 용량 단위는 빠르게, 테이블에 대해 정의한 초당 프로비저닝된 처리량 용량보다도 신속하게 소모될 수 있습니다.

DynamoDB는 또한 사전 통지 없이 배경 유지 관리와 다른 작업에 버스트 용량을 사용할 수도 있습니다.

향후 버스트 용량의 세부 정보가 변경될 수도 있습니다.

## 조정 용량

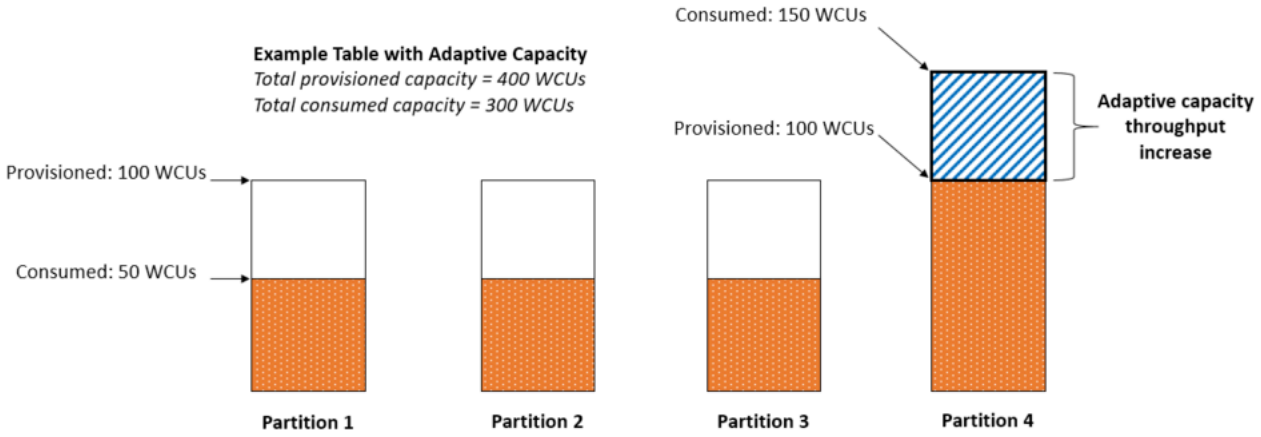
DynamoDB는 AWS 클라우드의 여러 서버에 저장되어 있는 파티션에 데이터를 자동으로 분산합니다. 항상 읽기와 쓰기 작업을 골고루 배포할 수 있는 것은 아닙니다. 데이터 액세스가 불균형할 때, '핫' 파티션은 다른 파티션보다 볼륨이 많은 읽기와 쓰기 트래픽을 받을 수 있습니다. 파티션의 읽기 및 쓰기 작업은 독립적으로 관리되므로 단일 파티션이 3,000개 이상의 읽기 작업 또는 1,000개 이상의 쓰기 작업을 수신할 경우 제한이 발생합니다. 조정 용량은 더 많은 트래픽을 받는 파티션의 처리량 용량을 자동으로 증가시킵니다.

균일하지 않은 액세스 패턴을 더 효과적으로 수용하기 위해 DynamoDB 조정 용량을 사용하면 트래픽이 테이블의 총 프로비저닝된 용량이나 파티션 최대 용량을 초과하지 않을 경우 애플리케이션에서는 조절 없이 핫 파티션에 계속 읽기 및 쓰기 작업을 수행할 수 있습니다. 조정 용량은 더 많은 트래픽을 받는 파티션의 처리량 용량을 즉시 자동으로 증가시킵니다.

다음 다이어그램은 조정 용량 작동 방식을 설명합니다. 예제 테이블은 4개의 파티션에 균일하게 공유된 400WCU로 프로비저닝되어 있어 각 파티션은 초당 최대 100WCU를 유지할 수 있습니다. 파티션 1, 2 및 3은 각각 50WCU/초의 쓰기 트래픽을 수신합니다. 파티션 4는 150WCU/초를 수신합니다. 이 핫

파티션은 사용되지 않은 버스트 용량이 아직 있어도 쓰기 트래픽을 수락할 수 있지만, 결국 100WCU/초를 초과하는 트래픽을 제한합니다.

DynamoDB 조정 용량은 파티션 4의 용량을 늘리는 것으로 대응하므로 해당 파티션이 제한되지 않고 150WCU/초의 높은 워크로드를 유지할 수 있습니다.




조정 용량은 모든 DynamoDB 테이블에 대해 추가 비용 없이 자동으로 활성화됩니다. 조정 용량을 명시적으로 활성화하거나 비활성화할 필요가 없습니다.

### 자주 액세스하는 항목 분리

애플리케이션 트래픽이 하나 이상의 항목으로 너무 많이 이동하는 경우 자주 액세스하는 항목이 동일한 파티션에 상주하지 않도록 조정 용량이 파티션 균형을 재조정합니다. 자주 액세스하는 항목을 분리하면 워크로드가 단일 파티션의 처리량 할당량을 초과하여 요청이 조절될 가능성이 줄어듭니다. 항목 컬렉션이 정렬 키의 일시적인 증가 또는 감소로 추적되는 트래픽이 아닌 한, 항목 컬렉션을 정렬 키를 통해 세그먼트로 나눌 수도 있습니다.

애플리케이션이 단일 항목에 지속적으로 높은 트래픽을 발생시키는 경우, 자주 액세스하는 단일 항목만 파티션에 포함되도록 조정 용량이 데이터를 리밸런싱할 수 있습니다. 이 경우 DynamoDB는 파티션에 최대 3,000RCU 및 1,000WCU 처리량을 단일 항목의 프라이머리 키에 전달할 수 있습니다. 조정 용량은 테이블에 로컬 보조 인덱스가 있는 경우 테이블의 여러 파티션에 항목 컬렉션을 분할하지 않습니다.

## DynamoDB에서 용량 모드 전환 시 고려 사항

DynamoDB 테이블을 생성할 때 온디맨드 또는 프로비저닝된 용량 모드를 선택해야 합니다.

테이블은 언제든지 온디맨드 모드에서 프로비저닝된 용량 모드로 전환할 수 있습니다. 용량 모드 간에 여러 번 전환하는 경우 다음 조건이 적용됩니다.

- 온디맨드 모드에서 새로 생성된 테이블은 언제든지 프로비저닝된 용량 모드로 전환할 수 있습니다. 하지만 테이블 생성 타임스탬프 이후 24시간이 지난 뒤에야 온디맨드 모드로 다시 전환할 수 있습니다.
- 온디맨드 모드의 기존 테이블은 언제든지 프로비저닝된 용량 모드로 전환할 수 있습니다. 하지만 온디맨드로의 전환을 나타내는 마지막 타임스탬프가 발생한 지 24시간이 지난 후에야 다시 온디맨드 모드로 전환할 수 있습니다.

주제


- 프로비저닝된 용량 모드에서 온디맨드 용량 모드로 전환
- 온디맨드 용량 모드에서 프로비저닝된 용량 모드로 전환


## 프로비저닝된 용량 모드에서 온디맨드 용량 모드로 전환

프로비저닝된 모드에서는 예상 애플리케이션 요구 사항에 따라 읽기 및 쓰기 용량을 설정합니다. 테이블을 프로비저닝된 모드에서 온디맨드 모드로 업데이트할 경우 애플리케이션에서 수행할 것으로 예상되는 읽기 및 쓰기 처리량을 지정할 필요가 없습니다. DynamoDB 온디맨드는 읽기 및 쓰기 요청에 대해 요청당 지불 요금이 부과되므로 사용하는 만큼에 대해서만 비용을 지불하면 됩니다. 따라서 비용과 성능의 균형을 쉽게 맞출 수 있습니다. 선택적으로, 개별 온디맨드 테이블 및 연결된 글로벌 보조 인덱스의 최대 읽기 또는 쓰기(또는 둘 다) 처리량을 구성하여 비용과 사용량을 제한할 수도 있습니다. 특정 테이블 또는 인덱스의 최대 처리량 설정에 대한 자세한 내용은 온디맨드 테이블의 최대 처리량 섹션을 참조하세요.

프로비저닝된 용량 모드에서 온디맨드 용량 모드로 전환할 경우 DynamoDB에서 테이블 및 파티션의 구조가 다양하게 변경됩니다. 이 프로세스는 몇 분 정도 걸릴 수 있습니다. 전환 기간 동안 테이블은 이전에 프로비저닝된 쓰기 용량 단위 및 읽기 용량 단위량과 일치하는 처리량을 제공합니다.

### 온디맨드 용량 모드의 초기 처리량

최근에 처음으로 기존 테이블을 온디맨드 용량 모드로 전환한 경우, 이전에 온디맨드 용량 모드를 사용하여 테이블에서 트래픽을 처리한 적이 없더라도 테이블의 이전 피크 설정은 다음과 같습니다.

다음은 가능한 시나리오의 예입니다.

- 프로비저닝된 테이블이 4,000WCU 및 12,000RCU 미만으로 구성되었고 이전에 추가로 프로비저닝된 적이 없습니다. 이 테이블이 처음 온디맨드로 전환되면 DynamoDB는 적어도 초당 4,000개의 쓰기 단위와 초당 12,000개의 읽기 단위를 유지할 수 있도록 테이블을 즉시 스케일 아웃합니다.

- 프로비저닝된 테이블이 8,000WCU 및 24,000RCU로 구성되었습니다. 이 테이블이 온디맨드로 전환되면 언제든지 적어도 초당 8,000개의 쓰기 단위와 초당 24,000개의 읽기 단위를 계속해서 유지할 수 있습니다.
- 8,000WCU와 24,000RCU로 구성된 프로비저닝된 테이블이 일정 기간 동안 초당 6,000개의 쓰기 단위와 초당 18,000개의 읽기 단위를 소비했습니다. 이 테이블이 온디맨드로 전환되면 적어도 초당 8,000개의 쓰기 단위와 초당 24,000개의 읽기 단위를 계속해서 유지할 수 있습니다. 또한 이전 트래픽을 통해 테이블은 제한 없이 훨씬 더 높은 수준의 트래픽을 유지할 수 있습니다.
- 테이블이 이전에는 10,000WCU와 10,000RCU로 프로비저닝되었지만 현재는 10RCU와 10WCU로 프로비저닝되었습니다. 이 테이블이 온디맨드로 전환되면 적어도 초당 10,000개의 쓰기 단위와 초당 10,000개의 읽기 단위를 유지할 수 있습니다.

### Auto Scaling 설정

테이블을 프로비저닝된 모드에서 온디맨드 모드로 업데이트하는 경우:

- 콘솔을 사용 중인 경우 모든 Auto Scaling 설정(있는 경우)이 삭제됩니다.
- AWS CLI 또는 AWS SDK를 사용 중인 경우 모든 Auto Scaling 설정이 유지됩니다. 테이블을 프로비저닝된 청구 모드로 다시 업데이트할 때 이러한 설정을 적용할 수 있습니다.

## 온디맨드 용량 모드에서 프로비저닝된 용량 모드로 전환

온디맨드 용량 모드에서 프로비저닝된 용량 모드로 다시 전환할 경우 테이블이 온디맨드 용량 모드로 설정되었을 때 도달한 이전 피크와 일치하는 처리량을 제공합니다.

### 용량 관리

테이블을 온디맨드 모드에서 프로비저닝된 모드로 업데이트할 때 다음을 고려하세요.

- AWS CLI 또는 AWS SDK를 사용 중인 경우 Amazon CloudWatch로 사용 내역(`ConsumedWriteCapacityUnits` 및 `ConsumedReadCapacityUnits` 지표)을 조회하여 테이블 및 글로벌 보조 인덱스에 적절한 프로비저닝된 용량 설정을 선택하여 새로운 처리량 설정을 결정합니다.

> **Note**
>
> 글로벌 테이블을 프로비저닝 모드로 전환할 경우, 새 처리량 설정을 결정할 때 기본 테이블과 글로벌 보조 인덱스에 대한 모든 리전 복제본에서 최대 사용량을 조회하세요.

- 온디맨드 모드에서 프로비저닝 모드로 다시 전환하는 경우, 전환 중에 테이블 또는 인덱스 용량을 처리할 수 있을 만큼 충분히 높게 초기 프로비저닝 단위를 설정해야 합니다.

## Auto Scaling 관리

테이블을 온디맨드 모드에서 프로비저닝된 모드로 업데이트하는 경우:

- 콘솔을 사용 중인 경우 다음 기본값을 사용하여 Auto Scaling을 활성화하는 것이 좋습니다.
  - 목표 사용률: 70%
  - 최소 프로비저닝 용량: 5 단위
  - 최대 프로비저닝 용량: 리전 최대값
- AWS CLI 또는 SDK를 사용 중인 경우 이전 Auto Scaling 설정(있는 경우)이 유지됩니다.

# DynamoDB 및 AWS SDK를 사용한 프로그래밍

이 단원에서는 개발자 관련 주제를 다룹니다. 코드 예제를 실행하려면 이 개발자 안내서의 코드 예시 실행 단원을 참조하세요.

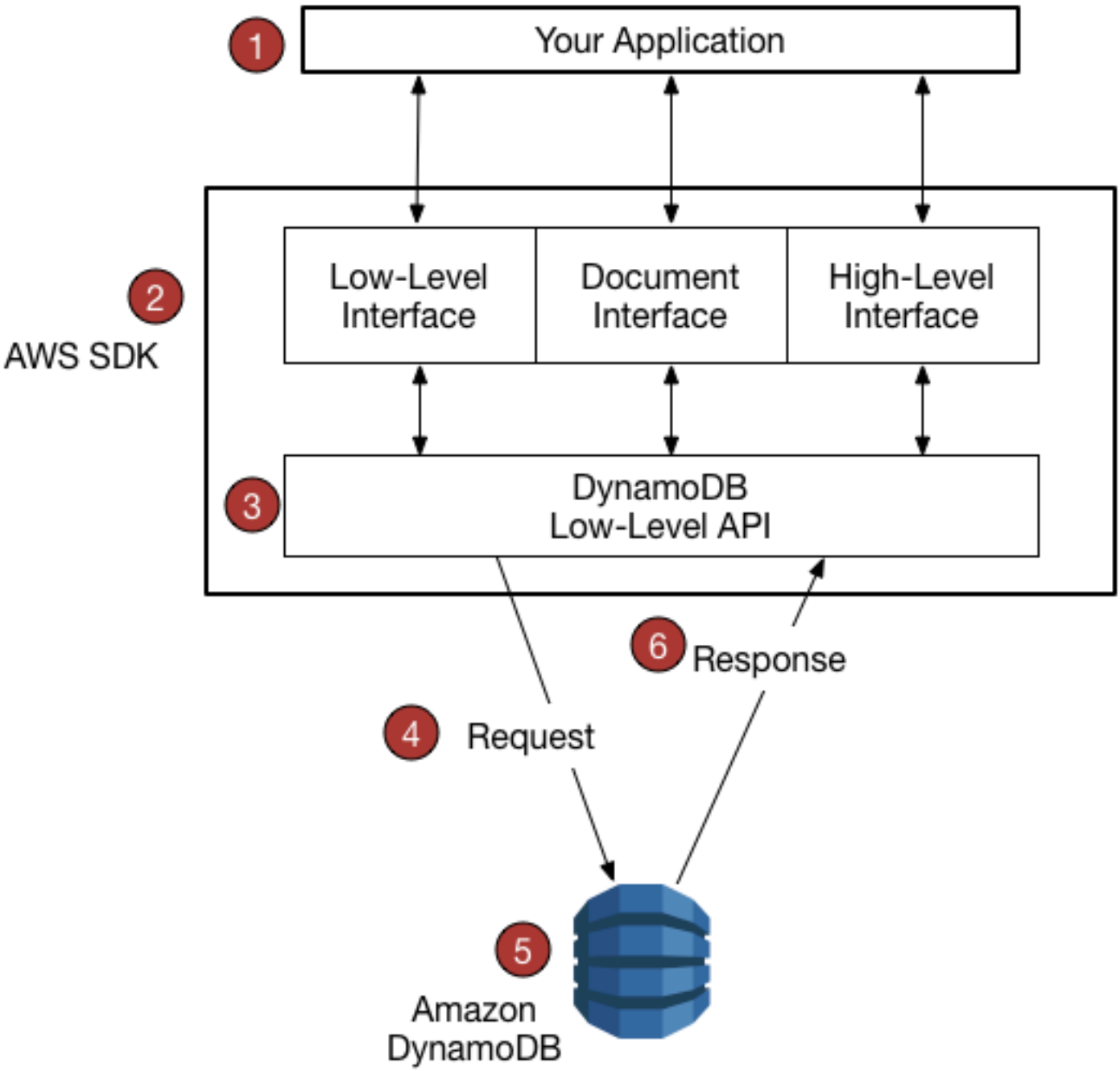
> **Note**
>
> 2017년 12월, AWS는 ATS(Amazon Trust Services)에서 발급된 보안 인증서를 사용하기 위해 모든 Amazon DynamoDB 엔드포인트를 마이그레이션하는 프로세스를 시작했습니다. 자세한 내용은 DynamoDB와의 SSL/TLS 연결 설정 문제 해결 섹션을 참조하세요.

주제


- DynamoDB에 대한 AWS SDK 지원 개요
- Python과 Boto3를 사용한 Amazon DynamoDB 프로그래밍
- JavaScript를 사용한 Amazon DynamoDB 프로그래밍
- AWS SDK for Java 2.x를 사용한 DynamoDB 프로그래밍
- DynamoDB 관련 오류 처리
- AWS SDK와 함께 DynamoDB 사용


## DynamoDB에 대한 AWS SDK 지원 개요

다음 다이어그램은 AWS SDK를 사용한 Amazon DynamoDB 애플리케이션 프로그래밍의 종합적 개요를 제공합니다.




1. 해당 프로그래밍 언어의 AWS SDK를 사용하여 애플리케이션을 작성합니다.
2. 각각의 AWS SDK는 DynamoDB 작업을 위한 하나 이상의 프로그래밍 인터페이스를 제공합니다. 사용 가능한 특정 인터페이스는 사용하는 프로그래밍 언어와 AWS SDK에 따라 달라집니다. 옵션에는 다음이 포함됩니다.
   - DynamoDB에서 작동하는 하위 수준 인터페이스
   - DynamoDB에서 작동하는 문서 인터페이스
   - DynamoDB에서 작동하는 객체 지속성 인터페이스

- 상위 수준 인터페이스

3. AWS SDK가 하위 수준 DynamoDB API와 함께 사용할 HTTP(S) 요청을 구성합니다.
4. AWS SDK가 DynamoDB 엔드포인트에 요청을 보냅니다.
5. DynamoDB가 요청을 실행합니다. 요청이 성공하면 DynamoDB는 HTTP 200 응답 코드(OK)를 반환합니다. 요청이 실패하면 DynamoDB는 HTTP 오류 코드와 오류 메시지를 반환합니다.
6. AWS SDK는 응답을 처리하여 이를 다시 애플리케이션에 전파합니다.

각각의 AWS SDK가 애플리케이션에 제공하는 중요 서비스에는 다음이 포함됩니다.

- HTTP(S) 요청 서식 설정 및 요청 파라미터 직렬화.
- 각 요청의 암호화 서명 생성.
- 요청을 DynamoDB 엔드포인트에 전달하고 DynamoDB에서 응답을 수신.
- 이러한 응답에서 결과 추출.
- 오류 발생 시 기본적 재시도 로직 구현.

이들 작업을 위한 코드는 작성할 필요가 없습니다.

> **Note**
>
> 설치 지침과 설명서 등 AWS SDK에 대한 자세한 내용은 Amazon Web Services용 도구를 참조하세요.

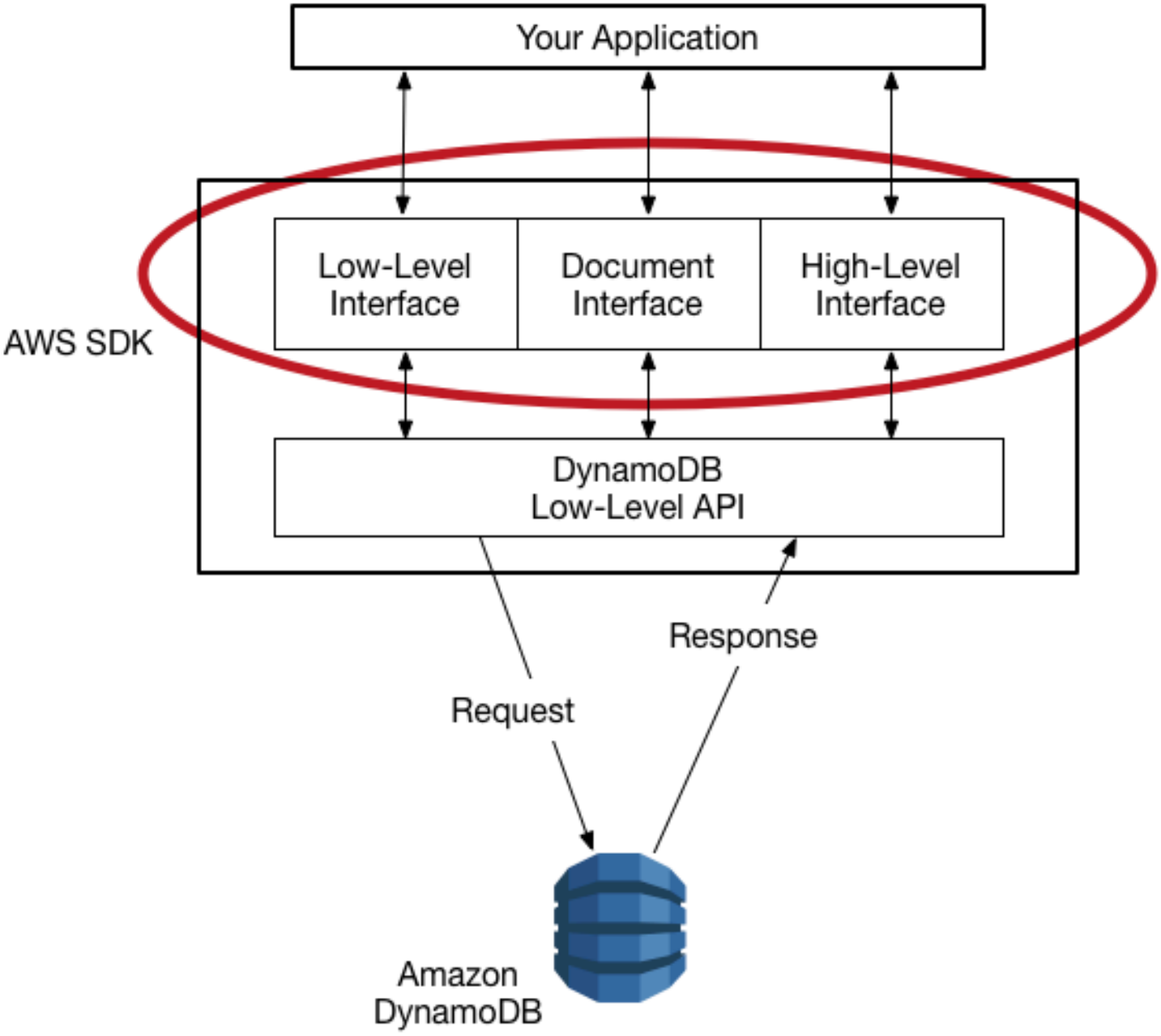
## SDK에서 AWS 계정 기반 엔드포인트 지원

AWS에서는 2024년 9월 4일 AWS SDK for Java V1을 시작으로 DynamoDB용 AWS 계정 기반 엔드포인트에 대한 SDK 지원을 롤아웃합니다. 이러한 새 엔드포인트는 AWS에서 높은 성능과 확장성을 보장하는 데 도움이 됩니다. 업데이트된 SDK는 `https://(account-id).ddb.(region).amazonaws.com` 형식을 가진 새 엔드포인트를 자동으로 사용합니다.

SDK 클라이언트의 단일 인스턴스를 사용하여 여러 계정에 요청을 보내는 경우 애플리케이션에서 연결을 재사용할 기회가 줄어듭니다. AWS에서는 SDK 클라이언트 인스턴스당 더 적은 수의 계정에 연결하도록 애플리케이션을 수정할 것을 권장합니다. 대안은 AWS SDK 및 도구 참조 가이드에 설명된 대로 `ACCOUNT_ID_ENDPOINT_MODE` 설정을 사용하여 SDK 클라이언트가 리전 엔드포인트를 계속 사용하도록 설정하는 것입니다.

# DynamoDB에서 작동하는 프로그래밍 인터페이스

각각의 AWS SDK는 Amazon DynamoDB 작업을 위한 하나 이상의 프로그래밍 인터페이스를 제공합니다. 이들 인터페이스는 단순한 하위 수준 DynamoDB 래퍼부터 객체 지향적인 지속성 계층까지 다양합니다. 사용 가능한 인터페이스는 사용하는 AWS SDK와 프로그래밍 언어에 따라 달라집니다.




다음 단원에서는 AWS SDK for Java를 예제로 삼아 사용 가능한 몇 가지 인터페이스를 중점적으로 살펴봅니다. (AWS SDK에 따라 일부 인터페이스는 사용할 수 없습니다.)

주제

- DynamoDB에서 작동하는 하위 수준 인터페이스
- DynamoDB에서 작동하는 문서 인터페이스
- DynamoDB에서 작동하는 객체 지속성 인터페이스


## DynamoDB에서 작동하는 하위 수준 인터페이스

모든 언어별 AWS SDK는 하위 수준 DynamoDB API 요청과 매우 비슷한 메서드를 사용하여 Amazon DynamoDB용 하위 수준 인터페이스를 제공합니다.

경우에 따라 데이터 형식 서술자(예: 문자열의 경우 `N`, 숫자의 경우 `S`)을(를) 사용하여 속성의 데이터 형식을 식별해야 합니다.

> **Note**
>
> 하위 수준 인터페이스는 모든 언어별 AWS SDK에서 제공됩니다.

다음 Java 프로그램은 AWS SDK for Java의 하위 수준 인터페이스를 사용합니다.

하위 수준 인터페이스 예제

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
```

```
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
 retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName, String
 key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
```

```
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem = ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

## DynamoDB에서 작동하는 문서 인터페이스

많은 AWS SDK는 테이블과 인덱스에서 데이터 영역 작업(생성, 읽기, 업데이트, 삭제)을 수행할 수 있는 문서 인터페이스를 제공합니다. 문서 인터페이스를 사용하면 데이터 형식 서술자를 지정할 필요가 없습니다. 데이터 형식은 데이터 자체의 의미론으로 암시됩니다. 이러한 AWS SDK는 JSON 문서를 기본 Amazon DynamoDB 데이터 형식으로, 또 그 반대로 쉽게 변환하는 메서드도 제공합니다.

> **Note**
>
> 문서 인터페이스는 AWS SDK for Java, .NET, Node.js, JavaScript SDK에서 사용할 수 있습니다.

다음 Java 프로그램은 AWS SDK for Java의 문서 인터페이스를 사용합니다. 이 프로그램은 `Music` 테이블을 나타내는 `Table` 객체를 생성한 다음 해당 객체에게 `GetItem`을 사용하여 노래를 검색하라고 요청합니다. 그런 다음 프로그램은 해당 노래가 발표된 연도를 인쇄합니다.

`software.amazon.dynamodb.document.DynamoDB` 클래스는 DynamoDB 문서 인터페이스를 구현합니다. DynamoDB가 어떻게 하위 수준 클라이언트(AmazonDynamoDB)를 둘러싼 래퍼 역할을 하는지에 유의하세요.

문서 인터페이스 예제

```
package com.amazonaws.codesamples.gsg;

import software.amazon.dynamodb.AmazonDynamoDB;
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.document.DynamoDB;
import software.amazon.dynamodb.document.GetItemOutcome;
import software.amazon.dynamodb.document.Table;

public class MusicDocumentDemo {

    public static void main(String[] args) {

        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");
        GetItemOutcome outcome = table.getItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Call Me Today");

        int year = outcome.getItem().getInt("Year");
        System.out.println("The song was released in " + year);

    }
}
```

## DynamoDB에서 작동하는 객체 지속성 인터페이스

일부 AWS SDK는 직접 데이터 영역 작업을 수행하지 않는 객체 지속성 인터페이스를 제공합니다. 그 대신 Amazon DynamoDB 테이블 및 인덱스의 항목을 나타내는 객체를 생성하고 이러한 객체와 상호 작용합니다. 이를 통해 데이터베이스 중심 코드가 아니라 객체 중심 코드를 만들 수 있습니다.

> **Note**
>
> 객체 지속성 인터페이스는 Java 및 .NET용 AWS SDK에서 제공됩니다. 자세한 내용은 DynamoDB의 DynamoDB에 대한 높은 수준의 프로그래밍 인터페이스를 참조하세요.

### 객체 지속성 인터페이스 예제

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;

/*
 * Before running this code example, create an Amazon DynamoDB table named Customer
 with these columns:
 *   - id - the id of the record that is the key. Be sure one of the id values is
 `id101`
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table. These
 values
 *                        need to be in the form of `YYYY-MM-DDTHH:mm:ssZ`, such as
 2022-07-11T00:00:00Z
 *
 * Also, ensure that you have set up your development environment, including your
 credentials.
```

```
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class EnhancedGetItem {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

        getItem(enhancedClient);
        ddb.close();
    }

    public static String getItem(DynamoDbEnhancedClient enhancedClient) {
        Customer result = null;
        try {
            DynamoDbTable<Customer> table = enhancedClient.table("Customer",
 TableSchema.fromBean(Customer.class));
            Key key = Key.builder()
                    .partitionValue("id101").sortValue("tred@noserver.com")
                    .build();

            // Get the item by using the key.
            result = table.getItem(
                    (GetItemEnhancedRequest.Builder requestBuilder) ->
 requestBuilder.key(key));
            System.out.println("******* The description value is " +
 result.getCustName());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return result.getCustName();
    }
```

```
}
```

## DynamoDB에 대한 높은 수준의 프로그래밍 인터페이스

AWS SDK는 애플리케이션에 Amazon DynamoDB에서 사용할 수 있는 하위 수준의 인터페이스를 제공합니다. 이러한 클라이언트 측 클래스와 메서드는 하위 수준 DynamoDB API에 직접적으로 대응합니다. 그러나 많은 개발자들이 복잡한 데이터 형식을 데이터베이스 테이블의 항목으로 매핑해야 할 때 연결 끊김 현상, 다시 말해 임피던스 불일치를 경험합니다. 하위 수준의 데이터베이스 인터페이스를 사용하는 개발자는 객체 데이터를 읽거나 데이터베이스 테이블에 쓰기 위한 메서드를 작성해야 하며 그 반대의 경우도 마찬가지입니다. 각 객체 유형 및 데이터베이스 테이블 조합에 필요한 추가 코드의 양은 엄청날 수 있습니다.

Java 및 .NET용 AWS SDK는 개발을 간소화하기 위해 높은 수준의 추상화를 갖춘 인터페이스를 추가로 제공합니다. 높은 수준의 DynamoDB 인터페이스를 사용하면 프로그램의 객체와 해당 객체의 데이터를 저장하는 데이터베이스 테이블 간의 관계를 정의할 수 있습니다. 이러한 매핑을 정의한 후 `save`, `load` 또는 `delete`와 같은 간단한 객체 메서드를 호출하면 기본 하위 수준 DynamoDB 작업이 사용자를 대신하여 자동으로 호출됩니다. 이를 통해 데이터베이스 중심 코드가 아니라 객체 중심 코드를 만들 수 있습니다.

높은 수준의 DynamoDB 프로그래밍 인터페이스는 Java 및 .NET용 AWS SDK에서 제공됩니다.

Java

- Java 1.x: DynamoDBMapper
- Java 2.x: DynamoDB 고급형 클라이언트

.NET

- DynamoDB에서 .NET 문서 모델을 사용하여 작업
- .NET 객체 지속성 모델 및 DynamoDB로 작업

## Java 1.x: DynamoDBMapper

> **Note**
>
> SDK for Java에는 1.x와 2.x의 두 가지 버전이 있습니다. 1.x의 경우 2024년 1월 12일에 지원 종료가 발표되었습니다. 2025년 12월 31일에 지원이 종료될 예정입니다. 새 개발의 경우 2.x를 사용하는 것이 좋습니다.

AWS SDK for Java에서는 클라이언트 측 클래스를 Amazon DynamoDB 테이블로 매핑할 수 있는 `DynamoDBMapper` 클래스를 제공합니다. `DynamoDBMapper`를 사용하려면 코드에서 DynamoDB 테이블의 항목과 해당하는 객체 인스턴스 간의 관계를 정의합니다. `DynamoDBMapper` 클래스를 사용하면 항목에 대한 다양한 생성, 읽기, 업데이트 및 삭제(CRUD) 작업을 수행하고 테이블에 대한 쿼리 및 스캔을 실행할 수 있습니다.

주제

- DynamoDBMapper 클래스
- Java용 DynamoDBMapper에서 지원되는 데이터 형식
- DynamoDB에 사용되는 Java 주석
- DynamoDBMapper의 구성 설정(선택 사항)
- DynamoDB 및 버전 번호를 이용한 낙관적 잠금
- DynamoDB에서 임의 데이터 매핑
- DynamoDBMapper 예제

> **Note**
>
> `DynamoDBMapper` 클래스는 테이블 생성, 업데이트 또는 삭제를 허용하지 않습니다. 이러한 작업을 위해서는 대신 하위 수준 SDK for Java 인터페이스를 사용해야 합니다.

SDK for Java는 클래스를 테이블로 매핑할 수 있도록 일련의 주석 유형을 제공합니다. 예를 들어, `ProductCatalog` 테이블의 `Id`가 파티션 키인 경우

```
ProductCatalog(Id, ...)
```

아래 Java 코드와 같이 클라이언트 애플리케이션의 클래스를 ProductCatalog 테이블로 매핑할 수 있습니다. 이 코드는 CatalogItem이라는 POJO(Plain Old Java Object)를 정의합니다. 여기서는 주석을 사용하여 객체 필드를 DynamoDB 속성 이름에 매핑합니다.

Example

```
package com.amazonaws.codesamples;

import java.util.Set;

import software.amazon.dynamodb.datamodeling.DynamoDBAttribute;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIgnore;
import software.amazon.dynamodb.datamodeling.DynamoDBTable;

@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) {this.id = id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() {return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN; }

    @DynamoDBAttribute(attributeName="Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
    public String getSomeProp() { return someProp; }
```

```
    public void setSomeProp(String someProp) { this.someProp = someProp; }
}
```

위의 코드에서 @DynamoDBTable 주석은 CatalogItem 클래스를 ProductCatalog 테이블로 매핑합니다. 각 클래스 인스턴스는 항목 형태로 이 테이블에 저장할 수 있습니다. 클래스 정의에서 @DynamoDBHashKey 주석은 Id 속성을 기본 키로 매핑합니다.

기본적으로 클래스 속성은 테이블에서 동일한 이름의 속성으로 매핑됩니다. 예를 들어 Title 및 ISBN 속성은 테이블에서 동일한 이름의 속성으로 매핑됩니다.

DynamoDB 속성의 이름이 클래스에서 선언된 속성의 이름과 일치하는 경우 @DynamoDBAttribute 주석을 선택적으로 사용할 수 있습니다. 이러한 이름이 서로 다르면 이 주석을 attributeName 파라미터와 함께 사용하여 이 속성에 대응하는 DynamoDB 속성을 지정합니다.

위 예제에서는 속성 이름이 이전 단계에서 만든 테이블과 정확히 일치하면서 이 가이드의 다른 코드 예제에서도 일관된 속성 이름을 사용할 수 있도록 @DynamoDBAttribute 주석이 각 속성에 추가되었습니다.

클래스 정의는 테이블의 어떤 속성에도 매핑되지 않는 속성을 가질 수도 있습니다. @DynamoDBIgnore 주석을 추가하여 이러한 속성을 확인할 수 있습니다. 위 예제에서는 SomeProp 속성이 @DynamoDBIgnore 주석으로 표시되어 있습니다. CatalogItem 인스턴스를 테이블에 업로드할 때 DynamoDBMapper 인스턴스에는 SomeProp 속성이 포함되지 않습니다. 또한 테이블에서 항목을 가져오더라도 매퍼가 이 속성을 반환하지 않습니다.

매핑 클래스를 정의한 후에는 DynamoDBMapper 메서드를 사용하여 해당 클래스의 인스턴스를 Catalog 테이블의 해당 항목에 쓸 수 있습니다. 다음 코드 예제에서는 이 과정을 보여줍니다.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapper mapper = new DynamoDBMapper(client);

CatalogItem item = new CatalogItem();
item.setId(102);
item.setTitle("Book 102 Title");
item.setISBN("222-2222222222");
item.setBookAuthors(new HashSet<String>(Arrays.asList("Author 1", "Author 2")));
item.setSomeProp("Test");

mapper.save(item);
```

다음 코드 예제에서는 항목을 검색하고 일부 속성에 액세스하는 방법을 보여줍니다.

```
CatalogItem partitionKey = new CatalogItem();

partitionKey.setId(102);
DynamoDBQueryExpression<CatalogItem> queryExpression = new
 DynamoDBQueryExpression<CatalogItem>()
    .withHashKeyValues(partitionKey);

List<CatalogItem> itemList = mapper.query(CatalogItem.class, queryExpression);

for (int i = 0; i < itemList.size(); i++) {
    System.out.println(itemList.get(i).getTitle());
    System.out.println(itemList.get(i).getBookAuthors());
}
```

DynamoDBMapper는 Java에서 DynamoDB 데이터를 사용하는 직관적이고 자연스러운 방법을 제공합니다. 또한 낙관적 잠금, ACID 트랜잭션, 자동 생성된 파티션 키 및 정렬 키 값, 객체 버전 관리 등 기본적인 여러 기능을 제공합니다.

DynamoDBMapper 클래스

DynamoDBMapper 클래스는 Amazon DynamoDB API의 진입점입니다. DynamoDB 엔드포인트에 액세스하고 여러 테이블의 데이터에 액세스할 수 있습니다. 또한 항목에 대한 다양한 생성, 읽기, 업데이트 및 삭제(CRUD) 작업을 수행하고 테이블에 대한 쿼리 및 스캔을 실행할 수 있습니다. 이 클래스는 DynamoDB를 작업하기 위한 다음과 같은 메서드를 제공합니다.

해당 Javadoc 설명서는 AWS SDK for Java API 참조의 DynamoDBMapper를 참조하세요.

주제

- 저장
- 로드
- 삭제
- 쿼리
- queryPage
- 스캔
- scanPage
- parallelScan
- batchSave

- batchLoad
- batchDelete
- batchWrite
- transactionWrite
- transactionLoad
- count
- generateCreateTableRequest
- createS3Link
- getS3ClientCache

저장

지정한 객체를 테이블에 저장합니다. 여기서 저장하는 객체가 이 메서드에 유일하게 필요한 파라미터입니다. DynamoDBMapperConfig 객체를 사용하여 옵션으로 구성 파라미터를 입력할 수도 있습니다.

동일한 기본 키를 사용하는 항목이 없을 경우에는 이 메서드가 테이블에 새로운 항목을 생성합니다. 그리고, 동일한 기본 키를 사용하는 항목이 있을 경우에는 기존 항목을 업데이트합니다. 파티션 키와 정렬 키가 String 형식이고 @DynamoDBAutoGeneratedKey로 주석된 경우 초기화되지 않았으면 랜덤 UUID(Universally Unique Identifier)가 부여됩니다. @DynamoDBVersionAttribute 주석이 추가된 버전 필드는 1씩 증가합니다. 또한 버전 필드를 업데이트하거나 키를 생성하는 경우 작업 결과에 따라 전달되는 객체도 업데이트됩니다.

기본적으로 매핑된 클래스 속성에 해당되는 속성만 업데이트됩니다. 항목의 기존 추가 속성은 영향을 받지 않습니다. 하지만 SaveBehavior.CLOBBER를 지정할 경우 항목을 강제로 완전히 덮어쓸 수 있습니다.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER).build();

mapper.save(item, config);
```

버전 관리를 활성화하면 클라이언트 측 항목 버전과 서버 측 항목 버전이 일치해야 합니다. 하지만 SaveBehavior.CLOBBER 옵션을 사용하면 버전이 일치하지 않아도 괜찮습니다. 버전 관리에 대한 자세한 내용은 DynamoDB 및 버전 번호를 이용한 낙관적 잠금 섹션을 참조하십시오.

로드

테이블에서 항목을 가져옵니다. 검색할 항목의 기본 키를 제공해야 합니다. DynamoDBMapperConfig 객체를 사용하여 옵션으로 구성 파라미터를 입력할 수도 있습니다. 예를 들어 아래 Java 문과 같이 이 메서드를 실행하여 최신 항목 값만 가져오려면 옵션으로 strongly consistent read를 요청하면 됩니다.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT).build();

CatalogItem item = mapper.load(CatalogItem.class, item.getId(), config);
```

기본적으로 DynamoDB는 최종적으로 일관된 값을 갖는 항목을 반환하기 때문입니다. DynamoDB의 최종 일관성 모델에 대한 자세한 내용은 DynamoDB 읽기 일관성 단원을 참조하세요.

삭제

항목을 테이블에서 삭제합니다. 이때는 매핑된 클래스의 객체 인스턴스를 전달해야 합니다.

버전 관리를 활성화하면 클라이언트 측 항목 버전과 서버 측 항목 버전이 일치해야 합니다. 하지만 SaveBehavior.CLOBBER 옵션을 사용하면 버전이 일치하지 않아도 괜찮습니다. 버전 관리에 대한 자세한 내용은 DynamoDB 및 버전 번호를 이용한 낙관적 잠금 섹션을 참조하십시오.

쿼리

테이블 또는 보조 인덱스를 쿼리합니다.

포럼 스레드 회신이 저장되는 Reply 테이블이 있는 경우 스레드 제목마다 회신 수가 0개 또는 그 이상이 될 수 있습니다. Reply 테이블의 기본 키는 Id 및 ReplyDateTime 필드로 구성됩니다. 여기에서 Id는 파티션 키이고, ReplyDateTime은 기본 키의 정렬 키입니다.

```
Reply ( Id, ReplyDateTime, ... )
```

Reply 클래스와 DynamoDB의 해당 Reply 테이블 간 매핑을 생성한 경우 다음 Java 코드는 DynamoDBMapper를 사용하여 특정 스레드 제목에 대한 지난 2주 간 모든 회신을 찾습니다.

Example

```
String forumName = "&DDB;";
```

```
String forumSubject = "&DDB; Thread 1";
String partitionKey = forumName + "#" + forumSubject;

long twoWeeksAgoMilli = (new Date()).getTime() - (14L*24L*60L*60L*1000L);
Date twoWeeksAgo = new Date();
twoWeeksAgo.setTime(twoWeeksAgoMilli);
SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
String twoWeeksAgoStr = df.format(twoWeeksAgo);

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1", new AttributeValue().withS(partitionKey));
eav.put(":v2",new AttributeValue().withS(twoWeeksAgoStr.toString()));

DynamoDBQueryExpression<Reply> queryExpression = new DynamoDBQueryExpression<Reply>()
    .withKeyConditionExpression("Id = :v1 and ReplyDateTime > :v2")
    .withExpressionAttributeValues(eav);

List<Reply> latestReplies = mapper.query(Reply.class, queryExpression);
```

쿼리 결과 `Reply` 객체 컬렉션이 반환됩니다.

기본적으로 `query` 메서드는 "지연 로딩된(lazy-loaded)" 컬렉션을 반환합니다. 즉, 처음에는 결과 페이지를 하나만 반환하고, 필요에 따라 서비스를 호출하여 다음 페이지를 반환합니다. 일치하는 항목을 모두 가져오려면 `latestReplies` 컬렉션을 반복합니다.

컬렉션에서 `size()` 메서드를 호출하면 정확한 개수를 제공하기 위해 모든 결과가 로드됩니다. 이로 인해 프로비저닝된 처리량이 많이 사용될 수 있으며, 매우 큰 테이블에서는 JVM의 모든 메모리가 소모될 수도 있습니다.

인덱스를 쿼리하려면 먼저 인덱스를 매퍼 클래스로 모델링해야 합니다. `Reply` 테이블에는 PostedBy-Message-Index라는 글로벌 보조 인덱스가 있습니다. 이 인덱스의 파티션 키는 `PostedBy`이고, 정렬 키는 `Message`입니다. 이 인덱스에서 항목의 클래스 정의는 다음과 같을 것입니다.

```
@DynamoDBTable(tableName="Reply")
public class PostedByMessage {
    private String postedBy;
    private String message;

    @DynamoDBIndexHashKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "PostedBy")
    public String getPostedBy() { return postedBy; }
    public void setPostedBy(String postedBy) { this.postedBy = postedBy; }
```

```
    @DynamoDBIndexRangeKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "Message")
    public String getMessage() { return message; }
    public void setMessage(String message) { this.message = message; }

   // Additional properties go here.
}
```

@DynamoDBTable 주석은 이 인덱스가 Reply 테이블과 연결되어 있음을 나타냅니다. @DynamoDBIndexHashKey 주석은 인덱스의 파티션 키(PostedBy)를 표시하고, @DynamoDBIndexRangeKey는 인덱스의 정렬 키(Message)를 표시합니다.

이제 DynamoDBMapper를 사용하여 특정 사용자가 게시한 메시지 하위 집합을 검색하는 인덱스 쿼리를 실행할 수 있습니다. 테이블과 인덱스에 충돌하는 매핑이 없고 매핑이 이미 매퍼에서 만들어진 경우에는 인덱스 이름을 지정할 필요가 없습니다. 매퍼는 프라이머리 키와 정렬 키를 기반으로 추론합니다. 다음 코드에서는 글로벌 보조 인덱스를 쿼리합니다. 글로벌 보조 인덱스는 최종적 일관된 읽기는 지원하지만 강력히 일관된 읽기는 지원하지 않으므로 withConsistentRead(false)를 지정해야 합니다.

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1",  new AttributeValue().withS("User A"));
eav.put(":v2",  new AttributeValue().withS("DynamoDB"));

DynamoDBQueryExpression<PostedByMessage> queryExpression = new
 DynamoDBQueryExpression<PostedByMessage>()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false)
    .withKeyConditionExpression("PostedBy = :v1 and begins_with(Message, :v2)")
    .withExpressionAttributeValues(eav);

List<PostedByMessage> iList =  mapper.query(PostedByMessage.class, queryExpression);
```

쿼리 결과 PostedByMessage 객체 컬렉션이 반환됩니다.

queryPage

테이블 또는 보조 인덱스를 쿼리하고 일치하는 결과를 단일 페이지로 반환합니다. query 메서드에서 그랬듯이 이번에도 파티션 키 값을 비롯해 정렬 키 속성에 적용되는 쿼리 필터를 지정해야 합니다. 그러나 queryPage는 첫 번째 데이터 "페이지"만 반환합니다. 다시 말해서 1M를 넘지 않는 범위에서 데이터를 반환합니다.

### 스캔

전체 테이블 또는 보조 인덱스를 스캔합니다. FilterExpression를 지정하여 결과 집합을 필터링할 수도 있습니다(선택 사항).

포럼 스레드 회신이 저장되는 Reply 테이블이 있는 경우 스레드 제목마다 회신 수가 0개 또는 그 이상이 될 수 있습니다. Reply 테이블의 기본 키는 Id 및 ReplyDateTime 필드로 구성됩니다. 여기에서 Id는 파티션 키이고, ReplyDateTime은 기본 키의 정렬 키입니다.

```
Reply ( Id, ReplyDateTime, ... )
```

Java 클래스를 Reply 테이블로 매핑한 경우 DynamoDBMapper를 사용하여 테이블을 스캔할 수 있습니다. 예를 들어, 다음 Java 코드에서는 전체 Reply 테이블을 스캔하여 특정 연도의 회신만 반환합니다.

#### Example

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1", new AttributeValue().withS("2015"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("begins_with(ReplyDateTime,:v1)")
    .withExpressionAttributeValues(eav);

List<Reply> replies =  mapper.scan(Reply.class, scanExpression);
```

기본적으로 scan 메서드는 "지연 로딩된(lazy-loaded)" 컬렉션을 반환합니다. 즉, 처음에는 결과 페이지를 하나만 반환하고, 필요에 따라 서비스를 호출하여 다음 페이지를 반환합니다. 일치하는 항목을 모두 가져오려면 replies 컬렉션을 반복합니다.

컬렉션에서 size() 메서드를 호출하면 정확한 개수를 제공하기 위해 모든 결과가 로드됩니다. 이로 인해 프로비저닝된 처리량이 많이 사용될 수 있으며, 매우 큰 테이블에서는 JVM의 모든 메모리가 소모될 수도 있습니다.

인덱스를 스캔하려면 먼저 인덱스를 매퍼 클래스로 모델링해야 합니다. Reply 테이블에 PostedBy-Message-Index라는 글로벌 보조 인덱스가 있다고 가정합니다. 이 인덱스의 파티션 키는 PostedBy이고, 정렬 키는 Message입니다. 이 인덱스에 대한 매퍼 클래스는 쿼리 단원에 나와 있습니다. 이 클래스는 @DynamoDBIndexHashKey 및 @DynamoDBIndexRangeKey 주석을 사용하여 인덱스 파티션 키 및 정렬 키를 지정합니다.

다음 코드 예제에서는 PostedBy-Message-Index를 스캔합니다. 스캔 필터를 사용하지 않으므로 인덱스의 모든 항목이 반환됩니다.

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false);

    List<PostedByMessage> iList =  mapper.scan(PostedByMessage.class, scanExpression);
    Iterator<PostedByMessage> indexItems = iList.iterator();
```

### scanPage

테이블 또는 보조 인덱스를 스캔하고 일치하는 결과를 단일 페이지로 반환합니다. scan 메서드와 마찬가지로 FilterExpression을 지정하여 결과 집합을 필터링할 수도 있습니다(선택 사항). 그러나 scanPage는 첫 번째 데이터 '페이지'만 반환합니다. 다시 말해서 1MB를 넘지 않는 범위에서 데이터를 반환합니다.

### parallelScan

전체 테이블 또는 보조 인덱스를 병렬 방식으로 스캔합니다. 즉, 결과를 필터링할 스캔 표현식과 함께 테이블의 논리 세그먼트를 다수 지정합니다. parallelScan은 스캔 작업을 각 논리 세그먼트마다 하나씩 여러 작업자로 분할합니다. 그러면 작업자가 데이터를 병렬 방식으로 처리하여 결과를 반환합니다.

다음 Java 코드 예제에서는 Product 테이블에 대한 병렬 스캔을 수행합니다.

```
int numberOfThreads = 4;

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":n", new AttributeValue().withN("100"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("Price <= :n")
    .withExpressionAttributeValues(eav);

List<Product> scanResult = mapper.parallelScan(Product.class, scanExpression,
 numberOfThreads);
```

### batchSave

AmazonDynamoDB.batchWriteItem 메서드를 1회 이상 호출하여 객체를 하나 이상의 테이블에 저장합니다. 이 메서드는 트랜잭션을 보장하지는 않습니다.

다음 Java 코드에서는 두 항목(책)을 ProductCatalog 테이블에 저장합니다.

```
Book book1 = new Book();
book1.setId(901);
book1.setProductCategory("Book");
book1.setTitle("Book 901 Title");

Book book2 = new Book();
book2.setId(902);
book2.setProductCategory("Book");
book2.setTitle("Book 902 Title");

mapper.batchSave(Arrays.asList(book1, book2));
```

### batchLoad

기본 키를 사용하여 다수의 항목을 하나 이상의 테이블에서 가져옵니다.

다음 Java 코드에서는 두 개의 서로 다른 테이블에서 두 개의 항목을 검색합니다.

```
ArrayList<Object> itemsToGet = new ArrayList<Object>();

ForumItem forumItem = new ForumItem();
forumItem.setForumName("Amazon DynamoDB");
itemsToGet.add(forumItem);

ThreadItem threadItem = new ThreadItem();
threadItem.setForumName("Amazon DynamoDB");
threadItem.setSubject("Amazon DynamoDB thread 1 message text");
itemsToGet.add(threadItem);

Map<String, List<Object>> items = mapper.batchLoad(itemsToGet);
```

### batchDelete

AmazonDynamoDB.batchWriteItem 메서드를 1회 이상 호출하여 객체를 하나 이상의 테이블에서 삭제합니다. 이 메서드는 트랜잭션을 보장하지는 않습니다.

다음은 Java 코드에서는 ProductCatalog 테이블에서 두 항목(책)을 삭제합니다.

```
Book book1 = mapper.load(Book.class, 901);
Book book2 = mapper.load(Book.class, 902);
```

```
mapper.batchDelete(Arrays.asList(book1, book2));
```

### batchWrite

`AmazonDynamoDB.batchWriteItem` 메서드를 1회 이상 호출하여 객체를 하나 이상의 테이블에 저장하거나 삭제합니다. 이 메서드는 트랜잭션을 보장하거나 버전 관리를 지원하지 않습니다(조건부 업로드 또는 삭제).

다음 Java 코드에서는 새 항목을 `Forum` 테이블에 쓰고, 새 항목을 `Thread` 테이블에 쓰고, `ProductCatalog` 테이블에서 항목을 삭제합니다.

```
// Create a Forum item to save
Forum forumItem = new Forum();
forumItem.setName("Test BatchWrite Forum");

// Create a Thread item to save
Thread threadItem = new Thread();
threadItem.setForumName("AmazonDynamoDB");
threadItem.setSubject("My sample question");

// Load a ProductCatalog item to delete
Book book3 = mapper.load(Book.class, 903);

List<Object> objectsToWrite = Arrays.asList(forumItem, threadItem);
List<Book> objectsToDelete = Arrays.asList(book3);

mapper.batchWrite(objectsToWrite, objectsToDelete);
```

### transactionWrite

`AmazonDynamoDB.transactWriteItems` 메서드를 1회 호출하여 객체를 하나 이상의 테이블에 저장하거나 삭제합니다.

트랜잭션별 예외 목록을 보려면 TransactWriteItems 오류를 참조하세요.

DynamoDB 트랜잭션과 제공된 원자성, 일관성, 격리성 및 지속성(ACID) 보장에 대한 자세한 내용은 Amazon DynamoDB Transactions를 참조하세요.

> **Note**
>
> 이 메서드는 다음을 지원하지 않습니다.

- DynamoDBMapperConfig.SaveBehavior.

다음 Java 코드에서는 트랜잭션 형태로 각 Forum 및 Thread 테이블에 새 항목을 씁니다.

```
Thread s3ForumThread = new Thread();
s3ForumThread.setForumName("S3 Forum");
s3ForumThread.setSubject("Sample Subject 1");
s3ForumThread.setMessage("Sample Question 1");

Forum s3Forum = new Forum();
s3Forum.setName("S3 Forum");
s3Forum.setCategory("Amazon Web Services");
s3Forum.setThreads(1);

TransactionWriteRequest transactionWriteRequest = new TransactionWriteRequest();
transactionWriteRequest.addPut(s3Forum);
transactionWriteRequest.addPut(s3ForumThread);
mapper.transactionWrite(transactionWriteRequest);
```

transactionLoad

AmazonDynamoDB.transactGetItems 메서드를 1회 호출하여 객체를 하나 이상의 테이블에서 로드합니다.

트랜잭션별 예외 목록을 보려면 TransactGetItems 오류를 참조하세요.

DynamoDB 트랜잭션과 제공된 원자성, 일관성, 격리성 및 지속성(ACID) 보장에 대한 자세한 내용은 Amazon DynamoDB Transactions를 참조하세요.

다음 Java 코드에서는 트랜잭션 형태로 각 Forum 및 Thread 테이블에서 한 개의 항목을 로드합니다.

```
Forum dynamodbForum = new Forum();
dynamodbForum.setName("DynamoDB Forum");
Thread dynamodbForumThread = new Thread();
dynamodbForumThread.setForumName("DynamoDB Forum");

TransactionLoadRequest transactionLoadRequest = new TransactionLoadRequest();
transactionLoadRequest.addLoad(dynamodbForum);
transactionLoadRequest.addLoad(dynamodbForumThread);
mapper.transactionLoad(transactionLoadRequest);
```

count

지정한 스캔 표현식을 평가하여 일치하는 항목 수를 반환합니다. 항목 데이터를 따로 반환하지는 않습니다.

generateCreateTableRequest

DynamoDB 테이블을 나타내는 POJO 클래스를 구문 분석한 다음 해당 테이블의 `CreateTableRequest`를 반환합니다.

createS3Link

Amazon S3 객체에 대한 링크를 생성합니다. 이때 버킷 객체의 고유 식별자로서 버킷 이름과 키 이름을 지정해야 합니다.

`createS3Link`를 사용하려면 접근자(getter/setter) 메서드를 정의해야 합니다. 다음 코드 예제에서는 새 속성 및 getter/setter 메서드를 `CatalogItem` 클래스에 추가하여 이를 설명합니다.

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    ...

    public S3Link productImage;

    ....

    @DynamoDBAttribute(attributeName = "ProductImage")
    public S3Link getProductImage() {
            return productImage;
    }

    public void setProductImage(S3Link productImage) {
        this.productImage = productImage;
    }

...
}
```

다음 Java 코드에서는 `Product` 테이블에 쓸 새 항목을 정의합니다. 코드를 보면 항목에 제품 이미지 링크가 추가되어 있습니다. 이 이미지 데이터는 Amazon S3에 업로드됩니다.

```
CatalogItem item = new CatalogItem();
```

```
item.setId(150);
item.setTitle("Book 150 Title");

String amzn-s3-demo-bucket = "amzn-s3-demo-bucket";
String myS3Key = "productImages/book_150_cover.jpg";
item.setProductImage(mapper.createS3Link(amzn-s3-demo-bucket, myS3Key));

item.getProductImage().uploadFrom(new File("/file/path/book_150_cover.jpg"));

mapper.save(item);
```

`S3Link` 클래스는 Amazon S3의 객체를 조작할 수 있는 여러 가지 다른 메서드를 제공합니다. 자세한 내용은 `S3Link` Javadocs를 참조하세요.

getS3ClientCache

Amazon S3에 액세스할 수 있는 기본 `S3ClientCache`를 반환합니다. `S3ClientCache`는 `AmazonS3Client` 객체를 위한 스마트 맵입니다. 클라이언트가 다수일 때는 `S3ClientCache`가 AWS 리전별로 클라이언트를 구성하는 데 효과적일 뿐만 아니라 필요하다면 새 Amazon S3 클라이언트를 생성할 수도 있습니다.

Java용 DynamoDBMapper에서 지원되는 데이터 형식

이 단원에서는 Amazon DynamoDB에서 지원되는 기본 Java 데이터 형식, 컬렉션 및 임의의 데이터 형식을 설명합니다.

Amazon DynamoDB는 다음과 같은 기본 Java 데이터 형식 및 기본 래퍼 클래스를 지원합니다.

- `String`
- `Boolean`, `boolean`
- `Byte`, `byte`
- `Date`(ISO_8601 밀리초 정밀도 문자열에 따라 UTC로 전환)
- `Calendar`(ISO_8601 밀리초 정밀도 문자열에 따라 UTC로 전환)
- `Long`, `long`
- `Integer`, `int`
- `Double`, `double`
- `Float`, `float`

- `BigDecimal`
- `BigInteger`

> **Note**
>
> - DynamoDB 이름 지정 규칙 및 지원되는 다양한 데이터 형식에 대한 자세한 내용은 Amazon DynamoDB에서 지원되는 데이터 형식 및 이름 지정 규칙 단원을 참조하세요.
> - DynamoDBMapper에서 빈 이진수 값이 지원됩니다.
> - AWS SDK for Java 2.x에서 빈 문자열 값이 지원됩니다.
>
>  AWS SDK for Java 1.x에서 DynamoDBMapper는 빈 문자열 속성 값 읽기를 지원하지만, 해당 속성이 요청에서 삭제되므로 빈 문자열 속성 값을 쓰지 않습니다.

DynamoDB는 Java Set, List 및 Map 컬렉션 형식을 지원합니다. 다음 표에는 이러한 Java 형식이 DynamoDB 형식으로 매핑되는 방법이 요약되어 있습니다.

| Java 형식 | DynamoDB 형식 |
|---|---|
| 모두 숫자 형식 | N(숫자 형식) |
| 문자열 | S(문자열 형식) |
| 불 | BOOL(부울 형식), 0 또는 1 |
| ByteBuffer | B(이진수 형식) |
| 날짜 | S(문자열 형식) 날짜 값은 ISO-8601 포맷 문자열로 저장됩니다. |
| Set(집합) 컬렉션 형식 | SS(문자열 집합) 형식, NS(숫자 집합) 형식, 또는 BS(이진수 집합) 형식 |

`DynamoDBTypeConverter` 인터페이스를 사용하면 자체적인 임의 데이터 형식을 DynamoDB에서 기본적으로 지원되는 데이터 형식으로 매핑할 수 있습니다. 자세한 내용은 DynamoDB에서 임의 데이터 매핑 섹션을 참조하세요.

## DynamoDB에 사용되는 Java 주석

이 단원에서는 클래스 및 속성을 Amazon DynamoDB의 테이블 및 속성으로 매핑하는 데 사용되는 주석을 설명합니다.

해당 Javadoc 설명서는 AWS SDK for Java API 참조의 주석 유형 요약 단원을 참조하세요.

> **Note**
>
> 다음 주석에서 `DynamoDBTable` 및 `DynamoDBHashKey`만 필요합니다.

### 주제

- DynamoDBAttribute
- DynamoDBAutoGeneratedKey
- DynamoDBAutoGeneratedTimestamp
- DynamoDBDocument
- DynamoDBHashKey
- DynamoDBIgnore
- DynamoDBIndexHashKey
- DynamoDBIndexRangeKey
- DynamoDBRangeKey
- DynamoDBTable
- DynamoDBTypeConverted
- DynamoDBTyped
- DynamoDBVersionAttribute

### DynamoDBAttribute

속성을 테이블 속성으로 매핑합니다. 기본적으로 각 클래스 속성은 이름이 동일한 항목 속성으로 매핑됩니다. 하지만 이름이 다를 경우에는 이 주석을 사용하여 클래스 속성을 테이블 속성으로 매핑할 수 있습니다. 아래 Java 코드 조각에서는 `DynamoDBAttribute`가 `BookAuthors` 속성을 테이블의 속성 이름으로 매핑하고 있습니다.`Authors`

```
@DynamoDBAttribute(attributeName = "Authors")
public List<String> getBookAuthors() { return BookAuthors; }
```

```
public void setBookAuthors(List<String> BookAuthors) { this.BookAuthors =
 BookAuthors; }
```

DynamoDBMapper가 객체를 테이블에 저장할 때는 Authors를 속성 이름으로 사용합니다.

### DynamoDBAutoGeneratedKey

파티션 키 또는 정렬 키 속성을 자동 생성되는 것으로 표시합니다. DynamoDBMapper는 이러한 속성을 저장할 때 임의의 UUID를 생성합니다. String 속성만 자동 생성된 키로 표시될 수 있습니다.

다음 예제에서는 자동 생성된 키를 사용하는 방법을 보여줍니다.

```
@DynamoDBTable(tableName="AutoGeneratedKeysExample")
public class AutoGeneratedKeys {
    private String id;
    private String payload;

    @DynamoDBHashKey(attributeName = "Id")
    @DynamoDBAutoGeneratedKey
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    @DynamoDBAttribute(attributeName="payload")
    public String getPayload() { return this.payload; }
    public void setPayload(String payload) { this.payload = payload; }

    public static void saveItem() {
        AutoGeneratedKeys obj = new AutoGeneratedKeys();
        obj.setPayload("abc123");

        // id field is null at this point
        DynamoDBMapper mapper = new DynamoDBMapper(dynamoDBClient);
        mapper.save(obj);

        System.out.println("Object was saved with id " + obj.getId());
    }
}
```

### DynamoDBAutoGeneratedTimestamp

자동으로 타임스탬프를 생성합니다.

```
@DynamoDBAutoGeneratedTimestamp(strategy=DynamoDBAutoGenerateStrategy.ALWAYS)
```

```
public Date getLastUpdatedDate() { return lastUpdatedDate; }
public void setLastUpdatedDate(Date lastUpdatedDate) { this.lastUpdatedDate =
 lastUpdatedDate; }
```

원하는 경우 전략 속성을 제공하여 자동 생성 전략을 정의할 수 있습니다. 기본값은 ALWAYS입니다.

DynamoDBDocument

클래스를 Amazon DynamoDB 문서로 직렬화할 수 있음을 나타냅니다.

예를 들어, JSON 문서를 Map 형식(M)의 DynamoDB 속성으로 매핑하려는 경우 다음 코드 예제에서는 Map 유형의 중첩된 속성(Pictures)을 포함하는 항목을 정의합니다.

```
public class ProductCatalogItem {

    private Integer id;  //partition key
    private Pictures pictures;
    /* ...other attributes omitted... */

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id;}
    public void setId(Integer id) {this.id = id;}

    @DynamoDBAttribute(attributeName="Pictures")
    public Pictures getPictures() { return pictures;}
    public void setPictures(Pictures pictures) {this.pictures = pictures;}

    // Additional properties go here.

    @DynamoDBDocument
    public static class Pictures {
        private String frontView;
        private String rearView;
        private String sideView;

        @DynamoDBAttribute(attributeName = "FrontView")
        public String getFrontView() { return frontView; }
        public void setFrontView(String frontView) { this.frontView = frontView; }

        @DynamoDBAttribute(attributeName = "RearView")
        public String getRearView() { return rearView; }
        public void setRearView(String rearView) { this.rearView = rearView; }
```

```
        @DynamoDBAttribute(attributeName = "SideView")
        public String getSideView() { return sideView; }
        public void setSideView(String sideView) { this.sideView = sideView; }

    }
}
```

그러면 다음 예제와 같이 새 ProductCatalog 항목을 Pictures와 함께 저장할 수 있습니다.

```
ProductCatalogItem item = new ProductCatalogItem();

Pictures pix = new Pictures();
pix.setFrontView("http://example.com/products/123_front.jpg");
pix.setRearView("http://example.com/products/123_rear.jpg");
pix.setSideView("http://example.com/products/123_left_side.jpg");
item.setPictures(pix);

item.setId(123);

mapper.save(item);
```

결과로 얻은 ProductCatalog 항목은 다음과 같을 것입니다(JSON 형식).

```
{
  "Id" : 123
  "Pictures" : {
    "SideView" : "http://example.com/products/123_left_side.jpg",
    "RearView" : "http://example.com/products/123_rear.jpg",
    "FrontView" : "http://example.com/products/123_front.jpg"
  }
}
```

### DynamoDBHashKey

클래스 속성을 테이블의 파티션 키 속성으로 매핑합니다. 속성은 스칼라 문자열, 숫자 또는 이진수 형식 중 하나여야 합니다. 속성은 컬렉션 형식일 수 없습니다.

ProductCatalog 테이블이 있고 이 테이블에서 기본 키로 Id를 사용하는 경우 다음 Java 코드에서는 CatalogItem 클래스를 정의하고 @DynamoDBHashKey 태그를 사용하여 해당 Id 속성을 ProductCatalog 테이블의 기본 키로 매핑합니다.

```
@DynamoDBTable(tableName="ProductCatalog")
```

```
public class CatalogItem {
    private Integer Id;
   @DynamoDBHashKey(attributeName="Id")
   public Integer getId() {
        return Id;
   }
   public void setId(Integer Id) {
        this.Id = Id;
   }
   // Additional properties go here.
}
```

DynamoDBIgnore

DynamoDBMapper 인스턴스에게 연동되어 있는 속성을 무시하라고 지시합니다. 그러면 데이터를 테이블에 저장할 때도 DynamoDBMapper는 이 속성을 테이블에 저장하지 않습니다.

모델링되지 않은 속성의 클래스 필드 또는 getter 메서드에 적용됩니다. 클래스 필드에 주석이 직접 적용되는 경우 해당 getter 및 setter를 동일한 클래스에 선언해야 합니다.

DynamoDBIndexHashKey

클래스 속성을 글로벌 보조 인텍스의 파티션 키로 매핑합니다. 속성은 스칼라 문자열, 숫자 또는 이진수 형식 중 하나여야 합니다. 속성은 컬렉션 형식일 수 없습니다.

이 주석은 글로벌 보조 인덱스를 Query해야 하는 경우 사용합니다. 이때는 인덱스 이름(globalSecondaryIndexName)을 지정해야 합니다. 클래스 속성 이름이 인덱스 파티션 키와 다른 경우 인덱스 속성 이름(attributeName)도 지정해야 합니다.

DynamoDBIndexRangeKey

클래스 속성을 글로벌 보조 인덱스 또는 로컬 보조 인덱스의 정렬 키로 매핑합니다. 속성은 스칼라 문자열, 숫자 또는 이진수 형식 중 하나여야 합니다. 속성은 컬렉션 형식일 수 없습니다.

이 주석은 로컬 보조 주석 또는 글로벌 보조 주석을 Query해야 하고 인덱스 정렬 키를 사용해 결과를 구체화하고 싶을 때 사용합니다. 이때는 인덱스 이름(globalSecondaryIndexName 또는 localSecondaryIndexName)을 지정해야 합니다. 그리고 클래스 속성 이름과 인덱스 정렬 키가 다른 경우에는 인덱스 속성 이름(attributeName)도 지정해야 합니다.

DynamoDBRangeKey

클래스 속성을 테이블의 정렬 키 속성으로 매핑합니다. 속성은 스칼라 문자열, 숫자 또는 이진수 형식 중 하나여야 합니다. 속성은 컬렉션 형식일 수 없습니다.

기본 키가 복합형(파티션 키 및 정렬 키)일 경우 이 태그를 사용하여 클래스 필드를 정렬 키로 매핑할 수 있습니다. 예를 들어, 포럼 스레드에 대한 회신을 저장하는 Reply 테이블이 있는 경우 스레드마다 다수의 댓글을 가질 수 있습니다. 따라서 이 테이블의 기본 키는 ThreadId 및 ReplyDateTime입니다. ThreadId는 파티션 키이고, ReplyDateTime은 정렬 키입니다.

다음은 Reply 클래스를 정의하여 Reply 테이블에 매핑하는 Java 코드입니다. 코드 조각을 보면 @DynamoDBHashKey 및 @DynamoDBRangeKey 태그를 사용하여 기본 키에 매핑되는 클래스 속성을 식별하고 있습니다.

```
@DynamoDBTable(tableName="Reply")
public class Reply {
    private Integer id;
    private String replyDateTime;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }

    @DynamoDBRangeKey(attributeName="ReplyDateTime")
    public String getReplyDateTime() { return replyDateTime; }
    public void setReplyDateTime(String replyDateTime) { this.replyDateTime =
 replyDateTime; }

   // Additional properties go here.
}
```

### DynamoDBTable

DynamoDB의 대상 테이블을 식별합니다. 예를 들어, 다음 Java 코드는 Developer 클래스를 정의하여 DynamoDB의 People 테이블에 매핑합니다.

```
@DynamoDBTable(tableName="People")
public class Developer { ...}
```

@DynamoDBTable 주석은 상속이 가능합니다. Developer 클래스에서 상속되는 새 클래스도 People 테이블로 매핑됩니다. 예를 들어 Lead 클래스를 Developer 클래스에서 상속하여 생성한다고 가정하겠습니다. Developer 클래스를 People 테이블로 매핑했기 때문에 Lead 클래스 객체도 동일한 테이블에 저장됩니다.

@DynamoDBTable 또한 재정의가 가능합니다. 기본적으로 Developer 클래스에서 상속되는 새 클래스는 모두 동일한 People 테이블로 매핑됩니다. 하지만 이 기본 매핑을 재정의할 수도 있습니다.

예를 들어, Developer 클래스에서 상속되는 클래스를 생성하는 경우 다음 Java 코드 예제와 같이 @DynamoDBTable 주석을 추가하여 해당 클래스를 다른 테이블로 명시적으로 매핑할 수 있습니다.

```
@DynamoDBTable(tableName="Managers")
public class Manager extends Developer { ...}
```

DynamoDBTypeConverted

속성을 사용자 지정 유형 변환기를 사용하는 것으로 표시하는 주석 DynamoDBTypeConverter에 추가 속성을 전달하기 위한 사용자 정의 주석에 작성할 수 있습니다.

DynamoDBTypeConverter 인터페이스를 사용하면 자체적인 임의 데이터 형식을 DynamoDB에서 기본적으로 지원되는 데이터 형식으로 매핑할 수 있습니다. 자세한 내용은 DynamoDB에서 임의 데이터 매핑 섹션을 참조하세요.

DynamoDBTyped

표준 속성 형식 바인딩을 재정의하는 주석 표준 형식에서 해당 형식에 기본 속성 바인딩을 적용할 경우 주석이 필요하지 않습니다.

DynamoDBVersionAttribute

낙관적 잠금 버전 번호를 저장하는 클래스 속성을 식별합니다. DynamoDBMapper는 새로운 항목을 저장할 때 버전 번호를 이 속성에 할당한 후 항목을 업데이트할 때마다 버전 번호를 올립니다. 여기서는 숫자 스칼라 형식만 지원됩니다. 데이터 형식에 대한 자세한 내용은 데이터 타입 단원을 참조하세요. 버전 관리에 대한 자세한 내용은 DynamoDB 및 버전 번호를 이용한 낙관적 잠금 섹션을 참조하십시오.

DynamoDBMapper의 구성 설정(선택 사항)

DynamoDBMapper의 인스턴스를 생성할 때는 일정한 기본 동작이 있지만 DynamoDBMapperConfig 클래스를 사용하면 이러한 기본 동작을 재정의할 수 있습니다.

다음은 사용자 지정 설정으로 DynamoDBMapper를 생성하는 코드 조각입니다.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapperConfig mapperConfig = DynamoDBMapperConfig.builder()
        .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER)
        .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT)
        .withTableNameOverride(null)

  .withPaginationLoadingStrategy(DynamoDBMapperConfig.PaginationLoadingStrategy.EAGER_LOADING)
     .build();
```

```
DynamoDBMapper mapper = new DynamoDBMapper(client, mapperConfig);
```

자세한 내용은 AWS SDK for Java API 참조에서 DynamoDBMapperConfig를 참조하세요.

DynamoDBMapperConfig 인스턴스에 사용할 수 있는 인수는 다음과 같습니다.

- DynamoDBMapperConfig.ConsistentReads 열거 값:
  - EVENTUAL - 매퍼 인스턴스가 최종적으로 일관된 읽기 요청을 사용합니다.
  - CONSISTENT - 매퍼 인스턴스가 강력히 일관된 읽기 요청을 사용합니다. 이 옵션 설정은 load, query 또는 scan 작업에 사용할 수 있습니다. 강력히 일관된 읽기는 성능과 결제에 영향을 미칩니다. 자세한 내용은 DynamoDB 제품 세부 정보 페이지를 참조하세요.

  매퍼 인스턴스에 읽기 일관성 설정을 지정하지 않으면 EVENTUAL이 기본값으로 설정됩니다.

  > **Note**
  >
  > 이 값은 DynamoDBMapper의 query, querypage, load 및 batch load 작업에만 적용됩니다.

- DynamoDBMapperConfig.PaginationLoadingStrategy 열거 값 - 매퍼 인스턴스가 query 또는scan의 결과와 같은 페이지 매긴 데이터 목록을 처리하는 방식을 제어합니다.
  - LAZY_LOADING - 매퍼 인스턴스는 가능할 경우 데이터를 로드하고, 로드한 결과를 모두 메모리에 저장합니다.
  - EAGER_LOADING - 목록 초기화 직후 매퍼 인스턴스가 데이터를 로드합니다.
  - ITERATION_ONLY - 반복자를 사용해야만 목록에서 읽을 수 있습니다. 반복 단계에서는 다음 페이지를 로드하기 전에 이전 결과가 목록에서 사라집니다. 따라서 로드된 결과는 단일 페이지 목록으로만 메모리에 저장합니다. 이는 목록의 반복 횟수가 1회로 제한된다는 것을 의미하기도 합니다. 이러한 전략은 다수의 항목을 처리하면서 메모리 오버헤드를 줄여야 할 때 바람직합니다.

  매퍼 인스턴스에 페이지 매김 로딩 전략을 지정하지 않으면 LAZY_LOADING이 기본값으로 설정됩니다.

- DynamoDBMapperConfig.SaveBehavior 열거 값 - 저장 작업 시 매퍼 인스턴스의 속성 처리 방식을 지정합니다.
  - UPDATE - 저장 작업 중 모델링된 속성만 모두 업데이트되고 모델링되지 않은 속성은 업데이트되지 않습니다. 기본적인 숫자 형식(byte, int, long)은 0으로 설정됩니다. 그리고 객체 형식은 null로 설정됩니다.

- `CLOBBER` - 저장 작업 중 모델링되지 않은 속성을 비롯하여 모든 속성을 지우고 교체합니다. 즉, 항목을 삭제했다가 다시 생성합니다. 버전이 지정된 필드 제약 조건 역시 무시됩니다.

  매퍼 인스턴스에 저장 동작을 지정하지 않으면 `UPDATE`가 기본값으로 설정됩니다.

  > **Note**
  >
  > DynamoDBMapper 트랜잭션 작업은 `DynamoDBMapperConfig.SaveBehavior` 열거를 지원하지 않습니다.

- `DynamoDBMapperConfig.TableNameOverride` - 객체 매퍼 인스턴스에 클래스의 `DynamoDBTable` 주석으로 지정한 테이블 이름은 무시하고 대신에 직접 입력하는 다른 테이블 이름을 사용하도록 지시합니다. 이는 런타임에서 데이터를 여러 테이블로 분할할 때 유용합니다.

필요하다면 작업 단위로 `DynamoDBMapper`의 기본 구성 객체를 재정의할 수도 있습니다.

### DynamoDB 및 버전 번호를 이용한 낙관적 잠금

낙관적 잠금은 업데이트 또는 삭제하려는 클라이언트 측 항목이 Amazon DynamoDB의 항목과 동일하도록 하려는 전략입니다. 이 전략을 사용하면 다른 사용자의 쓰기 작업이 자신의 데이터베이스 쓰기 작업을 덮어쓰지 못하도록 보호하며, 그 반대도 마찬가지입니다.

낙관적 잠금 전략에서는 각 항목마다 버전 번호 역할을 하는 속성이 있습니다. 항목을 테이블에서 가져오면 애플리케이션이 해당 항목의 버전 번호를 기록합니다. 이 항목을 업데이트할 수는 있지만 서버 쪽 버전 번호가 바뀌지 않는 경우에 한합니다. 버전 불일치가 있는 경우 사용자가 항목을 수정하기 전에 다른 사람이 해당 항목을 수정했음을 의미합니다. 더 이상 유효하지 않은 항목 버전이 있으므로 업데이트 시도가 실패합니다. 이러한 경우 항목을 검색한 후 해당 항목 업데이트를 시도하여 재시도합니다. 낙관적 잠금을 통해 다른 사람이 수행한 변경 사항을 잘못 덮어쓰지 않게 됩니다. 또한 다른 사람이 사용자의 변경 사항을 잘못 덮어쓰지 않도록 합니다.

자체적인 낙관적 잠금 전략을 구현할 수 있지만 AWS SDK for Java에서는 `@DynamoDBVersionAttribute` 주석을 제공합니다. 테이블 매핑 클래스에서 버전 번호를 저장할 속성을 하나 지정하여 이 주석을 사용해 표시하면 됩니다. 그러면 객체를 저장할 때 DynamoDB 테이블의 해당 항목이 버전 번호를 저장하는 속성을 갖게 됩니다. 처음 객체를 저장할 때 `DynamoDBMapper`가 버전 번호를 할당하고, 이후 항목을 업데이트할 때마다 버전 번호가 일정하게 자동으로 오릅니다. 업데이트 또는 삭제 요청은 클라이언트 측 객체 버전이 DynamoDB 테이블의 해당 항목 버전 번호와 일치해야만 가능합니다.

다음의 경우 `ConditionalCheckFailedException`이 발생합니다.

- `@DynamoDBVersionAttribute`를 이용한 낙관적 잠금을 사용하며, 서버의 버전 값이 클라이언트 측의 값과 다를 경우
- `DynamoDBSaveExpression`과 함께 `DynamoDBMapper`를 사용하여 데이터를 저장하는 동안 고유의 조건부 제약 조건을 지정하며 이러한 제약 조건이 실패한 경우

> **Note**
>
> - DynamoDB 전역 테이블은 동시 업데이트 간에 "last writer wins" 조정을 사용합니다. 전역 테이블을 사용할 경우 last writer 정책을 우선 적용합니다. 따라서 잠금 전략이 작동하지 않습니다.
> - `DynamoDBMapper` 트랜잭션 쓰기 작업은 동일한 객체에 대해 `@DynamoDBVersionAttribute` 주석 및 조건 표현식을 지원하지 않습니다. 트랜잭션 쓰기 내 객체에 `@DynamoDBVersionAttribute` 주석이 추가되고 조건식도 있는 경우에는 SdkClientException이 발생합니다.

예를 들어, 다음 Java 코드에서는 몇 가지 속성이 있는 `CatalogItem` 클래스를 정의합니다. `Version` 속성이 `@DynamoDBVersionAttribute` 주석으로 표시되어 있습니다.

Example

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;
    private Long version;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer Id) { this.id = Id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
```

```
    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN;}

    @DynamoDBAttribute(attributeName = "Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
    public String getSomeProp() { return someProp;}
    public void setSomeProp(String someProp) {this.someProp = someProp;}

    @DynamoDBVersionAttribute
    public Long getVersion() { return version; }
    public void setVersion(Long version) { this.version = version;}
}
```

@DynamoDBVersionAttribute 주석을 Long이나 Integer 같이 null이 허용된 형식을 제공하는 기본 래퍼 클래스에서 제공하는 null 형식에 적용할 수 있습니다.

낙관적 잠금 전략은 이러한 DynamoDBMapper 메서드에 아래와 같은 영향을 끼칩니다.

- save - 새 항목의 경우 DynamoDBMapper는 초기 버전 번호로 1을 할당합니다. 항목을 검색하고, 해당 속성 중 하나 이상을 업데이트하고, 변경 사항 저장을 시도하면 클라이언트 측 버전 번호와 서버 측 버전 번호가 일치하는 경우에만 저장 작업이 성공합니다. 그런 다음 DynamoDBMapper가 버전 번호를 자동으로 일정하게 올립니다.
- delete - delete 메서드가 객체를 파라미터로 사용하고 DynamoDBMapper가 항목을 삭제하기 전에 버전을 검사합니다. 요청 시 DynamoDBMapperConfig.SaveBehavior.CLOBBER를 지정하면 버전 검사는 비활성화됩니다.

  DynamoDBMapper에서 낙관적 잠금을 내부 구현할 경우에는 DynamoDB가 제공하는 조건부 업데이트와 조건부 삭제 지원을 사용합니다.
- transactionWrite —
  - Put - 새 항목의 경우 DynamoDBMapper는 초기 버전 번호로 1을 할당합니다. 이후 항목을 가져와 속성을 하나 이상 업데이트한 후 변경 사항을 저장하려고 해도 클라이언트 측과 서버 쪽의 버전 번호가 일치해야만 넣기 작업이 가능합니다. 그런 다음 DynamoDBMapper가 버전 번호를 자동으로 일정하게 올립니다.
  - Update - 새 항목의 경우 DynamoDBMapper는 초기 버전 번호로 1을 할당합니다. 이후 항목을 가져와 속성을 하나 이상 업데이트한 후 변경 사항을 저장하려고 해도 클라이언트 측과 서버 쪽의 버

전 번호가 일치해야만 업데이트 작업이 가능합니다. 그런 다음 DynamoDBMapper가 버전 번호를 자동으로 일정하게 올립니다.

- Delete - DynamoDBMapper는 항목을 삭제하기 전에 버전을 확인합니다. 클라이언트 측과 서버 측 버전 번호가 일치할 경우에만 삭제 작업이 가능합니다.
- ConditionCheck - @DynamoDBVersionAttribute 주석은 ConditionCheck 작업에 지원되지 않습니다. ConditionCheck 항목에 @DynamoDBVersionAttribute 주석이 추가될 때 SdkClientException이 발생합니다.

### 낙관적 잠금 비활성화

낙관적 잠금은 DynamoDBMapperConfig.SaveBehavior 열거 값을 UPDATE에서 CLOBBER로 변경하면 비활성화할 수 있습니다. 그런 다음 버전 검사를 제외한 DynamoDBMapperConfig 인스턴스를 생성하여 모든 요청에 사용하면 됩니다. DynamoDBMapperConfig.SaveBehavior와 그 밖에 옵션으로 제공되는 DynamoDBMapper 파라미터에 대한 자세한 내용은 DynamoDBMapper의 구성 설정(선택 사항) 단원을 참조하세요.

또한 잠금 기능을 특정 작업에만 설정할 수도 있습니다. 예를 들어 다음은 DynamoDBMapper를 사용하여 카탈로그 항목을 저장하는 Java 코드 조각입니다. 이 조각을 보면 옵션인 DynamoDBMapperConfig 파라미터를 save 메서드에 추가하여 DynamoDBMapperConfig.SaveBehavior를 지정하고 있습니다.

> **Note**
>
> transactionWrite 메서드는 DynamoDBMapperConfig.SaveBehavior 구성을 지원하지 않습니다. transactionWrite의 낙관적 잠금 비활성화는 지원되지 않습니다.

### Example

```
DynamoDBMapper mapper = new DynamoDBMapper(client);

// Load a catalog item.
CatalogItem item = mapper.load(CatalogItem.class, 101);
item.setTitle("This is a new title for the item");
...
// Save the item.
mapper.save(item,
    new DynamoDBMapperConfig(
```

```
        DynamoDBMapperConfig.SaveBehavior.CLOBBER));
```

### DynamoDB에서 임의 데이터 매핑

지원되는 Java 형식(Java용 DynamoDBMapper에서 지원되는 데이터 형식 참조) 외에도 애플리케이션에서 직접적으로 Amazon DynamoDB 형식에 매핑되지 않는 형식을 사용할 수 있습니다. 이러한 형식을 매핑하려면 복합 형식을 DynamoDB 지원 형식으로 또는 그 반대로의 변환을 구현하고 `@DynamoDBTypeConverted` 주석을 사용하여 복합 형식의 접근자 메서드에 주석을 추가해야 합니다. 이후 객체를 저장하거나 로드하면 변환기 코드가 데이터를 변환합니다. 그 밖에 복합 형식을 사용하는 모든 작업에서도 유용합니다. 단, 쿼리나 스캔 작업 중 데이터를 비교하면 DynamoDB에 저장된 데이터를 비교하게 됩니다.

아래에서 `Dimension` 속성, 즉 `DimensionType`을 정의하고 있는 `CatalogItem` 클래스를 예로 들어 보겠습니다. 이 속성에는 높이, 너비, 두께 등의 항목 차원이 저장됩니다. DynamoDB에서 이 항목 크기를 문자열(8.5x11x.05)로 저장한다고 가정할 때, 다음 예제를 보면 `DimensionType` 객체를 문자열로, 혹은 문자열을 `DimensionType`으로 변환하는 변환기 코드가 나와있습니다.

> **Note**
>
> 아래 코드 예제는 DynamoDB에서 테이블 생성 및 코드 예시에 대한 데이터 로드 단원의 지침에 따라 이미 계정의 DynamoDB에 데이터를 로드하였다고 가정한 것입니다.
> 다음 예제를 실행하기 위한 단계별 지침은 Java 코드 예 섹션을 참조하세요.

### Example

```
public class DynamoDBMapperExample {

    static AmazonDynamoDB client;

    public static void main(String[] args) throws IOException {

        // Set the AWS region you want to access.
        Regions usWest2 = Regions.US_WEST_2;
        client = AmazonDynamoDBClientBuilder.standard().withRegion(usWest2).build();

        DimensionType dimType = new DimensionType();
        dimType.setHeight("8.00");
        dimType.setLength("11.0");
        dimType.setThickness("1.0");
```

```
        Book book = new Book();
        book.setId(502);
        book.setTitle("Book 502");
        book.setISBN("555-5555555555");
        book.setBookAuthors(new HashSet<String>(Arrays.asList("Author1", "Author2")));
        book.setDimensions(dimType);

        DynamoDBMapper mapper = new DynamoDBMapper(client);
        mapper.save(book);

        Book bookRetrieved = mapper.load(Book.class, 502);
        System.out.println("Book info: " + "\n" + bookRetrieved);

        bookRetrieved.getDimensions().setHeight("9.0");
        bookRetrieved.getDimensions().setLength("12.0");
        bookRetrieved.getDimensions().setThickness("2.0");

        mapper.save(bookRetrieved);

        bookRetrieved = mapper.load(Book.class, 502);
        System.out.println("Updated book info: " + "\n" + bookRetrieved);
    }

    @DynamoDBTable(tableName = "ProductCatalog")
    public static class Book {
        private int id;
        private String title;
        private String ISBN;
        private Set<String> bookAuthors;
        private DimensionType dimensionType;

        // Partition key
        @DynamoDBHashKey(attributeName = "Id")
        public int getId() {
            return id;
        }

        public void setId(int id) {
            this.id = id;
        }

        @DynamoDBAttribute(attributeName = "Title")
        public String getTitle() {
```

```
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        @DynamoDBAttribute(attributeName = "ISBN")
        public String getISBN() {
            return ISBN;
        }

        public void setISBN(String ISBN) {
            this.ISBN = ISBN;
        }

        @DynamoDBAttribute(attributeName = "Authors")
        public Set<String> getBookAuthors() {
            return bookAuthors;
        }

        public void setBookAuthors(Set<String> bookAuthors) {
            this.bookAuthors = bookAuthors;
        }

        @DynamoDBTypeConverted(converter = DimensionTypeConverter.class)
        @DynamoDBAttribute(attributeName = "Dimensions")
        public DimensionType getDimensions() {
            return dimensionType;
        }

        @DynamoDBAttribute(attributeName = "Dimensions")
        public void setDimensions(DimensionType dimensionType) {
            this.dimensionType = dimensionType;
        }

        @Override
        public String toString() {
            return "Book [ISBN=" + ISBN + ", bookAuthors=" + bookAuthors + ",
dimensionType= "
                    + dimensionType.getHeight() + " X " + dimensionType.getLength() + "
X "
                    + dimensionType.getThickness()
                    + ", Id=" + id + ", Title=" + title + "]";
```

```
        }
    }

    static public class DimensionType {

        private String length;
        private String height;
        private String thickness;

        public String getLength() {
            return length;
        }

        public void setLength(String length) {
            this.length = length;
        }

        public String getHeight() {
            return height;
        }

        public void setHeight(String height) {
            this.height = height;
        }

        public String getThickness() {
            return thickness;
        }

        public void setThickness(String thickness) {
            this.thickness = thickness;
        }
    }

    // Converts the complex type DimensionType to a string and vice-versa.
    static public class DimensionTypeConverter implements DynamoDBTypeConverter<String,
DimensionType> {

        @Override
        public String convert(DimensionType object) {
            DimensionType itemDimensions = (DimensionType) object;
            String dimension = null;
            try {
                if (itemDimensions != null) {
```

```
                    dimension = String.format("%s x %s x %s",
 itemDimensions.getLength(), itemDimensions.getHeight(),
                            itemDimensions.getThickness());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return dimension;
        }

        @Override
        public DimensionType unconvert(String s) {

            DimensionType itemDimension = new DimensionType();
            try {
                if (s != null && s.length() != 0) {
                    String[] data = s.split("x");
                    itemDimension.setLength(data[0].trim());
                    itemDimension.setHeight(data[1].trim());
                    itemDimension.setThickness(data[2].trim());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }

            return itemDimension;
        }
    }
}
```

### DynamoDBMapper 예제

AWS SDK for Java에서는 클라이언트 측 클래스를 DynamoDB 테이블로 매핑할 수 있는 `DynamoDBMapper` 클래스를 제공합니다. `DynamoDBMapper`를 사용하려면 코드에서 DynamoDB 테이블의 항목과 해당하는 객체 인스턴스 간의 관계를 정의합니다. `DynamoDBMapper` 클래스를 사용하면 항목에 대한 다양한 생성, 읽기, 업데이트 및 삭제(CRUD) 작업을 수행하고 테이블에 대한 쿼리 및 스캔을 실행할 수 있습니다.

`DynamoDBMapper` 사용 방법에 대한 자세한 내용은 AWS SDK for Java 1.x 개발자 안내서에 나와 있는 AWS SDK for Java 사용에 관한 DynamoDB 예제를 참조하시기 바랍니다.

## Java 2.x: DynamoDB 고급형 클라이언트

DynamoDB 고급형 클라이언트는 AWS SDK for Java 버전 2(v2)의 일부인 상위 수준 라이브러리로서, 클라이언트 측 클래스를 DynamoDB 테이블에 매핑하는 간단한 방법을 제공합니다. 코드에서 테이블과 해당 모델 클래스 간의 관계를 정의합니다. 이러한 관계를 정의한 다음 DynamoDB의 테이블 또는 항목에 대해 다양한 생성, 읽기, 업데이트 또는 삭제(CRUD) 작업을 직관적으로 수행할 수 있습니다.

고급형 클라이언트를 DynamoDB와 함께 사용하는 방법에 대한 자세한 내용은 AWS SDK for Java 2.x에서 DynamoDB 고급형 클라이언트 사용을 참조하세요.

## DynamoDB에서 .NET 문서 모델을 사용하여 작업

AWS SDK for .NET은 일부 하위 수준 Amazon DynamoDB 작업을 래핑하여 코딩을 더 간단하게 만드는 문서 모델 클래스를 제공합니다. 이 문서 모델에서 기본 클래스는 `Table` 및 `Document`입니다. `Table` 클래스는 `PutItem`, `GetItem` 및 `DeleteItem`과 같은 데이터 작업 메서드를 제공합니다. 또한 `Query` 및 `Scan` 메서드를 제공합니다. `Document` 클래스는 테이블의 단일 항목을 나타냅니다.

위의 문서 모델 클래스는 `Amazon.DynamoDBv2.DocumentModel` 네임스페이스에서 사용할 수 있습니다.

> **Note**
>
> 이 문서 모델 클래스를 사용하여 테이블을 생성, 업데이트 및 삭제할 수 없습니다. 그러나 이 문서 모델은 대부분의 일반적인 데이터 작업을 지원합니다.

주제

- 지원되는 데이터 유형

지원되는 데이터 유형

문서 모델은 기본 .NET 데이터 형식 및 컬렉션 데이터 형식 세트를 지원합니다. 모델이 지원하는 기본 데이터 유형은 다음과 같습니다.

- `bool`
- `byte`
- `char`

- `DateTime`
- `decimal`
- `double`
- `float`
- `Guid`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

다음 표에서는 앞서 나온 .NET 형식을 DynamoDB 형식으로 매핑하는 것을 요약하여 보여 줍니다.

| .NET 기본 유형 | DynamoDB 형식 |
| --- | --- |
| 모두 숫자 형식 | `N`(숫자 형식) |
| 모든 문자열 형식 | `S`(문자열 형식) |
| MemoryStream, byte[] | `B`(이진수 형식) |
| bool | `N`(숫자 형식). 0은 false를 나타내고 1은 true를 나타냅니다. |
| DateTime | `S`(문자열 형식) `DateTime` 값은 ISO-8601 형식의 문자열로 저장됩니다. |
| Guid | `S`(문자열 형식) |
| 컬렉션 유형(List, HashSet 및 array) | `BS`(이진수 세트) 형식, `SS`(문자열 세트) 형식, 그리고 `NS`(숫자 세트) 형식 |

AWS SDK for .NET은 DynamoDB의 부울, null, 목록 및 맵 형식을 .NET 문서 모델 API에 매핑하기 위한 형식을 정의합니다.

- 부울 형식에 `DynamoDBBool`을 사용합니다.
- null 형식에 `DynamoDBNull`을 사용합니다.
- 목록 형식에 `DynamoDBList`를 사용합니다.
- 맵 형식에 `Document`를 사용합니다.

> **Note**
>
> - 빈 이진수 값이 지원됩니다.
> - 빈 문자열 값 읽기가 지원됩니다. DynamoDB에 쓰는 동안 문자열 집합 형식의 속성 값에서 빈 문자열 속성 값이 지원됩니다. 문자열 형식의 빈 문자열 속성 값과 목록 또는 맵 형식에 포함된 빈 문자열 값은 쓰기 요청에서 삭제됩니다.

## .NET 객체 지속성 모델 및 DynamoDB로 작업

AWS SDK for .NET은 객체 지속성 모델을 제공하므로 이것으로 클라이언트 측 클래스를 Amazon DynamoDB 테이블에 매핑할 수 있습니다. 그러면 각 객체 인스턴스도 해당 테이블 항목으로 매핑됩니다. 클라이언트 측 객체를 테이블에 저장하기 위해 객체 지속성 모델에서는 DynamoDB에 대한 진입점인 `DynamoDBContext` 클래스를 제공합니다. 이 클래스는 DynamoDB로 연결하는 역할을 하기 때문에 테이블에 액세스하여 다양한 CRUD 작업이 가능할 뿐만 아니라 쿼리를 실행할 수 있습니다.

객체 지속성 모델은 속성 세트를 제공하여 클라이언트 측 클래스를 테이블로 매핑할 수 있으며, 속성/필드를 테이블 속성으로 매핑할 수 있습니다.

> **Note**
>
> 객체 지속성 모델은 테이블을 생성, 업데이트 또는 삭제할 수 있는 API를 제공하지 않으며 데이터 작업만 제공합니다. 테이블 생성, 업데이트 및 삭제에는 AWS SDK for .NET 하위 수준 API만 사용할 수 있습니다.

다음 예제에서는 객체 지속성 모델이 작동하는 방법을 보여줍니다. `ProductCatalog` 테이블로 시작합니다. `Id`를 기본 키로 갖고 있습니다.

```
ProductCatalog(Id, ...)
```

Book 클래스를 Title, ISBN 및 Authors 속성과 함께 갖고 있는 경우 다음 C# 코드 예제와 같이 객체 지속성 모델로 정의된 속성을 추가하여 Book 클래스를 ProductCatalog 테이블에 매핑할 수 있습니다.

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]
    public int Id { get; set; }

    public string Title { get; set; }
    public int ISBN { get; set; }

    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }

    [DynamoDBIgnore]
    public string CoverPage { get; set; }
  }
```

앞의 예제에서는 DynamoDBTable 속성이 Book 클래스를 ProductCatalog 테이블에 매핑합니다.

객체 지속성 모델은 클래스 속성 및 테이블 속성 간 명시적 매핑과 기본 매핑 모두를 지원합니다.

- 명시적 매핑 - 속성을 기본 키에 매핑하려면 DynamoDBHashKey 및 DynamoDBRangeKey 객체 지속성 모델 속성을 사용해야 합니다. 또한 기본이 아닌 키 속성의 경우 클래스의 속성 이름과 이를 매핑하려는 해당 테이블 속성이 같지 않다면 DynamoDBProperty 속성을 명시적으로 추가하여 매핑을 정의해야 합니다.

  앞의 예제에서는 Id 속성이 같은 이름의 기본 키로 매핑되고 BookAuthors 속성이 ProductCatalog 테이블의 Authors 속성으로 매핑됩니다.
- 기본 매핑 - 기본적으로 객체 지속성 모델은 클래스 속성을 같은 이름의 테이블 속성으로 매핑합니다.

  앞의 예제에서는 Title 및 ISBN 속성이 ProductCatalog 테이블에 있는 같은 이름의 속성으로 매핑됩니다.

각 클래스 속성을 전부 하나씩 매핑할 필요는 없습니다. DynamoDBIgnore 속성을 추가하여 이러한 속성을 확인할 수 있습니다. Book 인스턴스를 테이블에 저장하더라도 DynamoDBContext에는 CoverPage 속성이 포함되지 않습니다. 또한 이러한 책 인스턴스를 가져와도 이 속성은 반환되지 않습니다.

int 및 문자열과 같은 .NET 기본 유형의 속성을 매핑할 수 있습니다. 적절한 변환기를 사용하여 임의 데이터를 DynamoDB 형식 중 하나로 매핑만 한다면 어떤 임의 데이터 형식도 매핑할 수 있습니다. 임의 유형 매핑에 대한 자세한 내용은 AWS SDK for .NET 객체 지속성 모델을 사용하여 DynamoDB에서 임의 데이터 매핑 단원을 참조하세요.

객체 지속성 모델은 낙관적 잠금을 지원합니다. 따라서 업데이트 작업 동안 업데이트하려는 항목의 최신 복사본을 가질 수 있습니다. 자세한 내용은 DynamoDB 및 AWS SDK for .NET 객체 지속성 모델을 사용하여 낙관적 잠금 수행 단원을 참조하십시오.

자세한 내용은 아래 주제를 참조하십시오.

주제

- 지원되는 데이터 유형
- .NET 객체 지속성 모델의 DynamoDB 속성
- .NET 객체 지속성 모델의 DynamoDBContext 클래스
- DynamoDB 및 AWS SDK for .NET 객체 지속성 모델을 사용하여 낙관적 잠금 수행
- AWS SDK for .NET 객체 지속성 모델을 사용하여 DynamoDB에서 임의 데이터 매핑

### 지원되는 데이터 유형

객체 지속성 모델은 기본 .NET 데이터 유형 세트와 컬렉션, 그리고 임의 데이터 유형을 지원합니다. 모델이 지원하는 기본 데이터 유형은 다음과 같습니다.

- bool
- byte
- char
- DateTime
- decimal
- double
- float
- Int16

- Int32
- Int64
- SByte
- string
- UInt16
- UInt32
- UInt64

객체 지속성 모델은 .NET 컬렉션 형식도 지원합니다. DynamoDBContext는 구체적인 컬렉션 형식과 단순한 POCO(Plain Old CLR Object)를 변환할 수 있습니다.

다음 표에서는 앞서 나온 .NET 형식을 DynamoDB 형식으로 매핑하는 것을 요약하여 보여 줍니다.

| .NET 기본 유형 | DynamoDB 형식 |
| --- | --- |
| 모두 숫자 형식 | N(숫자 형식) |
| 모든 문자열 형식 | S(문자열 형식) |
| MemoryStream, byte[] | B(이진수 형식) |
| bool | N(숫자 형식). 0은 false를 나타내고 1은 true를 나타냅니다. |
| 컬렉션 유형 | BS(이진수 세트) 형식, SS(문자열 세트) 형식, 그리고 NS(숫자 세트) 형식 |
| DateTime | S(문자열 형식) DateTime 값은 ISO-8601 형식의 문자열로 저장됩니다. |

객체 지속성 모델은 임의 데이터 형식 또한 지원합니다. 하지만 여러 형식을 DynamoDB 형식으로 매핑하려면 변환기 코드를 입력해야 합니다.

> **Note**
>
> - 빈 이진수 값이 지원됩니다.

- 빈 문자열 값 읽기가 지원됩니다. DynamoDB에 쓰는 동안 문자열 집합 형식의 속성 값에서 빈 문자열 속성 값이 지원됩니다. 문자열 형식의 빈 문자열 속성 값과 목록 또는 맵 형식에 포함된 빈 문자열 값은 쓰기 요청에서 삭제됩니다.

### .NET 객체 지속성 모델의 DynamoDB 속성

이 단원에서는 사용자가 클래스 및 속성을 DynamoDB 테이블 및 속성으로 매핑할 수 있도록 객체 지속성 모델이 제공하는 속성에 대해 설명합니다.

> **Note**
>
> 다음 속성에서 `DynamoDBTable` 및 `DynamoDBHashKey`만 필요합니다.

#### DynamoDBGlobalSecondaryIndexHashKey

클래스 속성을 글로벌 보조 인덱스의 파티션 키로 매핑합니다. 이 속성은 글로벌 보조 인덱스를 `Query`해야 하는 경우 사용합니다.

#### DynamoDBGlobalSecondaryIndexRangeKey

클래스 속성을 글로벌 보조 인덱스의 정렬 키로 매핑합니다. 이 속성은 글로벌 보조 인덱스를 `Query`해야 하거나 인덱스 정렬 키를 사용하여 결과를 구체화하고 싶은 경우에 사용합니다.

#### DynamoDBHashKey

클래스 속성을 테이블의 기본 키 파티션 키로 매핑합니다. 기본 키 속성은 컬렉션 형식이 될 수 없습니다.

다음 C# 코드 예제에서는 `Book` 클래스를 `ProductCatalog` 테이블로 매핑하고, `Id` 속성을 테이블의 기본 키 파티션 키로 매핑합니다.

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }

    // Additional properties go here.
```

```
}
```

### DynamoDBIgnore

연결된 속성을 무시해야 함을 나타냅니다. 어떠한 클래스 속성도 저장하지 않으려면 이 속성을 추가하여 DynamoDBContext가 객체를 테이블에 저장할 때 이 속성을 포함하지 않도록 지시할 수 있습니다.

### DynamoDBLocalSecondaryIndexRangeKey

클래스 속성을 로컬 보조 인텍스의 정렬 키로 매핑합니다. 이 속성은 로컬 보조 인덱스를 Query해야 하거나 인덱스 정렬 키를 사용하여 결과를 구체화하고 싶은 경우에 사용합니다.

### DynamoDBProperty

클래스 속성을 테이블 속성으로 매핑합니다. 클래스 속성을 같은 이름의 테이블 속성으로 매핑할 경우에는 이 속성을 지정할 필요가 없습니다. 하지만 이름이 다를 경우에는 매핑을 제공하는 데 이 태그를 사용할 수 있습니다. 다음 C# 문에서는 DynamoDBProperty가 BookAuthors 속성을 테이블의 Authors 속성으로 매핑합니다.

```
[DynamoDBProperty("Authors")]
public List<string> BookAuthors { get; set; }
```

DynamoDBContext는 객체 데이터를 해당 테이블에 저장할 때 이 매핑 정보를 사용하여 Authors 속성을 생성합니다.

### DynamoDBRenamable

클래스 속성의 대체 이름을 지정합니다. 이것은 클래스 속성 이름이 테이블 속성과 다른 DynamoDB 테이블로 임의 데이터를 매핑하기 위한 사용자 지정 변환기를 쓰고 있을 때 유용합니다.

### DynamoDBRangeKey

클래스 속성을 테이블의 기본 키 정렬 키로 매핑합니다. 테이블에 복합 기본 키(파티션 키 및 정렬 키)가 있는 경우 클래스 매핑에서 DynamoDBHashKey 속성과 DynamoDBRangeKey 속성을 모두 지정해야 합니다.

예를 들어, 샘플 테이블 Reply에는 Id 파티션 키 및 Replenishment 정렬 키로 구성된 기본 키가 있습니다. 다음 C# 코드 예제에서는 Reply 클래스를 Reply 테이블로 매핑합니다. 클래스 정의는 또한 그 속성 중 두 가지를 기본 키로 매핑하도록 지시합니다.

```
[DynamoDBTable("Reply")]
```

```
public class Reply
{
   [DynamoDBHashKey]
   public int ThreadId { get; set; }
   [DynamoDBRangeKey]
   public string Replenishment { get; set; }

   // Additional properties go here.
}
```

### DynamoDBTable

클래스가 매핑하는 대상으로 DynamoDB의 대상 테이블을 식별합니다. 예를 들어, 다음 C# 코드 예제에서는 `Developer` 클래스를 DynamoDB의 `People` 테이블로 매핑합니다.

```
[DynamoDBTable("People")]
public class Developer { ...}
```

이 속성은 상속 또는 재정의될 수 있습니다.

- `DynamoDBTable` 속성은 상속될 수 있습니다. 앞의 예제에서는 `Developer` 클래스에서 상속된 새 클래스 `Lead`를 추가하는 경우 해당 클래스도 `People` 테이블에도 매핑됩니다. `Developer` 객체와 `Lead` 객체 모두 `People` 테이블에 저장됩니다.
- `DynamoDBTable` 속성 또한 재정의될 수 있습니다. 다음 C# 코드 예제에서는 `Manager` 클래스가 `Developer` 클래스에서 상속됩니다. 그러나 `DynamoDBTable` 속성을 명시적으로 추가하면 이 클래스가 다른 테이블(`Managers`)로 매핑됩니다.

  ```
  [DynamoDBTable("Managers")]
  public class Manager : Developer { ...}
  ```

다음 C# 코드 예제와 같이 테이블에 객체를 저장할 때 선택적 파라미터인 `LowerCamelCaseProperties`를 추가하면 속성 이름의 첫 번째 글자를 소문자로 만들도록 DynamoDB에 요청할 수 있습니다.

```
[DynamoDBTable("People", LowerCamelCaseProperties=true)]
public class Developer
{
    string DeveloperName;
    ...
```

```
}
```

`Developer` 클래스의 인스턴스를 저장할 때 `DynamoDBContext`가 `DeveloperName` 속성을 `developerName`으로 저장합니다.

DynamoDBVersion

항목 버전 번호를 저장하는 클래스 속성을 식별합니다. 버전 관리에 대한 자세한 내용은 DynamoDB 및 AWS SDK for .NET 객체 지속성 모델을 사용하여 낙관적 잠금 수행 섹션을 참조하십시오.

.NET 객체 지속성 모델의 DynamoDBContext 클래스

`DynamoDBContext` 클래스는 Amazon DynamoDB 데이터베이스의 진입점입니다. 이 클래스는 DynamoDB로의 연결을 제공하며, 데이터를 다양한 테이블에 액세스하고 다양한 CRUD 작업을 수행하며 쿼리를 실행할 수 있게 합니다. `DynamoDBContext` 클래스는 다음과 같은 메서드를 제공합니다.

주제

- CreateMultiTableBatchGet
- CreateMultiTableBatchWrite
- CreateBatchGet
- CreateBatchWrite
- 삭제
- Dispose
- ExecuteBatchGet
- ExecuteBatchWrite
- FromDocument
- FromQuery
- FromScan
- GetTargetTable
- Load
- Query
- Save
- 스캔
- ToDocument

- DynamoDBContext에 대한 옵션 파라미터 지정

CreateMultiTableBatchGet

여러 개의 개별 `BatchGet` 객체로 구성된 `MultiTableBatchGet` 객체를 만듭니다. 이러한 `BatchGet` 객체 각각은 단일 DynamoDB 테이블에서 항목을 가져오는 데 사용됩니다.

테이블에서 항목을 검색하려면 `ExecuteBatchGet` 메서드를 사용하여 `MultiTableBatchGet` 객체를 파라미터로 전달합니다.

CreateMultiTableBatchWrite

여러 개의 개별 `BatchWrite` 객체로 구성된 `MultiTableBatchWrite` 객체를 만듭니다. 이러한 `BatchWrite` 객체 각각은 단일 DynamoDB 테이블에서 항목을 쓰거나 삭제하는 데 사용됩니다.

테이블에 쓰려면 `ExecuteBatchWrite` 메서드를 사용하여 `MultiTableBatchWrite` 객체를 파라미터로 전달합니다.

CreateBatchGet

`BatchGet` 객체를 만들어 테이블에서 여러 개의 항목을 가져올 수 있습니다.

CreateBatchWrite

`BatchWrite` 객체를 만들어 여러 개의 항목을 테이블에 넣는 데 사용하거나 테이블에서 여러 개의 항목을 삭제하는 데 사용할 수 있습니다.

삭제

항목을 테이블에서 삭제합니다. 메서드에는 삭제하려는 항목의 기본 키가 필요합니다. 기본 키 값 또는 기본 키 값을 포함하는 클라이언트 측 객체 중 한 가지를 파라미터로 이 메서드에 제공할 수 있습니다.

- 클라이언트 측 객체를 파라미터로 지정하고 낙관적 잠금을 활성화했다면, 클라이언트 측과 객체의 서버 쪽 버전이 일치해야만 삭제가 성공적으로 이루어집니다.
- 기본 키 값만을 파라미터로 지정한 경우, 낙관적 잠금을 활성화했는지의 여부와 상관없이 삭제가 성공적으로 이루어집니다.

> **Note**
>
> 백그라운드에서 이 작업을 수행하려면 `DeleteAsync` 메서드를 대신 사용하세요.

### Dispose

관리되거나 관리되지 않는 리소스를 모두 일괄합니다.

### ExecuteBatchGet

모든 `BatchGet` 객체를 `MultiTableBatchGet`에서 처리하여, 하나 또는 그 이상의 테이블에서 데이터를 읽습니다.

> **Note**
>
> 백그라운드에서 이 작업을 수행하려면 `ExecuteBatchGetAsync` 메서드를 대신 사용하세요.

### ExecuteBatchWrite

모든 `BatchWrite` 객체를 `MultiTableBatchWrite`에서 처리하여, 하나 또는 그 이상의 테이블에 데이터를 쓰거나 삭제합니다.

> **Note**
>
> 백그라운드에서 이 작업을 수행하려면 `ExecuteBatchWriteAsync` 메서드를 대신 사용하세요.

### FromDocument

할당된 `Document` 인스턴스에서 `FromDocument` 메서드가 클라이언트 측 클래스의 인스턴스를 반환합니다.

이것은 문서 모델 클래스를 객체 지속성 모델과 함께 사용하여 데이터 작업을 수행할 때 유용합니다. AWS SDK for .NET에 의해 제공되는 문서 모델 클래스에 대한 자세한 내용은 DynamoDB에서 .NET 문서 모델을 사용하여 작업 단원을 참조하세요.

`Forum` 항목 표시 정보를 포함하는 `doc`라는 이름의 `Document` 객체가 있는 경우 (이 객체를 구성하는 방법은 이 주제의 후반부에 있는 `ToDocument` 메서드 설명을 참조하세요.) 다음 C# 코드 예제와 같이 `FromDocument`를 사용하여 `Document`에서 `Forum` 항목을 검색할 수 있습니다.

### Example

```
forum101 = context.FromDocument<Forum>(101);
```

> **Note**
>
> Document 객체가 IEnumerable 인터페이스를 구현하는 경우에는 FromDocuments 메서드를 대신 사용할 수 있습니다. 그러면 Document에서 모든 클래스 인스턴스를 반복적으로 처리할 수 있습니다.

FromQuery

QueryOperationConfig 객체에 정의된 쿼리 파라미터로 Query 작업을 실행합니다.

> **Note**
>
> 백그라운드에서 이 작업을 수행하려면 FromQueryAsync 메서드를 대신 사용하세요.

FromScan

ScanOperationConfig 객체에 정의된 스캔 파라미터로 Scan 작업을 실행합니다.

> **Note**
>
> 백그라운드에서 이 작업을 수행하려면 FromScanAsync 메서드를 대신 사용하세요.

GetTargetTable

지정된 유형에 대한 대상 테이블을 가져옵니다. 이는 임의 데이터를 DynamoDB 테이블로 매핑하기 위한 사용자 지정 변환기를 작성하고 사용자 지정 데이터 형식과 연결되는 테이블을 확인해야 할 때 유용합니다.

Load

테이블에서 항목을 가져옵니다. 이 메서드에는 가져오려는 항목의 기본 키만 필요합니다.

기본적으로 DynamoDB는 최종적으로 일관된 값을 갖는 항목을 반환하기 때문입니다. 최종 일관성 모델에 대한 자세한 내용은 DynamoDB 읽기 일관성 단원을 참조하세요.

Load 또는 LoadAsync 메서드는 GetItem 작업을 직접 호출합니다. 이 작업을 수행하려면 테이블의 프라이머리 키를 지정해야 합니다. GetItem에서는 IndexName 파라미터를 무시하므로, 인덱스의 파티

션 또는 정렬 키를 사용하여 항목을 로드할 수 없습니다. 따라서 테이블의 프라이머리 키를 사용하여 항목을 로드해야 합니다.

> **Note**
>
> 백그라운드에서 이 작업을 수행하려면 LoadAsync 메서드를 대신 사용하세요. LoadAsync 메서드를 사용하여 DynamoDB 테이블에서 상위 수준의 CRUD 작업을 수행하는 예제를 보려면 다음 예제를 참조하세요.

```
/// <summary>
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
/// table.
/// </summary>
public class HighLevelItemCrud
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);
        await PerformCRUDOperations(context);
    }

    public static async Task PerformCRUDOperations(IDynamoDBContext context)
    {
        int bookId = 1001; // Some unique value.
        Book myBook = new Book
        {
            Id = bookId,
            Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
            Isbn = "111-1111111001",
            BookAuthors = new List<string> { "Author 1", "Author 2" },
        };

        // Save the book to the ProductCatalog table.
        await context.SaveAsync(myBook);

        // Retrieve the book from the ProductCatalog table.
        Book bookRetrieved = await context.LoadAsync<Book>(bookId);

        // Update some properties.
```

```
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
            }
        }
    }
```

### Query

입력하는 쿼리 파라미터를 기반으로 테이블을 쿼리합니다.

테이블에 복합 기본 키(파티션 키 및 정렬 키)가 있는 경우에만 테이블을 쿼리할 수 있습니다. 쿼리 시에는 파티션 키를 비롯하여 정렬 키에 적용되는 조건을 지정해야 합니다.

DynamoDB의 `Reply` 테이블에 클라이언트 측 `Reply` 클래스가 매핑되어 있는 경우 다음 C# 코드 예제에서는 과거 15일 간 게시된 포럼 스레드 회신을 찾기 위해 `Reply` 테이블을 쿼리합니다. `Reply` 테이블에는 `Id` 파티션 키와 `ReplyDateTime` 정렬 키로 구성된 기본 키가 있습니다.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

string replyId = "DynamoDB#DynamoDB Thread 1"; //Partition key
DateTime twoWeeksAgoDate = DateTime.UtcNow.Subtract(new TimeSpan(14, 0, 0, 0)); // Date
 to compare.
IEnumerable<Reply> latestReplies = context.Query<Reply>(replyId,
 QueryOperator.GreaterThan, twoWeeksAgoDate);
```

이를 통해 Reply 객체 컬렉션이 반환됩니다.

Query 메서드는 "지연 로딩된(lazy-loaded)" IEnumerable 컬렉션을 반환합니다. 즉, 처음에는 결과 페이지를 하나만 반환하고, 필요에 따라 서비스를 호출하여 다음 페이지를 반환합니다. 일치하는 항목을 모두 가져오려면 IEnumerable을 반복하기만 하면 됩니다.

테이블에 단순 기본 키(파티션 키)가 있는 경우에는 Query 메서드를 사용할 수 없습니다. 대신 Load 메서드를 사용하면 파티션 키를 입력하여 항목을 가져올 수 있습니다.

> **Note**
>
> 백그라운드에서 이 작업을 수행하려면 QueryAsync 메서드를 대신 사용하세요.

Save

지정한 객체를 테이블에 저장합니다. 입력 객체에 지정된 기본 키가 테이블에 존재하지 않는 경우 메서드가 테이블에 새 항목을 추가합니다. 기본 키가 있는 경우 메서드가 기존 항목을 업데이트합니다.

낙관적 잠금이 구성된 경우 항목의 서버 측 버전과 클라이언트 측 버전이 일치하는 경우에만 업데이트가 성공합니다. 자세한 내용은 DynamoDB 및 AWS SDK for .NET 객체 지속성 모델을 사용하여 낙관적 잠금 수행 단원을 참조하십시오.

> **Note**
>
> 백그라운드에서 이 작업을 수행하려면 SaveAsync 메서드를 대신 사용하세요.

스캔

전체 테이블 스캔을 수행합니다.

스캔 조건을 지정하여 스캔 결과를 필터링할 수 있습니다. 스캔 조건은 테이블의 어느 속성 상에서든지 평가될 수 있습니다. DynamoDB의 ProductCatalog 테이블에 클라이언트 측 Book 클래스가 매핑되어 있는 경우 다음 C# 코드 예제에서는 테이블을 스캔하고 가격이 0보다 작은 책 항목만 반환합니다.

Example

```
IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
                    new ScanCondition("Price", ScanOperator.LessThan, price),
                    new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
      );
```

Scan 메서드는 "지연 로딩된(lazy-loaded)" IEnumerable 컬렉션을 반환합니다. 즉, 처음에는 결과 페이지를 하나만 반환하고, 필요에 따라 서비스를 호출하여 다음 페이지를 반환합니다. 일치하는 항목을 모두 가져오려면 IEnumerable을 반복하기만 하면 됩니다.

성능 문제 때문에 테이블 스캔을 피하고 테이블을 쿼리해야 합니다.

> **Note**
>
> 백그라운드에서 이 작업을 수행하려면 ScanAsync 메서드를 대신 사용하세요.

ToDocument

클래스 인스턴스에서 Document 문서 모델 클래스의 인스턴스가 반환됩니다.

이것은 문서 모델 클래스를 객체 지속성 모델과 함께 사용하여 데이터 작업을 수행할 때 유용합니다. AWS SDK for .NET에 의해 제공되는 문서 모델 클래스에 대한 자세한 내용은 DynamoDB에서 .NET 문서 모델을 사용하여 작업 단원을 참조하세요.

샘플 Forum 테이블에 클라이언트 측 클래스가 매핑되어 있는 경우 다음 C# 코드 예제와 같이 DynamoDBContext를 사용하여 Forum 테이블에서 항목을 Document 객체로 얻을 수 있습니다.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

Forum forum101 = context.Load<Forum>(101); // Retrieve a forum by primary key.
Document doc = context.ToDocument<Forum>(forum101);
```

### DynamoDBContext에 대한 옵션 파라미터 지정

객체 지속성 모델을 사용하는 경우, DynamoDBContext에 대하여 다음과 같은 선택적 파라미터를 지정할 수 있습니다.

- **ConsistentRead** - Load, Query 또는 Scan 작업을 사용하여 데이터를 검색할 때 이 선택적 파라미터를 추가하여 데이터의 최신 값을 요청할 수 있습니다.
- **IgnoreNullValues** - 이 파라미터는 DynamoDBContext에 Save 작업 동안 속성의 null 값을 무시하도록 지시합니다. 이 파라미터가 false이거나 설정되지 않은 경우 null 값은 특정 속성을 삭제하라는 지시로 해석됩니다.
- **SkipVersionCheck** - 이 파라미터는 항목을 저장하거나 삭제할 때 DynamoDBContext에 버전을 비교하지 않도록 지시합니다. 버전 관리에 대한 자세한 내용은 DynamoDB 및 AWS SDK for .NET 객체 지속성 모델을 사용하여 낙관적 잠금 수행 섹션을 참조하십시오.
- **TableNamePrefix** - 모든 테이블 이름에서 특정 문자열을 접두사로 사용합니다. 이 파라미터가 null이거나 설정되지 않은 경우, 어떤 접두사도 사용되지 않습니다.
- DynamoDBEntryConversion - 클라이언트가 사용하는 변환 스키마를 지정합니다. 이 파라미터를 버전 V1 또는 V2로 설정할 수 있습니다. 기본 버전은 V1입니다.

  설정한 버전에 따라 이 파라미터의 동작이 달라집니다. 예시:

  - V1에서는 bool 데이터 형식이 N 숫자 형식으로 변환됩니다. 여기서 0은 false, 1은 true를 나타냅니다. V2에서는 bool이 BOOL로 변환됩니다.
  - V2에서는 목록과 배열이 HashSet과 함께 그룹화되지 않습니다. 숫자로 구성된 목록과 배열, 문자열 기반 형식, 이진 기반 형식은 L(목록) 형식으로 변환됩니다. 이 형식은 목록을 업데이트하기 위해 빈 상태로 보낼 수 있습니다. 이는 빈 목록이 유선으로 전송되지 않는 V1과는 다릅니다.

    V1에서는 목록, HashSet 및 배열과 같은 컬렉션 형식이 동일하게 취급됩니다. 숫자 목록, HashSet 및 배열은 NS(숫자 집합) 형식으로 변환됩니다.

  다음 예제에서는 변환 스키마 버전을 V2로 설정하여 .NET 형식과 DynamoDB 데이터 형식 간의 변환 동작을 변경합니다.

```
var config = new DynamoDBContextConfig
{
    Conversion = DynamoDBEntryConversion.V2
};
var contextV2 = new DynamoDBContext(client, config);
```

다음 C# 예제에서는 앞에 나온 선택적 파라미터 중 두 개인 ConsistentRead 및 SkipVersionCheck를 지정하여 새 DynamoDBContext를 생성합니다.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context =
       new DynamoDBContext(client, new DynamoDBContextConfig { ConsistentRead = true,
 SkipVersionCheck = true});
```

DynamoDBContext에는 이 컨텍스트를 사용하여 보내는 각 요청과 함께 이러한 선택적 파라미터가 포함되어 있습니다.

이러한 파라미터를 DynamoDBContext 수준에 설정하는 대신 다음 C# 코드 예제와 같이 DynamoDBContext를 사용하여 실행하는 개별 작업에 대해 이러한 파라미터를 지정할 수 있습니다. 이 예제는 특정 책 항목을 로드합니다. DynamoDBContext의 Load 메서드는 선택적 파라미터인 ConsistentRead 및 SkipVersionCheck를 지정합니다.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context = new DynamoDBContext(client);
Book bookItem = context.Load<Book>(productId,new DynamoDBContextConfig{ ConsistentRead
 = true, SkipVersionCheck = true });
```

이 경우 DynamoDBContext에는 Get 요청을 보내는 경우에 한하여 이러한 파라미터가 포함됩니다.

### DynamoDB 및 AWS SDK for .NET 객체 지속성 모델을 사용하여 낙관적 잠금 수행

객체 지속성 모델의 낙관적 잠금 지원을 통해 항목을 업데이트하거나 삭제하기 전에 애플리케이션에 대한 항목 버전과 서버 측 항목 버전이 동일해집니다. 업데이트할 항목을 검색하는 경우 그러나 업데이트를 돌려보내기 전에 다른 애플리케이션이 같은 항목을 업데이트했습니다. 이 경우, 애플리케이션 항목의 오래된 복사본이 남게 됩니다. 낙관적 잠금이 없는 경우, 업데이트를 수행하면 다른 애플리케이션에 의해 생성된 업데이트를 덮어씁니다.

객체 지속성 모델의 낙관적 잠금 기능은 낙관적 잠금을 활성화하는 데 사용할 수 있는 DynamoDBVersion 태그를 제공합니다. 이 기능을 사용하려면 버전 번호를 저장하기 위해 클래스

에 속성을 추가합니다. 이 속성에 DynamoDBVersion 속성을 추가합니다. 처음 객체를 저장할 때 DynamoDBContext가 버전 번호를 할당하고, 항목이 업데이트될 때마다 이 값을 증가시킵니다.

업데이트나 삭제 요청은 클라이언트 측 객체 버전이 서버 측의 해당 항목 버전 번호와 일치해야만 성공합니다. 애플리케이션에 오래된 사본이 있는 경우에는 서버에서 최신 버전을 받아야만 항목을 업데이트하거나 삭제할 수 있습니다.

다음 C# 코드 예제에서는 객체 지속성 속성으로 Book 클래스를 정의하며 해당 클래스를 ProductCatalog 테이블로 매핑합니다. DynamoDBVersion 속성이 데코레이트된 클래스의VersionNumber 속성에는 버전 번호 값이 저장됩니다.

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]   //Partition key
    public int Id { get; set; }
    [DynamoDBProperty]
    public string Title { get; set; }
    [DynamoDBProperty]
    public string ISBN { get; set; }
    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }
    [DynamoDBVersion]
    public int? VersionNumber { get; set; }
  }
```

> **Note**
>
> DynamoDBVersion 속성은 int?와 같이 null이 허용된 숫자 기본 유형으로만 적용 가능합니다.

낙관적 잠금 전략은 DynamoDBContext 작업에 다음과 같은 영향을 끼칩니다.

- 새로운 항목이 있을 경우, DynamoDBContext는 초기 버전 번호를 0으로 할당합니다. 이후 기존 항목을 검색하고 해당 속성을 하나 이상 업데이트한 후 변경 사항을 저장하려고 해도 클라이언트 측 버전 번호와 서버 측 버전 번호가 일치해야만 저장 작업이 성공합니다. DynamoDBContext는 버전 번호를 증가시킵니다. 버전 번호를 설정할 필요는 없습니다.

- 다음 C# 코드 예제와 같이 Delete 메서드는 기본 키 값 또는 객체를 파라미터로 사용할 수 있는 오버로드를 제공합니다.

  Example

  ```
  DynamoDBContext context = new DynamoDBContext(client);
  ...
  // Load a book.
  Book book = context.Load<ProductCatalog>(111);
  // Do other operations.
  // Delete 1 - Pass in the book object.
  context.Delete<ProductCatalog>(book);

  // Delete 2 - Pass in the Id (primary key)
  context.Delete<ProductCatalog>(222);
  ```

  객체를 파라미터로 제공하는 경우 객체 버전이 서버 측 해당 항목 버전과 일치해야만 삭제가 성공합니다. 그러나 기본 키 값을 파라미터로 제공하는 경우 DynamoDBContext는 어떤 버전 번호도 알 수 없으며 버전 확인 없이 항목을 삭제합니다.

  단, 낙관적 잠금을 객체 지속성 모델 코드로 내부 구현할 경우에는 DynamoDB의 조건부 업데이트와 조건부 API 작업 삭제를 사용합니다.

### 낙관적 잠금 비활성화

낙관적 잠금을 비활성화하려면 SkipVersionCheck 구성 속성을 사용합니다. DynamoDBContext를 만들 때 이 속성을 설정할 수 있습니다. 이 경우 컨텍스트를 사용하여 수행하는 모든 요청에 대해 낙관적 잠금이 비활성화됩니다. 자세한 내용은 DynamoDBContext에 대한 옵션 파라미터 지정 단원을 참조하십시오.

속성을 컨텍스트 수준에 설정하는 대신 다음 C# 코드 예제와 같이 특정 작업에 대한 낙관적 잠금을 비활성화할 수 있습니다. 이 예제에서는 컨텍스트를 사용하여 책 항목을 삭제합니다. Delete 메서드는 선택적 SkipVersionCheck 속성을 true로 설정하여 버전 확인을 비활성화합니다.

Example

```
DynamoDBContext context = new DynamoDBContext(client);
// Load a book.
Book book = context.Load<ProductCatalog>(111);
...
// Delete the book.
```

```
context.Delete<Book>(book, new DynamoDBContextConfig { SkipVersionCheck = true });
```

### AWS SDK for .NET 객체 지속성 모델을 사용하여 DynamoDB에서 임의 데이터 매핑

지원되는 .NET 형식(지원되는 데이터 유형 참조) 외에도 애플리케이션에서 직접적으로 Amazon DynamoDB 형식에 매핑되지 않는 형식을 사용할 수 있습니다. 데이터를 임의 형식에서 DynamoDB 형식으로 또는 그 반대로 변환할 수 있는 변환기가 있는 한 객체 지속성 모델은 임의 형식의 데이터 저장을 지원합니다. 객체를 저장하거나 로드하는 동안 변환기 코드가 데이터를 변환합니다.

클라이언트 측에서 어떤 유형도 생성할 수 있습니다. 그러나 테이블에 저장되는 데이터는 DynamoDB 형식 중 하나이며 쿼리 및 스캔이 진행되는 동안 DynamoDB에 저장되어 있는 데이터에 대해 데이터 비교가 수행됩니다.

다음의 C# 코드 예제에서는 `Id`, `Title`, `ISBN` 및 `Dimension` 속성을 사용하여 `Book` 클래스를 정의하고 있습니다. `Dimension` 속성은 `Height`, `Width` 및 `Thickness` 속성을 기술하는 `DimensionType`에 포함되어 있습니다. 이 예제 코드는 데이터를 `DimensionType` 및 DynamoDB 문자열 형식 간에 전환하는 변환기 메서드 `ToEntry` 및 `FromEntry`를 제공합니다. 예를 들어, `Book` 인스턴스를 저장할 때 변환기는 "8.5x11x.05"와 같은 책 `Dimension` 문자열을 생성합니다. 책을 검색하면 문자열이 `DimensionType` 인스턴스로 변환됩니다.

이 예제에서는 `Book` 유형을 `ProductCatalog` 테이블로 매핑합니다. 샘플 `Book` 인스턴스를 저장하고, 해당 인스턴스를 검색하고, 해당 차원을 업데이트하고, 업데이트된 `Book`을 다시 저장합니다.

다음 예제를 테스트하기 위한 단계별 지침은 .NET 코드 예시 단원을 참조하세요.

#### Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class HighLevelMappingArbitraryData
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
```

```
        static void Main(string[] args)
        {
            try
            {
                DynamoDBContext context = new DynamoDBContext(client);

                // 1. Create a book.
                DimensionType myBookDimensions = new DimensionType()
                {
                    Length = 8M,
                    Height = 11M,
                    Thickness = 0.5M
                };

                Book myBook = new Book
                {
                    Id = 501,
                    Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
                    ISBN = "999-9999999999",
                    BookAuthors = new List<string> { "Author 1", "Author 2" },
                    Dimensions = myBookDimensions
                };

                context.Save(myBook);

                // 2. Retrieve the book.
                Book bookRetrieved = context.Load<Book>(501);

                // 3. Update property (book dimensions).
                bookRetrieved.Dimensions.Height += 1;
                bookRetrieved.Dimensions.Length += 1;
                bookRetrieved.Dimensions.Thickness += 0.2M;
                // Update the book.
                context.Save(bookRetrieved);

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }
    }
```

```
    [DynamoDBTable("ProductCatalog")]
    public class Book
    {
        [DynamoDBHashKey] //Partition key
        public int Id
        {
            get; set;
        }
        [DynamoDBProperty]
        public string Title
        {
            get; set;
        }
        [DynamoDBProperty]
        public string ISBN
        {
            get; set;
        }
        // Multi-valued (set type) attribute.
        [DynamoDBProperty("Authors")]
        public List<string> BookAuthors
        {
            get; set;
        }
        // Arbitrary type, with a converter to map it to DynamoDB type.
        [DynamoDBProperty(typeof(DimensionTypeConverter))]
        public DimensionType Dimensions
        {
            get; set;
        }
    }

    public class DimensionType
    {
        public decimal Length
        {
            get; set;
        }
        public decimal Height
        {
            get; set;
        }
        public decimal Thickness
        {
```

```
            get; set;
        }
    }

    // Converts the complex type DimensionType to string and vice-versa.
    public class DimensionTypeConverter : IPropertyConverter
    {
        public DynamoDBEntry ToEntry(object value)
        {
            DimensionType bookDimensions = value as DimensionType;
            if (bookDimensions == null) throw new ArgumentOutOfRangeException();

            string data = string.Format("{1}{0}{2}{0}{3}", " x ",
                              bookDimensions.Length, bookDimensions.Height,
bookDimensions.Thickness);

            DynamoDBEntry entry = new Primitive
            {
                Value = data
            };
            return entry;
        }

        public object FromEntry(DynamoDBEntry entry)
        {
            Primitive primitive = entry as Primitive;
            if (primitive == null || !(primitive.Value is String) ||
string.IsNullOrEmpty((string)primitive.Value))
                throw new ArgumentOutOfRangeException();

            string[] data = ((string)(primitive.Value)).Split(new string[] { " x " },
StringSplitOptions.None);
            if (data.Length != 3) throw new ArgumentOutOfRangeException();

            DimensionType complexData = new DimensionType
            {
                Length = Convert.ToDecimal(data[0]),
                Height = Convert.ToDecimal(data[1]),
                Thickness = Convert.ToDecimal(data[2])
            };
            return complexData;
        }
    }
```

```
}
```

## 이 개발자 안내서의 코드 예시 실행

AWS SDK는 Amazon DynamoDB에 대한 광범위한 지원을 다음 언어로 제공합니다.

- Java
- 브라우저에서의 JavaScript
- .NET
- Node.js
- PHP
- Python
- Ruby
- C++
- Go
- Android
- iOS

이 개발자 안내서의 코드 예제에서는 다음의 프로그래밍 언어를 사용하여 DynamoDB 작업을 보다 깊이 있게 다룹니다.

- Java 코드 예
- .NET 코드 예시

이 연습을 시작하려면 먼저 AWS 계정을 생성하고, 액세스 키 및 보안 키를 얻고, 컴퓨터에서 AWS Command Line Interface(AWS CLI)를 설정해야 합니다. 자세한 내용은 DynamoDB 설정(웹 서비스) 섹션을 참조하세요.

> **Note**
>
> DynamoDB 다운로드 가능 버전을 사용하는 경우, AWS CLI를 사용하여 테이블과 샘플 데이터를 생성해야 합니다. 또한 각 AWS CLI 명령으로 `--endpoint-url` 파라미터를 지정해야 합니다. 자세한 내용은 로컬 엔드포인트 설정 섹션을 참조하세요.

## DynamoDB에서 테이블 생성 및 코드 예시에 대한 데이터 로드

DynamoDB에서 테이블을 생성하고, 샘플 데이터 세트를 로드하고, 데이터를 쿼리하고, 데이터를 업데이트하는 방법에 대한 기본 사항은 아래를 참조하세요.

- 1단계: DynamoDB에 테이블 생성
- 2단계: DynamoDB 테이블에 데이터 쓰기
- 3단계: DynamoDB 테이블의 데이터 읽기
- 4단계: DynamoDB 테이블의 데이터 업데이트

## Java 코드 예

주제

- Java: AWS 보안 인증 정보 설정
- Java: AWS 리전 및 엔드포인트 설정

이 개발자 안내서에는 Java 코드 조각과 실행 준비된 프로그램이 포함되어 있습니다. 다음 단원에서 이들 코드 예제를 찾을 수 있습니다.

- DynamoDB의 항목 및 속성 작업
- DynamoDB의 테이블 및 데이터 작업
- DynamoDB에서 테이블 쿼리
- DynamoDB에서 테이블 스캔
- DynamoDB에서 보조 인덱스를 사용하여 데이터 액세스 개선
- Java 1.x: DynamoDBMapper
- DynamoDB Streams에 대한 변경 데이터 캡처

Eclipse와 함께 AWS Toolkit for Eclipse를 사용하여 빠르게 시작할 수 있습니다. 완벽한 기능의 IDE 외에 자동 업데이트가 포함된 AWS SDK for Java와 사전 구성된 AWS 애플리케이션 개발 템플릿도 포함되어 있습니다.

Java 코드 예제를 실행하려면(Eclipse 사용)

1. Eclipse IDE를 다운로드하고 설치합니다.
2. AWS Toolkit for Eclipse를 다운로드하여 설치합니다.

3. Eclipse를 시작하고 Eclipse 메뉴에서 파일, 새로 만들기, 기타를 차례대로 선택합니다.
4. Select a wizard에서 AWS, AWS Java Project, Next를 차례대로 선택합니다.
5. Create an AWS Java에서 다음을 수행합니다.

   a. 프로젝트 이름에 프로젝트의 이름을 입력합니다.

   b. Select Account의 목록에서 자격 증명 프로필을 선택합니다.

      AWS Toolkit for Eclipse를 처음 사용하는 경우, Configure AWS Accounts(계정 구성)를 선택하여 AWS 보안 인증 정보를 설정합니다.
6. 프로젝트를 생성하려면 Finish를 선택합니다.
7. Eclipse 메뉴에서 File, New, Class를 차례대로 선택합니다.
8. Java Class(Java 클래스)에서 이름에 클래스 이름을 입력(실행하려는 코드 예제와 동일한 이름 사용)한 다음 마침을 선택하여 클래스를 생성합니다.
9. 설명서 페이지에서 코드 예제를 Eclipse 에디터로 복사합니다.
10. 코드를 실행하려면 Eclipse 메뉴에서 실행을 선택합니다.

SDK for Java는 DynamoDB 작업을 위한 스레드 세이프(thread-safe) 클라이언트를 제공합니다. 모범 사례로서 애플리케이션에서 클라이언트 하나를 생성한 후 스레드 간에 재사용해야 합니다.

자세한 내용은 AWS SDK for Java을 참조하세요.

> **Note**
>
> 이 안내서의 코드 예제는 AWS SDK for Java의 최신 버전과 함께 사용해야 합니다. AWS Toolkit for Eclipse를 사용하는 경우 SDK for Java에 대해 자동 업데이트를 구성할 수 있습니다. Eclipse에서 이렇게 하려면 기본 설정으로 이동하고 AWS Toolkit, AWS SDK for Java, Download new SDKs automatically(새 SDK를 자동으로 다운로드)를 차례로 선택합니다.

### Java: AWS 보안 인증 정보 설정

SDK for Java에서는 런타임에 애플리케이션에 AWS 자격 증명을 제공해야 합니다. 이 가이드의 코드 예시에서는 AWS SDK for Java 개발자 안내서의 AWS 보안 인증 정보 설정에 설명된 대로 AWS 보안 인증 정보 파일을 사용한다고 가정합니다.

다음은 `~/.aws/credentials`라는 AWS 자격 증명 파일의 예입니다. 여기서 물결 문자(~)는 사용자의 홈 디렉터리입니다.

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java: AWS 리전 및 엔드포인트 설정

기본적으로 코드 예제는 미국 서부(오레곤) 리전에서 DynamoDB에 액세스합니다. AmazonDynamoDB 속성을 수정하여 리전을 변경할 수 있습니다.

다음 코드 예제에서는 새 AmazonDynamoDB를 인스턴스화합니다.

```
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import com.amazonaws.regions.Regions;
...
// This client will default to US West (Oregon)
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
.withRegion(Regions.US_WEST_2)
.build();
```

withRegion 메서드를 사용하면 사용할 수 있는 어떤 리전의 DynamoDB에 대해서도 코드를 실행할 수 있습니다. 전체 목록은 Amazon Web Services 일반 참조에서 AWS 리전 및 엔드포인트를 참조하세요.

컴퓨터에서 로컬로 DynamoDB를 사용하여 코드 예제를 실행하려면 다음과 같이 엔드포인트를 설정합니다.

AWS SDK V1

```
AmazonDynamoDB client =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
new AwsClientBuilder.EndpointConfiguration("http://localhost:8000", "us-west-2"))
.build();
```

AWS SDK V2

```
DynamoDbClient client = DynamoDbClient.builder()
    .endpointOverride(URI.create("http://localhost:8000"))
    // The region is meaningless for local DynamoDb but required for client builder
 validation
    .region(Region.US_EAST_1)
```

```
    .credentialsProvider(StaticCredentialsProvider.create(
    AwsBasicCredentials.create("dummy-key", "dummy-secret")))
    .build();
```

## .NET 코드 예시

주제

- .NET: AWS 보안 인증 정보 설정
- .NET: AWS 리전 및 엔드포인트 설정

이 안내서에는 .NET 코드 조각과 실행 준비된 프로그램이 포함되어 있습니다. 다음 단원에서 이들 코드 예제를 찾을 수 있습니다.

- DynamoDB의 항목 및 속성 작업
- DynamoDB의 테이블 및 데이터 작업
- DynamoDB에서 테이블 쿼리
- DynamoDB에서 테이블 스캔
- DynamoDB에서 보조 인덱스를 사용하여 데이터 액세스 개선
- DynamoDB에서 .NET 문서 모델을 사용하여 작업
- .NET 객체 지속성 모델 및 DynamoDB로 작업
- DynamoDB Streams에 대한 변경 데이터 캡처

AWS SDK for .NET 및 Toolkit for Visual Studio를 함께 사용하면 빠르게 시작할 수 있습니다.

.NET 코드 예제를 실행하려면((Visual Studio 사용)

1. Microsoft Visual Studio를 다운로드하여 설치합니다.
2. Toolkit for Visual Studio를 다운로드하여 설치합니다.
3. Visual Studio를 시작합니다. 파일, 새로 만들기, 프로젝트를 선택합니다.
4. New Project에서 AWS Empty Project를 선택한 다음 OK를 선택합니다.
5. AWS Access Credentials에서 Use existing profile을 선택하고, 목록에서 자격 증명 프로필을 선택한 다음 OK를 선택합니다.

   Toolkit for Visual Studio를 처음 사용하는 경우, Use a new profile(새 프로필 사용)을 선택하여 AWS 자격 증명을 설정합니다.

6. Visual Studio 프로젝트에서 프로그램 소스 코드(`Program.cs`) 탭을 선택합니다. 설명서 페이지에서 코드 예제를 Visual Studio 에디터로 복사하여 에디터에 표시된 일체의 코드를 대체합니다.
7. 유형 또는 네임스페이스 이름을...찾을 수 없습니다. 형식의 오류 메시지가 표시되면 다음과 같이 DynamoDB용 AWS SDK 어셈블리를 설치해야 합니다.

   a. Solution Exploer에서 프로젝트의 컨텍스트 메뉴를 열고(마우스 오른쪽 버튼 클릭) Manage NuGet Packages를 선택합니다.
   b. NuGet Package Manager에서 Browse를 선택합니다.
   c. 검색 상자에 **`AWSSDK.DynamoDBv2`**를 입력하고 검색이 완료되기를 기다립니다.
   d. AWSSDK.DynamoDBv2를 선택한 다음 설치를 선택합니다.
   e. 설치가 완료되면 Program.cs 탭을 선택하여 프로그램으로 돌아갑니다.
8. 코드를 실행하려면 Visual Studio 도구 모음에서 시작을 선택합니다.

SDK for .NET은 DynamoDB 작업을 위한 스레드 세이프(thread-safe) 클라이언트를 제공합니다. 모범 사례로서 애플리케이션에서 클라이언트 하나를 생성한 후 스레드 간에 재사용해야 합니다.

자세한 내용은 AWS SDK for .NET을 참조하세요.

> **Note**
>
> 이 안내서의 코드 예제는 AWS SDK for .NET의 최신 버전과 함께 사용해야 합니다.

### .NET: AWS 보안 인증 정보 설정

SDK for .NET에서는 런타임에 애플리케이션에 AWS 자격 증명을 제공해야 합니다. 이 가이드의 코드 예시에서는 AWS SDK for .NET 개발자 안내서의 SDK 스토어 사용에 설명된 대로 SDK 스토어를 사용하여 AWS 보안 인증 정보 파일을 관리한다고 가정합니다.

Toolkit for Visual Studio는 계정 수에 제한 없이 다수의 자격 증명 집합을 지원합니다. 각 집합을 프로필이라고 부릅니다. Visual Studio는 애플리케이션이 런타임 도중에도 AWS 자격 증명을 찾을 수 있도록 프로젝트의 `App.config` 파일에 항목을 추가합니다.

다음 예제에서는 Toolkit for Visual Studio를 사용하여 새 프로젝트를 만들 때 생성되는 기본 `App.config` 파일을 보여줍니다.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
```

```
    <appSettings>
    <add key="AWSProfileName" value="default"/>
    <add key="AWSRegion" value="us-west-2" />
 </appSettings>
</configuration>
```

런타임에 프로그램은 AWSProfileName 항목에 지정된 대로 default 자격 증명의 AWS 집합을 사용합니다. AWS 자격 증명 자체는 암호화된 형식으로 SDK 스토어에 보관됩니다. Toolkit for Visual Studio는 Visual Studio 안에서 자격 증명을 모두 관리할 수 있는 그래픽 사용자 인터페이스를 제공합니다. 자세한 내용은 AWS Toolkit for Visual Studio 사용 설명서에서 보안 인증 정보 지정을 참조하세요.

> **Note**
>
> 기본적으로 코드 예제는 미국 서부(오레곤) 리전에서 DynamoDB에 액세스합니다. App.config 파일의 AWSRegion 항목을 수정하여 리전을 변경할 수 있습니다. AWSRegion은 DynamoDB를 사용할 수 있는 어느 리전으로도 설정할 수 있습니다. 전체 목록은 Amazon Web Services 일반 참조에서 AWS 리전 및 엔드포인트를 참조하세요.

.NET: AWS 리전 및 엔드포인트 설정

기본적으로 코드 예제는 미국 서부(오레곤) 리전에서 DynamoDB에 액세스합니다. App.config 파일의 AWSRegion 항목을 수정하여 리전을 변경할 수 있습니다. 또는 AmazonDynamoDBClient 속성을 수정하여 리전을 변경할 수 있습니다.

다음 코드 예제에서는 새 AmazonDynamoDBClient를 인스턴스화합니다. 클라이언트가 수정되어 코드가 다른 리전의 DynamoDB에 대해 실행됩니다.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// This client will access the US East 1 region.
clientConfig.RegionEndpoint = RegionEndpoint.USEast1;
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

전체 리전 목록은 Amazon Web Services 일반 참조에서 AWS 리전 및 엔드포인트를 참조하세요.

컴퓨터에서 로컬로 DynamoDB를 사용하여 코드 예제를 실행하려면 다음과 같이 엔드포인트를 설정합니다.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// Set the endpoint URL
```

```
clientConfig.ServiceURL = "http://localhost:8000";
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

## DynamoDB 하위 수준 API

Amazon DynamoDB 하위 수준 API는 DynamoDB의 프로토콜 수준 인터페이스입니다. 이 수준에서는 모든 HTTP(S) 요청이 올바른 형식이어야 하며, 유효한 디지털 서명이 있어야 합니다.

AWS SDK는 사용자를 대신하여 하위 수준 DynamoDB API 요청을 구성하고 DynamoDB의 응답을 처리합니다. 이로써 하위 수준 세부 사항 대신 애플리케이션 로직에 집중할 수 있습니다. 그러나 하위 수준 DynamoDB API의 작동 방식에 대한 기본적 이해는 여전히 도움이 될 수 있습니다.

하위 수준 DynamoDB API에 대한 자세한 내용은 Amazon DynamoDB API 참조를 참조하세요.

> **Note**
>
> DynamoDB Streams의 자체 하위 수준 API는 DynamoDB의 그것과 별개이며, AWS SDK가 완벽하게 지원합니다.
> 자세한 내용은 DynamoDB Streams에 대한 변경 데이터 캡처 섹션을 참조하세요. 하위 수준 DynamoDB Streams API는 Amazon DynamoDB Streams API 참조를 참조하세요.

하위 수준 DynamoDB API는 JavaScript Object Notation(JSON)을 연결 프로토콜 형식으로 사용합니다. JSON은 데이터를 계층 구조로 나타내므로 데이터 값과 데이터 구조 모두 동시에 전달됩니다. 이름-값 페어는 `name:value` 형식으로 정의됩니다. 데이터 계층은 이름-값 페어의 중첩된 대괄호로 정의됩니다.

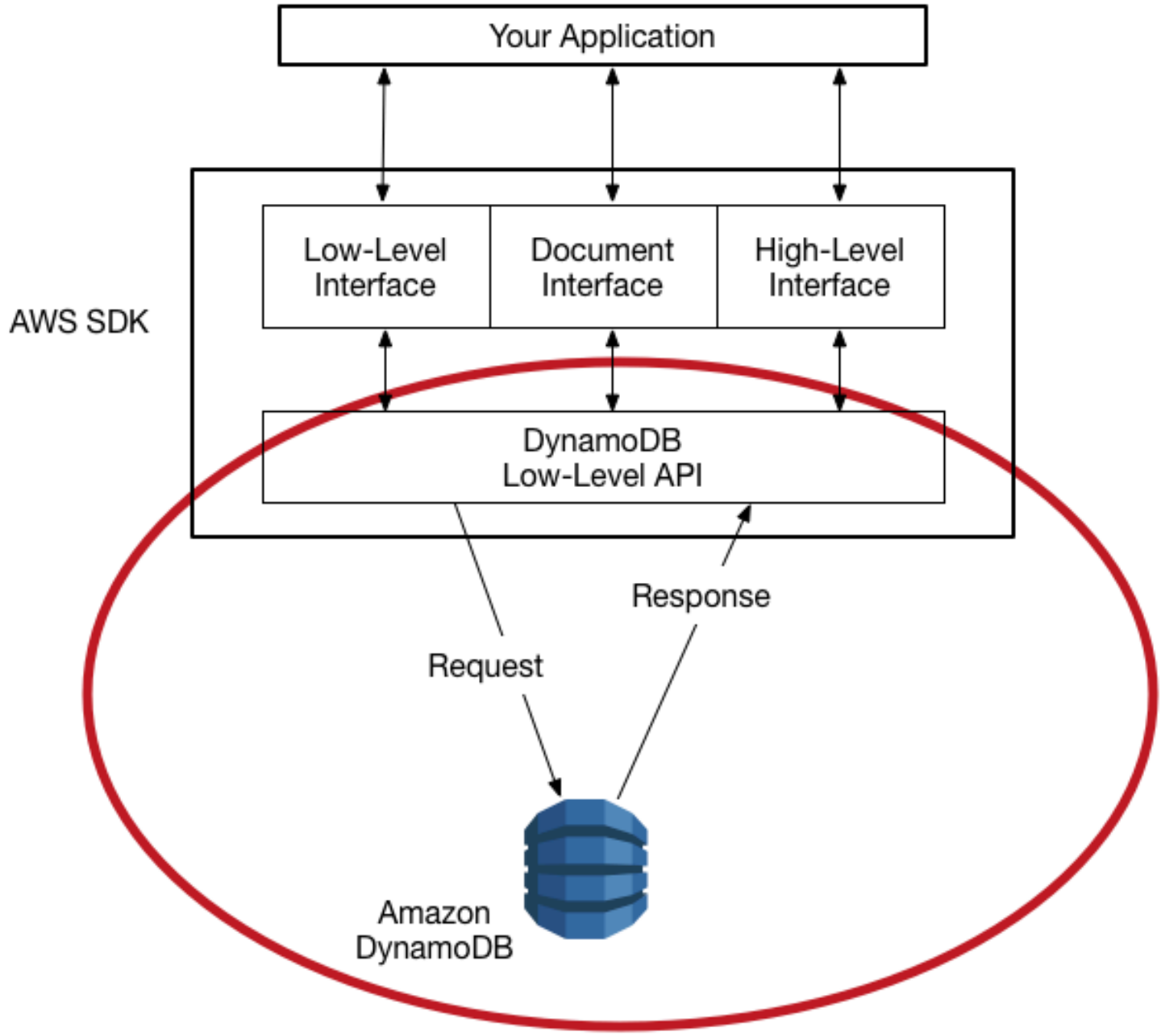
DynamoDB는 JSON을 오직 전송 프로토콜로 사용하며 스토리지 형식으로는 사용하지 않습니다. AWS SDK는 JSON을 사용하여 DynamoDB에 데이터를 보내고 DynamoDB는 JSON으로 응답합니다. DynamoDB는 데이터를 JSON 형식으로 지속적으로 저장하지 않습니다.

> **Note**
>
> JSON에 대한 자세한 내용은 `JSON.org` 웹사이트에서 JSON 소개를 참조하세요.

주제

- 요청 형식
- 응답 형식

- 데이터 형식 서술자
- 숫자 데이터
- 이진 데이터




## 요청 형식

DynamoDB 하위 수준 API는 HTTP(S) POST 요청을 입력으로 받아들입니다. AWS SDK는 이 요청을 구성합니다.

Pets(파티션 키) 및 AnimalType(정렬 키)으로 구성된 키 스키마가 포함된 Name이라는 이름의 테이블이 있는 경우 이 두 가지 속성 모두 string 유형입니다. Pets에서 항목을 검색하기 위해 AWS SDK는 다음 요청을 구성합니다.

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date>
X-Amz-Target: DynamoDB_20120810.GetItem

{
    "TableName": "Pets",
    "Key": {
        "AnimalType": {"S": "Dog"},
        "Name": {"S": "Fido"}
    }
}
```

이 요청에 대해 다음을 참조하세요.

- Authorization 헤더에는 DynamoDB가 요청을 인증하는 데 필요한 정보가 포함되어 있습니다. 자세한 내용은 Amazon Web Services 일반 참조의 AWS API 요청에 서명 및 서명 버전 4 서명 프로세스를 참조하세요.
- X-Amz-Target 헤더에는 DynamoDB 작업의 이름인 GetItem이 포함되어 있습니다. (여기에 하위 API 버전(이 경우에는 20120810)도 포함됩니다.)
- 요청의 페이로드(본문)에는 JSON 형식의 작업을 위한 파라미터가 포함되어 있습니다. GetItem 작업의 경우, 파라미터는 TableName 및 Key입니다.

## 응답 형식

DynamoDB는 요청을 수신하면 이를 처리하고 응답을 반환합니다. 이전에 나온 요청의 경우 다음 예제와 같이 HTTP(S) 응답 페이로드에 해당 작업 결과가 포함됩니다.

```
HTTP/1.1 200 OK
```

```
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
    "Item": {
        "Age": {"N": "8"},
        "Colors": {
            "L": [
                {"S": "White"},
                {"S": "Brown"},
                {"S": "Black"}
            ]
        },
        "Name": {"S": "Fido"},
        "Vaccinations": {
            "M": {
                "Rabies": {
                    "L": [
                        {"S": "2009-03-17"},
                        {"S": "2011-09-21"},
                        {"S": "2014-07-08"}
                    ]
                },
                "Distemper": {"S": "2015-10-13"}
            }
        },
        "Breed": {"S": "Beagle"},
        "AnimalType": {"S": "Dog"}
    }
}
```

이때 AWS SDK는 추가 처리를 위해 애플리케이션에 응답 데이터를 반환합니다.

> **Note**
>
> DynamoDB가 요청을 처리할 수 없는 경우 HTTP 오류 코드 및 메시지를 반환합니다. AWS SDK는 이를 예외 형식으로 애플리케이션에 전파합니다. 자세한 내용은 DynamoDB 관련 오류 처리 섹션을 참조하세요.

## 데이터 형식 서술자

하위 수준 DynamoDB API 프로토콜에서는 각 속성에 데이터 형식 서술자가 포함되어야 합니다. 데이터 형식 서술자는 DynamoDB에 각 속성을 해석하는 방법을 알려 주는 토큰입니다.

요청 형식 및 응답 형식의 예는 데이터 형식 서술자가 어떻게 사용되는지 보여 줍니다. `GetItem` 요청은 `Pets` 키 스키마 속성(`AnimalType` 및 `Name`)에 대해 `S`를 지정하며, 이는 `string` 유형입니다. `GetItem` 응답에는 `string`(`S`), `number`(`N`), `map`(`M`) 및 `list`(`L`) 형식의 Pets 항목이 포함됩니다.

다음은 DynamoDB 데이터 형식 서술자의 전체 목록입니다.

- **`S`** - 문자열
- **`N`** - 숫자
- **`B`** - 이진수
- **`BOOL`** - 부울
- **`NULL`** - Null
- **`M`** - 맵
- **`L`** - 목록
- **`SS`** - 문자열 집합
- **`NS`** - 숫자 집합
- **`BS`** - 이진수 집합

> **Note**
>
> DynamoDB 데이터 형식에 대한 자세한 내용은 데이터 타입 단원을 참조하세요.

## 숫자 데이터

프로그래밍 언어마다 JSON 지원 수준이 다릅니다. 경우에 따라서는 JSON 문서를 확인하고 구문 분석하는 데 타사 라이브러리를 사용해야 할 수도 있습니다.

일부 타사 라이브러리는 JSON 숫자 형식을 기초로 `int`, `long` 또는 `double` 같은 자체 형식을 제공합니다. 그러나 DynamoDB의 기본 숫자 데이터 형식은 이러한 다른 데이터 형식에 정확히 매핑되지 않으므로 이러한 형식 구분은 충돌을 야기할 수 있습니다. 뿐만 아니라 많은 JSON 라이브러리는 고정 전체 자릿수 숫자 값을 처리하지 않으며, 소수점을 포함하는 숫자열의 double 데이터 형식을 자동으로 유추합니다.

DynamoDB는 이러한 문제를 해결하기 위해 데이터 손실이 없는 단일 숫자 형식을 제공합니다. 또한 사용자의 동의 없이 double 값으로 암시적으로 변환되는 것을 방지하기 위해 DynamoDB는 숫자 값의 데이터 전송에 문자열을 사용합니다. 이러한 접근 방식은 속성 값을 업데이트하는 데 유연성을 제공하는 동시에 "1", "2", "3"을 적절한 순서로 입력하는 등 적절한 정렬 의미 체계를 유지합니다.

애플리케이션에 숫자 전체 자릿수가 중요한 경우, DynamoDB로 전달하기 전에 숫자 값을 문자열로 변환해야 합니다.

### 이진 데이터

DynamoDB는 이진 속성을 지원합니다. 그러나 JSON은 이진 데이터 인코딩은 기본적으로 지원하지 않습니다. 요청에서 이진 데이터를 보내려면 Base64 형식으로 인코딩해야 합니다. DynamoDB는 요청을 받아 Base64를 다시 이진수로 디코딩합니다.

DynamoDB에서 사용되는 base64 인코딩 체계는 IETF(Internet Engineering Task Force) 웹 사이트의 RFC 4648에 설명되어 있습니다.

## Python과 Boto3를 사용한 Amazon DynamoDB 프로그래밍

이 안내서는 Python을 통해 Amazon DynamoDB를 사용하려는 프로그래머에게 지침을 제공합니다. 다양한 추상화 계층, 구성 관리, 오류 처리, 재시도 정책 제어, 연결 유지 관리 등에 대해 알아봅니다.

주제

- Boto 소개
- Boto 설명서 사용
- 클라이언트 및 리소스 추상화 계층 이해
- batch_writer 테이블 리소스 사용
- 클라이언트 및 리소스 계층을 탐색하는 추가 코드 예시
- Client 및 Resource 객체가 세션 및 스레드와 상호 작용하는 방식 이해
- Config 객체 사용자 지정
- 오류 처리
- 로깅
- 이벤트 후크
- 페이지 매김 및 페이지네이터
- Waiters

# Boto 소개

공식 AWS SDK for Python(일반적으로 Boto3라고 함)을 사용하여 Python에서 DynamoDB에 액세스할 수 있습니다. Boto('보토'라고 발음함)라는 이름은 아마존 강에 서식하는 민물 돌고래에서 유래했습니다. Boto3 라이브러리는 2015년에 처음 출시된 이 라이브러리의 세 번째 메이저 버전입니다. Boto3 라이브러리는 DynamoDB뿐만 아니라 모든 AWS 서비스를 지원하므로 상당히 큽니다. 이 오리엔테이션에서는 Boto3에서 DynamoDB와 관련된 부분만 대상으로 합니다.

Boto는 GitHub에서 호스팅되는 오픈 소스 프로젝트로 AWS에서 유지 관리하고 게시합니다. Boto는 Botocore와 Boto3라는 두 개의 패키지로 나뉘어 있습니다.

- Botocore는 하위 수준 기능을 제공합니다. Botocore에서는 클라이언트, 세션, 자격 증명, 구성 및 예외 클래스를 찾을 수 있습니다.
- Boto3는 Botocore를 기반으로 빌드됩니다. Python과 더 유사한 더 높은 수준의 인터페이스를 제공합니다. 특히 DynamoDB 테이블을 리소스로 노출하고 하위 수준의 서비스 지향적 클라이언트 인터페이스에 비해 더 단순하고 세련된 인터페이스를 제공합니다.

이러한 프로젝트는 GitHub에서 호스팅되므로 소스 코드를 보거나 미해결 문제를 추적하거나 직접 문제를 제출할 수 있습니다.

## Boto 설명서 사용

다음 리소스로 Boto 설명서 사용을 시작하세요.

- 패키지 설치를 위한 탄탄한 출발점 역할을 하는 Quickstart 섹션부터 시작하세요. 아직 설치하지 않은 경우 Boto3를 설치하는 방법에 대한 지침을 보려면 Quickstart 섹션을 방문하세요(Boto3는 AWS Lambda와 같은 AWS 서비스 내에서 자동으로 제공되는 경우가 많습니다).
- 그 다음에는 설명서의 DynamoDB 안내서를 중점적으로 살펴보세요. 테이블 생성 및 삭제, 항목 조작, 배치 작업 실행, 쿼리 실행, 스캔 수행 등 기본적인 DynamoDB 활동을 수행하는 방법을 보여줍니다. 이 예시에서는 리소스 인터페이스를 사용합니다. `boto3.resource('dynamodb')`가 표시되면 상위 수준 리소스 인터페이스를 사용하고 있다는 뜻입니다.
- 안내서를 살펴본 후 DynamoDB 참조를 검토할 수 있습니다. 이 랜딩 페이지는 사용 가능한 클래스와 메서드의 전체 목록을 제공합니다. 상단에 `DynamoDB.Client` 클래스가 있습니다. 이 클래스는 모든 컨트롤 플레인 및 데이터 영역 작업에 대한 하위 수준 액세스를 제공합니다. 하단에는 `DynamoDB.ServiceResource` 클래스가 있습니다. 이 클래스는 더 상위 수준의 Python 스타일 인터페이스입니다. 이를 통해 테이블을 만들거나, 테이블 전체에 걸쳐 배치 작업을 수행하거나, 테이블별 작업을 위한 `DynamoDB.ServiceResource.Table` 인스턴스를 확보할 수 있습니다.

# 클라이언트 및 리소스 추상화 계층 이해

사용할 두 인터페이스는 클라이언트 인터페이스와 리소스 인터페이스입니다.

- 하위 수준 클라이언트 인터페이스는 기본 서비스 API에 대한 일대일 매핑을 제공합니다. DynamoDB에서 제공하는 모든 API는 클라이언트를 통해 사용할 수 있습니다. 즉, 클라이언트 인터페이스가 완전한 기능을 제공할 수 있지만 사용하기가 더 복잡한 경우가 많습니다.
- 상위 수준 리소스 인터페이스는 기본 서비스 API의 일대일 매핑을 제공하지 않습니다. 하지만 `batch_writer`와 같이 보다 편리하게 서비스에 액세스할 수 있는 메서드를 제공합니다.

다음은 클라이언트 인터페이스를 사용하여 항목을 삽입하는 예시입니다. 모든 값이 유형(문자열의 경우 'S', 숫자의 경우 'N')을 나타내는 키와 문자열 값으로 구성된 맵으로 전달되는 것을 확인할 수 있습니다. 이를 DynamoDB JSON 형식이라고 합니다.

```
import boto3

dynamodb = boto3.client('dynamodb')

dynamodb.put_item(
    TableName='YourTableName',
    Item={
        'pk': {'S': 'id#1'},
        'sk': {'S': 'cart#123'},
        'name': {'S': 'SomeName'},
        'inventory': {'N': '500'},
        # ... more attributes ...
    }
)
```

다음은 동일한 `PutItem` 작업을 리소스 인터페이스를 사용하여 수행한 것입니다. 데이터 입력은 암시적입니다.

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

table.put_item(
    Item={
```

```
        'pk': 'id#1',
        'sk': 'cart#123',
        'name': 'SomeName',
        'inventory': 500,
        # ... more attributes ...
    }
)
```

필요한 경우 boto3에서 제공하는 TypeSerializer 및 TypeDeserializer 클래스를 사용하여 일반 JSON과 DynamoDB JSON 간에 변환할 수 있습니다.

```
def dynamo_to_python(dynamo_object: dict) -> dict:
    deserializer = TypeDeserializer()
    return {
        k: deserializer.deserialize(v)
        for k, v in dynamo_object.items()
    }

def python_to_dynamo(python_object: dict) -> dict:
    serializer = TypeSerializer()
    return {
        k: serializer.serialize(v)
        for k, v in python_object.items()
    }
```

클라이언트 인터페이스를 사용하여 쿼리를 수행하는 방법은 다음과 같습니다. 쿼리를 JSON 구성으로 표현합니다. 잠재적 키워드 충돌을 처리하기 위해 변수 대체가 필요한 KeyConditionExpression 문자열을 사용합니다.

```
import boto3

client = boto3.client('dynamodb')

# Construct the query
response = client.query(
    TableName='YourTableName',
    KeyConditionExpression='pk = :pk_val AND begins_with(sk, :sk_val)',
    FilterExpression='#name = :name_val',
    ExpressionAttributeValues={
        ':pk_val': {'S': 'id#1'},
        ':sk_val': {'S': 'cart#'},
```

```
        ':name_val': {'S': 'SomeName'},
    },
    ExpressionAttributeNames={
        '#name': 'name',
    }
)
```

리소스 인터페이스를 사용한 동일한 쿼리 작업을 줄이고 단순화할 수 있습니다.

```
import boto3
from boto3.dynamodb.conditions import Key, Attr

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('YourTableName')

response = table.query(
    KeyConditionExpression=Key('pk').eq('id#1') & Key('sk').begins_with('cart#'),
    FilterExpression=Attr('name').eq('SomeName')
)
```

마지막 예로 테이블의 대략적인 크기(테이블에 보관되며 약 6시간마다 업데이트되는 메타데이터)를 구한다고 가정해 보겠습니다. 클라이언트 인터페이스에서 `describe_table()` 작업을 수행하고 반환된 JSON 구조에서 답을 가져옵니다.

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.describe_table(TableName='YourTableName')
size = response['Table']['TableSizeBytes']
```

리소스 인터페이스에서 테이블이 설명 작업을 암시적으로 수행하고 데이터를 속성으로 직접 표시합니다.

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')
size = table.table_size_bytes
```

> **Note**
>
> 개발에 클라이언트 인터페이스를 사용할지 리소스 인터페이스를 사용할지 고려할 때는 리소스 인터페이스에 신규 기능이 추가되지 않는다는 점을 유의하세요. 리소스 설명서에는 "AWS Python SDK 팀은 boto3의 리소스 인터페이스에 신규 기능을 추가할 의도가 없습니다. 기존 인터페이스는 boto3의 수명 주기 동안 계속 작동할 것입니다. 클라이언트 인터페이스를 통해 새로운 서비스 기능에 액세스할 수 있습니다."라고 명시되어 있습니다.

## batch_writer 테이블 리소스 사용

상위 수준 테이블 리소스에서만 사용할 수 있는 편리한 기능 중 하나는 `batch_writer`입니다. DynamoDB는 한 번의 네트워크 요청으로 최대 25개의 넣기 또는 삭제 작업을 허용하는 배치 쓰기 작업을 지원합니다. 이와 같은 배치 작업은 네트워크 왕복을 최소화하여 효율성을 개선합니다.

하위 수준 클라이언트 라이브러리에서는 `client.batch_write_item()` 작업을 사용하여 배치를 실행합니다. 작업을 25개씩 배치로 수동으로 분할해야 합니다. 각 작업 후에는 처리되지 않은 항목 목록도 수신을 요청해야 합니다. 쓰기 작업 중 일부는 성공하지만 다른 작업은 실패할 수 있습니다. 그런 다음 처리되지 않은 항목을 이후 `batch_write_item()` 작업으로 다시 전달해야 합니다. 보일러플레이트 코드가 상당히 많습니다.

Table.batch_writer 메서드는 객체를 배치로 작성하기 위한 컨텍스트 관리자를 만듭니다. 마치 한 번에 하나씩 항목을 쓰는 것처럼 보이지만 내부적으로는 항목을 버퍼링하고 배치로 전송하는 인터페이스를 제공합니다. 또한 처리되지 않은 항목 재시도를 암시적으로 처리합니다.

```
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

movies = # long list of movies in {'pk': 'val', 'sk': 'val', etc} format
with table.batch_writer() as writer:
    for movie in movies:
        writer.put_item(Item=movie)
```

## 클라이언트 및 리소스 계층을 탐색하는 추가 코드 예시

클라이언트와 리소스를 모두 사용하여 다양한 함수의 사용법을 탐색하는 다음 코드 샘플 리포지토리를 참조할 수도 있습니다.

- 공식 AWS 단일 작업 코드 예시
- 공식 AWS 시나리오 지향 코드 예시
- 커뮤니티에서 유지 관리하는 단일 작업 코드 예시

## Client 및 Resource 객체가 세션 및 스레드와 상호 작용하는 방식 이해

Resource 객체는 스레드로부터 안전하지 않으므로 스레드 또는 프로세스 간에 공유해서는 안 됩니다. 자세한 내용은 Resource 가이드를 참조하세요.

반면 Client 객체는 특정 고급 기능을 제외하고는 일반적으로 스레드로부터 안전합니다. 자세한 내용은 Clients 가이드를 참조하세요.

Session 객체는 스레드로부터 안전하지 않습니다. 따라서 다중 스레드 환경에서 Client나 Resource를 만들 때마다 먼저 새 Session을 만든 다음 Session에서 Client 또는 Resource를 만들어야 합니다. 자세한 내용은 Sessions 가이드를 참조하세요.

`boto3.resource()`를 직접 호출하면 암시적으로 기본 Session을 사용하게 됩니다. 이는 단일 스레드 코드를 작성할 때 편리합니다. 다중 스레드 코드를 작성할 때는 먼저 각 스레드에 대해 새 Session을 생성한 다음 해당 Session에서 리소스를 검색하는 것이 좋습니다.

```
# Explicitly create a new Session for this thread
session = boto3.Session()
dynamodb = session.resource('dynamodb')
```

## Config 객체 사용자 지정

Client 또는 Resource 객체를 구성할 때 선택적으로 명명된 파라미터를 전달하여 동작을 사용자 지정할 수 있습니다. `config`라는 파라미터를 사용하면 다양한 기능을 사용할 수 있습니다. 이 파라미터는 `botocore.client.Config`의 인스턴스이며 Config에 대한 참조 설명서에 사용자가 제어할 수 있도록 노출되는 모든 내용이 나와 있습니다. Configuration 가이드는 개요를 알아보기에 좋습니다.

> **Note**
>
> 이러한 동작 설정의 대부분은 Session 수준에서, AWS 구성 파일 내에서 또는 환경 변수로 수정할 수 있습니다.

제한 시간을 위한 Config

사용자 지정 구성의 용도 중 하나는 네트워킹 동작을 조정하는 것입니다.

- connect_timeout(float 또는 int) - 연결을 시도할 때 제한 시간 예외가 발생하기까지의 시간(초)입니다. 기본값은 60초입니다.
- read_timeout (float 또는 int) - 연결에서 읽기를 시도할 때 제한 시간 예외가 발생하기까지의 시간(초)입니다. 기본값은 60초입니다.

DynamoDB에서는 60초의 제한 시간은 너무 깁니다. 즉, 일시적인 네트워크 결함으로 인해 재시도하기 전에 클라이언트에게 1분 정도의 지연 시간이 발생한다는 뜻입니다. 다음 코드는 제한 시간을 1초로 단축합니다.

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

제한 시간에 대한 자세한 설명은 Tuning AWS Java SDK HTTP request settings for latency-aware DynamoDB applications를 참조하세요. 참고로 Java SDK에는 Python보다 더 많은 제한 시간 구성이 있습니다.

연결 유지를 위한 Config

botocore 1.27.84 이상을 사용하는 경우 TCP 연결 유지도 제어할 수 있습니다.

- tcp_keepalive(bool) - `False`(기본값 `True`)으로 설정된 경우 새 연결을 만들 때 사용하는 TCP 연결 유지 소켓 옵션을 활성화합니다. 이 기능은 botocore 1.27.84부터 사용할 수 있습니다.

TCP 연결 유지를 `True`로 설정하면 평균 지연 시간을 줄일 수 있습니다. 다음은 올바른 botocore 버전을 사용하는 경우 TCP 연결 유지를 조건부로 true로 설정하는 샘플 코드입니다.

```
import botocore
import boto3
from botocore.config import Config
from distutils.version import LooseVersion

required_version = "1.27.84"
```

```
current_version = botocore.__version__

my_config = Config(
   connect_timeout = 0.5,
   read_timeout = 0.5
)
if LooseVersion(current_version) > LooseVersion(required_version):
    my_config = my_config.merge(Config(tcp_keepalive = True))

dynamodb = boto3.resource('dynamodb', config=my_config)
```

> **Note**
>
> TCP 연결 유지는 HTTP 연결 유지와 다릅니다. TCP 연결 유지를 사용하면 기본 운영 체제가 소켓 연결을 통해 작은 패킷을 전송하여 연결을 유지하고 연결 해제를 즉시 감지합니다. HTTP 연결 유지를 사용하면 기본 소켓에 구축된 웹 연결이 재사용됩니다. Boto3에서는 HTTP 연결 유지가 항상 활성화되어 있습니다.

유휴 연결을 유지할 수 있는 기간에는 한도가 있습니다. 연결이 유휴 상태이지만 다음 요청에서 이미 설정된 연결을 사용하도록 하려면 주기적으로(예: 1분마다) 요청을 보내는 것을 고려하세요.

재시도를 위한 Config

이 구성에서는 원하는 재시도 동작을 지정할 수 있는 retry라는 딕셔너리도 허용합니다. SDK에서 오류를 수신하고 오류가 일시적 유형인 경우 SDK 내에서 재시도가 발생합니다. 오류가 내부적으로 재시도되어 결국 성공적인 응답이 나오는 경우 직접 호출하는 코드의 관점에서 보면 오류가 나타나지 않고 지연 시간이 약간 길어질 뿐입니다. 지정할 수 있는 값은 다음과 같습니다.

- max_attempts - 단일 요청에서 수행할 수 있는 최대 재시도 횟수를 나타내는 정수입니다. 예를 들어 이 값을 2로 설정하면 초기 요청 이후 요청이 최대 두 번 재시도됩니다. 이 값을 0으로 설정하면 초기 요청 이후 재시도하지 않습니다.
- total_max_attempts - 단일 요청에서 수행할 수 있는 총 시도 횟수를 나타내는 정수입니다. 여기에는 초기 요청이 포함되므로 값이 1이면 요청이 재시도되지 않음을 나타냅니다. `total_max_attempts`와 `max_attempts`를 모두 제공하는 경우 `total_max_attempts`가 우선합니다. `total_max_attempts`는 `AWS_MAX_ATTEMPTS` 환경 변수 및 `max_attempts` 구성 파일 값에 매핑되기 때문에 `max_attempts`보다 우선하는 것입니다.
- mode - Botocore가 사용해야 하는 재시도 모드의 유형을 나타내는 문자열입니다. 유효한 값은 다음과 같습니다.

- legacy - 기본 모드입니다. 첫 번째 재시도를 50ms 동안 기다렸다가 기본 계수 2의 지수 백오프를 사용합니다. DynamoDB의 경우 위 값으로 재정의하지 않는 한 모두 합쳐 최대 10회까지 시도합니다.

  > **Note**
  >
  > 지수 백오프를 사용할 경우 마지막 시도는 거의 13초간 기다리게 됩니다.

- standard - 다른 AWS SDK와 더 일관적이기 때문에 표준이라는 이름이 지정되었습니다. 첫 번째 재시도를 위해 0ms에서 1,000ms 사이의 임의의 시간 동안 기다립니다. 다시 재시도해야 하는 경우 0ms에서 1,000ms 사이의 또 다른 임의의 시간을 선택하여 2를 곱합니다. 추가 재시도가 필요한 경우 같은 범위에서 임의로 선택한 시간에 4를 곱하는 식으로 반복합니다. 각 대기 시간은 20초로 제한됩니다. 이 모드는 `legacy` 모드보다 더 많이 감지된 장애 조건에 대해 재시도를 수행합니다. DynamoDB의 경우 위 값으로 재정의하지 않는 한 모두 합쳐 최대 3회까지 시도합니다.
- adaptive - 표준 모드의 모든 기능을 포함하고 자동 클라이언트 측 제한도 포함하는 실험적인 재시도 모드입니다. 적응형 속도 제한을 사용하면 SDK가 요청 전송 속도를 늦춰 AWS 서비스의 요청 용량을 더 잘 수용할 수 있습니다. 이 모드는 동작이 변경될 수 있는 임시 모드입니다.

이러한 재시도 모드의 확장된 정의는 재시도 가이드 및 SDK 참조의 Retry behavior 주제에서 확인할 수 있습니다.

다음은 모두 합쳐 최대 3회의 요청(재시도 횟수 2회)을 수행하는 `legacy` 재시도 정책을 명시적으로 사용하는 예시입니다.

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0,
   retries = {
     'mode': 'legacy',
     'total_max_attempts': 3
   }
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

DynamoDB는 가용성이 높고 지연 시간이 짧은 시스템이므로 기본 제공되는 재시도 정책이 허용하는 것보다 더 공격적으로 재시도 속도를 높이는 것이 좋습니다. Boto3에 의존하여 암시적 재시도를 수행

하는 대신 최대 시도 횟수를 0으로 설정하고, 직접 예외를 포착하고, 자체 코드에서 적절하게 재시도함으로써 자체 재시도 정책을 구현할 수 있습니다.

자체 재시도 정책을 관리하는 경우 제한과 오류를 구분하는 것이 좋습니다.

- 제한(ProvisionedThroughputExceededException 또는 ThrottlingException으로 표시됨)은 정상 서비스에서 DynamoDB 테이블 또는 파티션의 읽기 또는 쓰기 용량을 초과했음을 알리는 것입니다. 밀리초가 지날 때마다 약간 더 많은 읽기 또는 쓰기 용량을 사용할 수 있게 되므로 빠르게(예: 50ms마다) 재시도하여 새로 확보된 용량에 액세스하려고 시도할 수 있습니다. 제한을 사용하면 지수 백오프가 특별히 필요하지 않습니다. 제한은 DynamoDB가 반환하기에 가볍고 요청당 요금이 부과되지 않기 때문입니다. 지수 백오프는 이미 가장 오래 기다린 클라이언트 스레드에 더 긴 지연을 할당하여 통계적으로 p50과 p99를 밖으로 확장합니다.
- 오류(특히 InternalServerError 또는 ServiceUnavailable로 표시됨)는 서비스에 일시적인 문제가 있음을 나타냅니다. 테이블 전체에 대한 것일 수도 있고, 읽거나 쓰고 있는 파티션에만 해당될 수도 있습니다. 오류가 발생하면 재시도 전에 더 오래(예: 250ms 또는 500ms) 중단하고 지터를 사용하여 재시도에 시차를 줄 수 있습니다.

최대 풀 연결을 위한 Config

마지막으로, 구성을 통해 연결 풀 크기를 제어할 수 있습니다.

- max_pool_connections(int) - 연결 풀에 유지할 수 있는 최대 연결 수입니다. 값을 설정하지 않으면 기본값 10이 사용됩니다.

이 옵션은 재사용을 위해 풀링된 상태로 유지할 최대 HTTP 연결 수를 제어합니다. Session마다 다른 풀이 보관됩니다. 동일한 Session에서 구축된 클라이언트나 리소스에 대해 스레드가 10개 이상 발생할 것으로 예상되면 스레드가 풀링된 연결을 사용하는 다른 스레드에서 대기하지 않아도 되도록 이 값을 늘리는 것이 좋습니다.

```
import boto3
from botocore.config import Config

my_config = Config(
   max_pool_connections = 20
)

# Setup a single session holding up to 20 pooled connections
session = boto3.Session(my_config)
```

```
# Create up to 20 resources against that session for handing to threads
# Notice the single-threaded access to the Session and each Resource
resource1 = session.resource('dynamodb')
resource2 = session.resource('dynamodb')
# etc
```

## 오류 처리

AWS 서비스 예외가 Boto3에서 모두 정적으로 정의되어 있지는 않습니다. 이는 AWS 서비스의 오류 및 예외가 매우 다양하고 변경될 수 있기 때문입니다. Boto3는 모든 서비스 예외를 `ClientError`로 래핑하고 세부 정보를 구조화된 JSON으로 노출합니다. 예를 들어, 오류 응답은 다음과 같이 구성될 수 있습니다.

```
{
    'Error': {
        'Code': 'SomeServiceException',
        'Message': 'Details/context around the exception or error'
    },
    'ResponseMetadata': {
        'RequestId': '1234567890ABCDEF',
        'HostId': 'host ID data will appear here as a hash',
        'HTTPStatusCode': 400,
        'HTTPHeaders': {'header metadata key/values will appear here'},
        'RetryAttempts': 0
    }
}
```

다음 코드는 모든 `ClientError` 예외를 포착하고 `Error` 내의 `Code` 문자열 값을 검토하여 취해야 할 조치를 결정합니다.

```
import botocore
import boto3

dynamodb = boto3.client('dynamodb')

try:
    response = dynamodb.put_item(...)

except botocore.exceptions.ClientError as err:
    print('Error Code: {}'.format(err.response['Error']['Code']))
    print('Error Message: {}'.format(err.response['Error']['Message']))
```

```
    print('Http Code: {}'.format(err.response['ResponseMetadata']['HTTPStatusCode']))
    print('Request ID: {}'.format(err.response['ResponseMetadata']['RequestId']))

    if err.response['Error']['Code'] in ('ProvisionedThroughputExceededException',
 'ThrottlingException'):
        print("Received a throttle")
    elif err.response['Error']['Code'] == 'InternalServerError':
        print("Received a server error")
    else:
        raise err
```

전부는 아니지만 일부 예외 코드는 최상위 수준 클래스로 구체화되었습니다. 이를 직접 처리할 수도 있습니다. Client 인터페이스를 사용하는 경우 이러한 예외는 클라이언트에 동적으로 채워지며 다음과 같이 클라이언트 인스턴스를 사용하여 이러한 예외를 포착할 수 있습니다.

```
except ddb_client.exceptions.ProvisionedThroughputExceededException:
```

Resource 인터페이스를 사용할 때는 `.meta.client`를 사용하여 다음과 같이 리소스에서 기본 Client로 이동해야 예외에 액세스할 수 있습니다.

```
except ddb_resource.meta.client.exceptions.ProvisionedThroughputExceededException:
```

구체화된 예외 유형 목록을 검토하려면 목록을 동적으로 생성하면 됩니다.

```
ddb = boto3.client("dynamodb")
print([e for e in dir(ddb.exceptions) if e.endswith('Exception') or
 e.endswith('Error')])
```

조건식을 사용하여 쓰기 작업을 수행할 때 표현식이 실패할 경우 오류 응답에서 항목 값을 반환하도록 요청할 수 있습니다.

```
try:
    response = table.put_item(
        Item=item,
        ConditionExpression='attribute_not_exists(pk)',
        ReturnValuesOnConditionCheckFailure='ALL_OLD'
    )
except table.meta.client.exceptions.ConditionalCheckFailedException as e:
    print('Item already exists:', e.response['Item'])
```

오류 처리 및 예외에 대한 추가 정보:

- 오류 처리에 대한 boto3 가이드에는 오류 처리 기법에 대한 자세한 정보가 있습니다.
- 프로그래밍 오류에 대한 DynamoDB 개발자 안내서 섹션에는 발생할 수 있는 오류가 나열되어 있습니다.
- API 참조의 Common Errors 섹션
- 각 API 작업의 설명서에는 직접 호출로 인해 발생할 수 있는 오류가 나열되어 있습니다(예: BatchWriteItem).

## 로깅

boto3 라이브러리는 Python의 기본 제공 로깅 모듈과 통합되어 세션 중에 발생하는 일을 추적합니다. 로깅 수준을 제어하기 위해 로깅 모듈을 구성할 수 있습니다.

```
import logging

logging.basicConfig(level=logging.INFO)
```

이렇게 하면 루트 로거가 `INFO` 이상 수준의 메시지를 로깅하도록 구성됩니다. 이 수준보다 덜 심각한 로깅 메시지는 무시됩니다. 로깅 수준은 `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`입니다. 기본값은 `WARNING`입니다.

Boto3의 로거는 계층적입니다. 라이브러리는 각각 라이브러리의 다른 부분에 해당하는 몇 가지 다른 로거를 사용합니다. 각 로거의 동작을 개별적으로 제어할 수 있습니다.

- boto3: boto3 모듈의 기본 로거입니다.
- botocore: botocore 패키지의 메인 로거입니다.
- botocore.auth: 요청에 대한 AWS 서명 생성을 로깅하는 데 사용됩니다.
- botocore.credentials: 자격 증명 가져오기 및 새로 고침 프로세스를 로깅하는 데 사용됩니다.
- botocore.endpoint: 네트워크를 통해 전송되기 전에 요청 생성을 로깅하는 데 사용됩니다.
- botocore.hooks: 라이브러리에서 트리거되는 이벤트를 로깅하는 데 사용됩니다.
- botocore.loaders: AWS 서비스 모델의 일부가 로드될 때 로깅하는 데 사용됩니다.
- botocore.parsers: AWS 서비스 응답을 구문 분석하기 전에 로깅하는 데 사용됩니다.
- botocore.retryhandler: AWS 서비스 요청 재시도 처리를 로깅하는 데 사용됩니다(레거시 모드).

- botocore.retries.standard: AWS 서비스 요청 재시도 처리를 로깅하는 데 사용됩니다(표준 또는 적응형 모드).
- botocore.utils: 라이브러리의 기타 활동을 로깅하는 데 사용됩니다.
- botocore.waiter: 특정 상태에 도달할 때까지 AWS 서비스를 폴링하는 웨이터의 기능을 로깅하는 데 사용됩니다.

다른 라이브러리도 로깅합니다. 내부적으로 boto3는 HTTP 연결 처리를 위해 서드 파티 urllib3를 사용합니다. 지연 시간이 중요한 경우 urllib3가 언제 새 연결을 설정하거나 유휴 연결을 종료하는지 확인하여 풀을 제대로 활용하고 있는지 확인할 수 있습니다.

- urllib3.connectionpool: 연결 풀 처리 이벤트를 로깅하는 데 사용합니다.

다음 코드 스니펫은 엔드포인트 및 연결 풀 활동에 대한 DEBUG 로깅과 함께 대부분의 로깅을 INFO로 설정합니다.

```
import logging

logging.getLogger('boto3').setLevel(logging.INFO)
logging.getLogger('botocore').setLevel(logging.INFO)
logging.getLogger('botocore.endpoint').setLevel(logging.DEBUG)
logging.getLogger('urllib3.connectionpool').setLevel(logging.DEBUG)
```

## 이벤트 후크

Botocore는 실행의 여러 부분에서 이벤트를 내보냅니다. 이러한 이벤트에 대한 핸들러를 등록하여 이벤트가 발생할 때마다 핸들러가 직접 호출되도록 할 수 있습니다. 이렇게 하면 내부를 수정하지 않고도 botocore의 동작을 확장할 수 있습니다.

예를 들어 애플리케이션의 DynamoDB 테이블에서 PutItem 작업이 직접 호출될 때마다 추적하고 싶다고 가정해 보겠습니다. 관련 Session에서 PutItem 작업이 간접 호출될 때마다 'provide-client-params.dynamodb.PutItem' 이벤트를 등록하여 포착하고 로깅할 수 있습니다. 다음은 그 예입니다.

```
import boto3
import botocore
import logging

def log_put_params(params, **kwargs):
```

```
    if 'TableName' in params and 'Item' in params:
        logging.info(f"PutItem on table {params['TableName']}: {params['Item']}")

logging.basicConfig(level=logging.INFO)

session = boto3.Session()
event_system = session.events

# Register our interest in hooking in when the parameters are provided to PutItem
event_system.register('provide-client-params.dynamodb.PutItem', log_put_params)

# Now, every time you use this session to put an item in DynamoDB,
# it will log the table name and item data.
dynamodb = session.resource('dynamodb')
table = dynamodb.Table('YourTableName')
table.put_item(
    Item={
        'pk': '123',
        'sk': 'cart#123',
        'item_data': 'YourItemData',
        # ... more attributes ...
    }
)
```

핸들러 내에서 파라미터를 프로그래밍 방식으로 조작하여 동작을 변경할 수도 있습니다.

```
params['TableName'] = "NewTableName"
```

이벤트에 대한 자세한 내용은 이벤트에 대한 botocore 설명서 및 이벤트에 대한 boto3 설명서를 참조하세요.

## 페이지 매김 및 페이지네이터

쿼리 및 스캔과 같은 일부 요청의 경우 단일 요청에서 반환되는 데이터 크기가 제한되므로 후속 페이지를 가져오려면 반복적으로 요청해야 합니다.

`limit` 파라미터를 사용하여 각 페이지에 대해 읽을 항목의 최대 수를 제어할 수 있습니다. 예를 들어, 마지막 10개의 항목을 원하는 경우 `limit`을 사용하여 마지막 10개만 검색할 수 있습니다. 이 제한은 필터링을 적용하기 전에 테이블에서 읽어야 하는 분량이라는 것을 참고하세요. 필터링 후 정확히 10개를 원한다고 지정할 수 있는 방법은 없습니다. 필터링 전의 개수를 제어하고 실제로 10개 항목을 검색한 경우에만 클라이언트 측에서 확인할 수 있습니다. 제한과 관계없이 모든 응답의 최대 크기는 항상 1MB입니다.

응답에 LastEvaluatedKey가 포함된 경우 개수 또는 크기 제한에 도달하여 응답이 종료되었음을 나타냅니다. 이 키는 해당 응답에 대해 마지막으로 평가된 키입니다. 이 LastEvaluatedKey를 검색하고 후속 직접 호출에 ExclusiveStartKey로 전달하여 해당 시작 지점부터 다음 청크를 읽을 수 있습니다. 이것을 반환한 LastEvaluatedKey가 없으면 Query 또는 Scan과 일치하는 항목이 더 이상 없음을 의미합니다.

다음은 Resource 인터페이스를 사용하지만 Client 인터페이스의 패턴이 같은 간단한 예시로, 페이지당 최대 100개의 항목을 읽고 모든 항목을 읽을 때까지 반복합니다.

```
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('YourTableName')

query_params = {
    'KeyConditionExpression': Key('pk').eq('123') & Key('sk').gt(1000),
    'Limit': 100
}

while True:
    response = table.query(**query_params)

    # Process the items however you like
    for item in response['Items']:
        print(item)

    # No LastEvaluatedKey means no more items to retrieve
    if 'LastEvaluatedKey' not in response:
        break

    # If there are possibly more items, update the start key for the next page
    query_params['ExclusiveStartKey'] = response['LastEvaluatedKey']
```

편의를 위해 boto3에서는 페이지네이터를 사용하여 이 작업을 대신 수행할 수 있습니다. 하지만 이 기능은 Client 인터페이스에서만 작동합니다. 페이지네이터를 사용하도록 다시 작성된 코드는 다음과 같습니다.

```
import boto3

dynamodb = boto3.client('dynamodb')

paginator = dynamodb.get_paginator('query')
```

```
query_params = {
    'TableName': 'YourTableName',
    'KeyConditionExpression': 'pk = :pk_val AND sk > :sk_val',
    'ExpressionAttributeValues': {
        ':pk_val': {'S': '123'},
        ':sk_val': {'N': '1000'},
    },
    'Limit': 100
}

page_iterator = paginator.paginate(**query_params)

for page in page_iterator:
    # Process the items however you like
    for item in page['Items']:
        print(item)
```

자세한 내용은 페이지네이터 가이드 및 DynamoDB.Paginator.Query에 대한 API 참조를 확인하세요.

> **Note**
>
> 페이지네이터에는 `MaxItems`, `StartingToken`, `PageSize`라는 자체 구성 설정도 있습니다. DynamoDB로 페이지를 매기는 경우 이러한 설정을 무시해야 합니다.

## Waiters

웨이터는 작업이 완료될 때까지 기다렸다가 진행하는 기능을 제공합니다. 현재는 테이블이 생성되거나 삭제될 때까지 기다리는 것만 지원합니다. 백그라운드에서 웨이터 작업은 20초마다 최대 25회까지 사용자를 대신하여 확인을 수행합니다. 이 작업은 직접 할 수도 있지만, 자동화를 작성할 때는 웨이터를 사용하는 것이 좋습니다.

다음 코드는 특정 테이블이 생성될 때까지 기다리는 방법을 보여줍니다.

```
# Create a table, wait until it exists, and print its ARN
response = client.create_table(...)
waiter = client.get_waiter('table_exists')
waiter.wait(TableName='YourTableName')
print('Table created:', response['TableDescription']['TableArn']
```

자세한 내용은 웨이터 가이드 및 웨이터에 대한 참조를 확인하세요.

# JavaScript를 사용한 Amazon DynamoDB 프로그래밍

이 안내서는 JavaScript를 통해 Amazon DynamoDB를 사용하려는 프로그래머에게 지침을 제공합니다. AWS SDK for JavaScript, 사용 가능한 추상화 계층, 연결 구성, 오류 처리, 재시도 정책 정의, 연결 유지 관리 등에 대해 알아봅니다.

주제


- AWS SDK for JavaScript 정보
- AWS SDK for JavaScript V3 사용
- JavScript 설명서 액세스
- 추상화 계층
- 마샬 유틸리티 함수 사용
- 항목 읽기
- 조건부 쓰기
- 페이지 매김
- 구성 지정
- Waiters
- 오류 처리
- 로깅
- 고려 사항


## AWS SDK for JavaScript 정보

AWS SDK for JavaScript에서는 브라우저 스크립트 또는 Node.js를 사용하여 AWS 서비스에 액세스할 수 있습니다. 이 문서에서는 최신 버전의 SDK(V3)에 중점을 둡니다. AWS SDK for JavaScript V3는 GitHub에서 호스팅되는 오픈 소스 프로젝트로 AWS에서 유지 관리합니다. 문제 및 기능 요청은 공개되며 GitHub 리포지토리의 문제 페이지에서 액세스할 수 있습니다.

JavaScript V2는 V3와 비슷하지만 구문 차이가 있습니다. V3는 더 모듈화되어 더 작은 종속성을 더 쉽게 제공할 수 있으며 최고 수준의 TypeScript를 지원합니다. SDK는 최신 버전을 사용하는 것이 좋습니다.

## AWS SDK for JavaScript V3 사용

노드 패키지 관리자를 사용하여 Node.js 애플리케이션에 SDK를 추가할 수 있습니다. 아래 예는 DynamoDB 작업을 위한 가장 일반적인 SDK 패키지를 추가하는 방법을 보여줍니다.

- `npm install @aws-sdk/client-dynamodb`
- `npm install @aws-sdk/lib-dynamodb`
- `npm install @aws-sdk/util-dynamodb`

패키지를 설치하면 package.json 프로젝트 파일의 종속성 섹션에 대한 참조가 추가됩니다. 최신 ECMAScript 모듈 구문을 사용할 수 있는 옵션이 있습니다. 이 두 가지 접근 방식에 대한 자세한 내용은 고려 사항 섹션을 참조하세요.

## JavScript 설명서 액세스

다음 리소스로 JavaScript 설명서 사용을 시작하세요.

- 핵심 JavaScript 설명서를 보려면 개발자 안내서를 참조하세요. 설치 지침은 설정 섹션에 있습니다.
- API 참조 설명서에 액세스하여 사용 가능한 모든 클래스와 메서드를 살펴보세요.
- SDK for JavaScript는 DynamoDB 외에도 많은 AWS 서비스를 지원합니다. DynamoDB에 대한 구체적인 API 적용 범위를 찾으려면 다음 절차를 따르세요.

  1. 서비스에서 DynamoDB 및 라이브러리를 선택합니다. 이는 하위 수준 클라이언트를 문서화합니다.
  2. lib-dynamodb를 선택합니다. 이는 상위 수준 클라이언트를 문서화합니다. 두 클라이언트는 사용자가 사용하기 위해 선택할 수 있는 두 가지 추상화 계층을 나타냅니다. 추상화 계층에 대한 자세한 내용은 아래 섹션을 참조하세요.

## 추상화 계층

SDK for JavaScript V3에는 하위 수준 클라이언트(`DynamoDBClient`)와 상위 수준 클라이언트(`DynamoDBDocumentClient`)가 있습니다.

주제

- 하위 수준 클라이언트(DynamoDBClient)
- 상위 수준 클라이언트(DynamoDBDocumentClient)

## 하위 수준 클라이언트(**DynamoDBClient**)

하위 수준 클라이언트는 기본 유선 프로토콜에 대한 추가 추상화를 제공하지 않습니다. 통신의 모든 측면을 완벽하게 제어할 수 있지만 추상화가 없으므로 DynamoDB JSON 형식을 사용하여 항목 정의를 제공하는 등의 작업을 수행해야 합니다.

아래 예시에서 볼 수 있듯이 이 형식을 사용할 경우 데이터 유형을 명시적으로 지정해야 합니다. S는 문자열 값을 나타내고 N은 숫자 값을 나타냅니다. 회선의 숫자는 정밀도가 떨어지지 않도록 항상 숫자 유형 태그가 지정된 문자열로 전송됩니다. 하위 수준 API 직접 호출에는 `PutItemCommand` 및 `GetItemCommand`와 같은 이름 지정 패턴이 있습니다.

다음은 DynamoDB JSON을 사용하여 `Item`을 정의한 하위 수준 클라이언트를 사용하는 예시입니다.

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      "id": { S: "Product01" },
      "description": { S: "Hiking Boots" },
      "category": { S: "footwear" },
      "sku": { S: "hiking-sku-01" },
      "size": { N: "9" }
    }
  };

  try {
    const data = await client.send(new PutItemCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}
addProduct();
```

## 상위 수준 클라이언트(DynamoDBDocumentClient)

상위 수준 DynamoDB 문서 클라이언트는 데이터를 수동으로 마샬링할 필요가 없고 표준 JavaScript 객체를 사용하여 직접 읽고 쓸 수 있는 등 편리한 기능을 기본적으로 제공합니다. lib-dynamodb 설명서에는 장점 목록이 나와 있습니다.

DynamoDBDocumentClient를 인스턴스화하려면 하위 수준 DynamoDBClient를 구성한 다음 DynamoDBDocumentClient로 래핑하세요. 함수 명명 규칙은 두 패키지가 약간 다릅니다. 예를 들어, 하위 수준에서는 PutItemCommand를, 상위 수준에서는 PutCommand를 사용합니다. 이름이 서로 다르기 때문에 두 함수가 같은 컨텍스트에서 공존할 수 있으므로 동일한 스크립트에서 두 함수를 혼용할 수 있습니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const { DynamoDBDocumentClient, PutCommand } = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    },
  };

  try {
    const data = await docClient.send(new PutCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

addProduct();
```

GetItem, Query 또는 Scan 같은 API 작업을 사용하여 항목을 읽을 때 사용 패턴이 일관됩니다.

## 마샬 유틸리티 함수 사용

하위 수준 클라이언트를 사용하고 직접 데이터 유형을 마샬링하거나 언마셜링할 수 있습니다. 유틸리티 패키지인 util-dynamodb에는 JSON을 받아들이고 DynamoDB JSON을 생성하는 `marshall()` 유틸리티 함수와 그 반대의 작업을 수행하는 `unmarshall()` 함수가 있습니다. 다음 예시에서는 `marshall()` 직접 호출로 데이터 마샬링을 처리하는 하위 수준 클라이언트를 사용합니다.

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");
const { marshall } = require("@aws-sdk/util-dynamodb");

const client = new DynamoDBClient({});

async function addProduct() {
  const params = {
    TableName: "products",
    Item: marshall({
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    }),
  };

  try {
    const data = await client.send(new PutItemCommand(params));
  } catch (error) {
    console.error("Error:", error);
  }
}
addProduct();
```

## 항목 읽기

DynamoDB에서 단일 항목을 읽으려면 `GetItem` API 작업을 사용합니다. `PutItem` 명령과 마찬가지로 하위 수준 클라이언트 또는 상위 수준 Document 클라이언트를 사용할 수 있습니다. 아래 예시는 상위 수준 Document 클라이언트를 사용하여 항목을 가져오는 방법을 보여줍니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const { DynamoDBDocumentClient, GetCommand } = require("@aws-sdk/lib-dynamodb");
```

```
const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function getProduct() {
  const params = {
    TableName: "products",
    Key: {
      id: "Product01",
    },
  };

  try {
    const data = await docClient.send(new GetCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

getProduct();
```

Query API 작업을 사용하여 여러 항목을 읽을 수 있습니다. 하위 수준 클라이언트 또는 Document 클라이언트를 사용할 수 있습니다. 아래 예에서는 상위 수준 Document 클라이언트를 사용합니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  QueryCommand,
} = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function productSearch() {
  const params = {
    TableName: "products",
    IndexName: "GSI1",
    KeyConditionExpression: "#category = :category and begins_with(#sku, :sku)",
    ExpressionAttributeNames: {
      "#category": "category",
      "#sku": "sku",
```

```
    },
    ExpressionAttributeValues: {
      ":category": "footwear",
      ":sku": "hiking",
    },
  };

  try {
    const data = await docClient.send(new QueryCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

productSearch();
```

## 조건부 쓰기

DynamoDB 쓰기 작업은 쓰기가 진행되려면 true로 평가되어야 하는 논리적 조건식을 지정할 수 있습니다. 조건이 true로 평가되지 않으면 쓰기 작업에서 예외가 발생합니다. 조건 표현식은 항목이 이미 존재하는지 또는 항목의 속성이 특정 제약 조건과 일치하는지 확인할 수 있습니다.

`ConditionExpression = "version = :ver AND size(VideoClip) < :maxsize"`

조건식이 실패할 경우 `ReturnValuesOnConditionCheckFailure`를 사용하여 조건을 충족하지 못한 항목을 오류 응답에 포함하도록 요청하여 문제가 무엇인지 추론할 수 있습니다. 자세한 내용은 Handle conditional write errors in high concurrency scenarios with Amazon DynamoDB를 참조하세요.

```
try {
      const response = await client.send(new PutCommand({
          TableName: "YourTableName",
          Item: item,
          ConditionExpression: "attribute_not_exists(pk)",
          ReturnValuesOnConditionCheckFailure: "ALL_OLD"
      }));
  } catch (e) {
      if (e.name === 'ConditionalCheckFailedException') {
          console.log('Item already exists:', e.Item);
      } else {
          throw e;
      }
```

```
}
```

JavaScript SDK V3 사용의 다른 측면을 보여주는 추가 코드 예시는 JavaScript SDK V3 설명서 및 DynamoDB-SDK-Examples GitHub 리포지토리에서 확인할 수 있습니다.

## 페이지 매김

주제

- paginateScan 편의 메서드 사용

Scan 또는 Query와 같은 읽기 요청은 데이터세트의 여러 항목을 반환할 가능성이 큽니다. Limit 파라미터와 함께 Scan 또는 Query를 수행하는 경우 시스템에서 해당 수의 항목을 읽은 후 부분적인 응답이 전송되므로 추가 항목을 가져오려면 페이지를 매겨야 합니다.

시스템은 요청당 최대 1메가바이트의 데이터만 읽습니다. Filter 표현식을 포함하는 경우 시스템은 여전히 디스크에서 최대 1메가바이트의 데이터를 읽지만 해당 메가바이트에서 필터와 일치하는 항목을 반환합니다. 필터 작업 시 한 페이지에 0개의 항목이 반환되어도 검색이 모두 끝나려면 페이지 매김을 더 해야 할 수도 있습니다.

데이터 검색을 계속하려면 응답에서 LastEvaluatedKey를 찾아 후속 요청에서 ExclusiveStartKey 파라미터로 사용해야 합니다. 이는 다음 예에서 볼 수 있듯이 북마크 역할을 합니다.

> **Note**
>
> 샘플은 첫 번째 반복에서와 같이 null lastEvaluatedKey를 ExclusiveStartKey로 전달하는데 이 동작은 허용됩니다.

LastEvaluatedKey 사용 예시:

```
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScan() {
  let lastEvaluatedKey;
  let pageCount = 0;
```

```
  do {
    const params = {
      TableName: "products",
      ExclusiveStartKey: lastEvaluatedKey,
    };

    const response = await client.send(new ScanCommand(params));
    pageCount++;
    console.log(`Page ${pageCount}, Items:`, response.Items);
    lastEvaluatedKey = response.LastEvaluatedKey;
  } while (lastEvaluatedKey);
}

paginatedScan().catch((err) => {
  console.error(err);
});
```

## **paginateScan** 편의 메서드 사용

SDK는 이 작업을 사용자를 대신해서 수행하고 백그라운드에서 반복적인 요청을 수행하는 편의 메서드 paginateScan 및 paginateQuery를 제공합니다. 표준 Limit 파라미터를 사용하여 요청당 읽을 수 있는 최대 항목 수를 지정합니다.

```
const { DynamoDBClient, paginateScan } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScanUsingPaginator() {
  const params = {
    TableName: "products",
    Limit: 100
  };

  const paginator = paginateScan({client}, params);

  let pageCount = 0;

  for await (const page of paginator) {
    pageCount++;
    console.log(`Page ${pageCount}, Items:`, page.Items);
  }
}
```

```
paginatedScanUsingPaginator().catch((err) => {
  console.error(err);
});
```

> **Note**
>
> 테이블이 작지 않은 이상 전체 테이블 스캔을 정기적으로 수행하는 것은 권장되지 않는 액세스 패턴입니다.

## 구성 지정

주제

- 제한 시간을 위한 Config
- 연결 유지를 위한 Config
- 재시도를 위한 Config

DynamoDBClient를 설정할 때 구성 객체를 생성자에 전달하여 다양한 구성 재정의를 지정할 수 있습니다. 예를 들어, 직접 호출하는 컨텍스트나 사용할 엔드포인트 URL에 아직 알려지지 않은 경우 연결할 리전을 지정할 수 있습니다. 이는 DynamoDB 로컬 인스턴스를 개발 목적으로 대상으로 지정하려는 경우에 유용합니다.

```
const client = new DynamoDBClient({
  region: "eu-west-1",
  endpoint: "http://localhost:8000",
});
```

### 제한 시간을 위한 Config

DynamoDB는 클라이언트와 서버 간 통신을 위해 HTTPS를 사용합니다. NodeHttpHandler 객체를 제공하여 HTTP 계층의 일부 측면을 제어할 수 있습니다. 예를 들어, 키 제한 시간 값 connectionTimeout 및 requestTimeout을 조정할 수 있습니다. connectionTimeout은 클라이언트가 연결을 포기하기 전까지 연결을 시도하는 동안 기다리는 최대 시간(밀리초)입니다.

requestTimeout은 요청이 전송된 후 클라이언트가 응답을 기다리는 시간(밀리초)입니다. 둘 다 기본값은 0입니다. 즉, 제한 시간이 비활성화되어 있고 응답이 도착하지 않을 경우 클라이언트가 대기하

는 시간에는 제한이 없다는 뜻입니다. 네트워크 문제 발생 시 요청에 오류가 발생하여 새 요청을 시작할 수 있도록 제한 시간을 적절한 수준으로 설정해야 합니다. 예시:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";

const requestHandler = new NodeHttpHandler({
  connectionTimeout: 2000,
  requestTimeout: 2000,
});

const client = new DynamoDBClient({
  requestHandler
});
```

> **Note**
>
> 제공된 예시에서는 Smithy 가져오기를 사용합니다. Smithy는 오픈 소스이며 AWS가 유지 관리하며 서비스와 SDK를 정의하는 언어입니다.

제한 시간 값을 구성하는 것 외에도 최대 소켓 수를 설정하여 오리진당 동시 연결 수를 늘릴 수 있습니다. 개발자 안내서에는 `maxSockets` 파라미터 구성에 대한 세부 정보가 포함되어 있습니다.

### 연결 유지를 위한 Config

HTTPS를 사용하는 경우 첫 번째 요청은 보안 연결을 설정하기 위해 항상 주고받는 통신이 필요합니다. HTTP 연결 유지를 사용하면 후속 요청에서 이미 설정된 연결을 재사용할 수 있으므로 요청의 효율성을 높이고 지연 시간을 줄일 수 있습니다. JavaScript V3에서는 HTTP 연결 유지가 기본적으로 활성화되어 있습니다.

유휴 연결을 유지할 수 있는 기간에는 한도가 있습니다. 연결이 유휴 상태이지만 다음 요청에서 이미 설정된 연결을 사용하도록 하려면 주기적으로(예: 1분마다) 요청을 보내는 것을 고려하세요.

> **Note**
>
> 참고로 SDK의 이전 V2에서는 연결 유지가 기본적으로 꺼져 있었습니다. 즉, 각 연결은 사용 후 즉시 닫혔습니다. V2를 사용하는 경우 이 설정을 재정의할 수 있습니다.

## 재시도를 위한 Config

SDK에서 오류 응답을 수신하고 SDK의 판단에 따라 오류를 재개할 수 있는 경우(예: 제한 예외 또는 임시 서비스 예외) SDK는 다시 재시도합니다. 요청이 성공하는 데 시간이 더 오래 걸렸다는 점을 제외하면 호출자 입장에서는 이러한 상황이 눈에 띄지 않게 발생합니다.

SDK for JavaScript V3는 기본적으로 총 3번의 요청을 한 후 포기하고 직접 호출하는 컨텍스트로 오류를 전달합니다. 이러한 재시도 횟수와 빈도를 조정할 수 있습니다.

`DynamoDBClient` 생성자는 시도 횟수를 제한하는 `maxAttempts` 설정을 허용합니다. 아래 예시는 값을 기본값인 3에서 총 5로 올립니다. 0 또는 1로 설정하면 자동 재시도를 원하지 않으며 catch 블록 내에서 재개 가능한 오류를 직접 처리하고 싶다는 뜻입니다.

```
const client = new DynamoDBClient({
  maxAttempts: 5,
});
```

사용자 지정 재시도 전략을 사용하여 재시도 타이밍을 제어할 수도 있습니다. 이렇게 하려면 `util-retry` 유틸리티 패키지를 가져와서 현재 재시도 횟수를 기반으로 재시도 간 대기 시간을 계산하는 사용자 지정 백오프 함수를 만드세요.

아래 예시에서는 첫 번째 시도가 실패할 경우 지연 시간이 15, 30, 90, 360밀리초인 상태에서 최대 5회 시도하도록 되어 있습니다. 사용자 지정 백오프 함수인 `calculateRetryBackoff`는 재시도 횟수(첫 번째 재시도는 1로 시작)를 수락하여 지연을 계산하고 해당 요청을 기다리는 데 걸리는 시간(밀리초)을 반환합니다.

```
const { ConfiguredRetryStrategy } = require("@aws-sdk/util-retry");

const calculateRetryBackoff = (attempt) => {
  const backoffTimes = [15, 30, 90, 360];
  return backoffTimes[attempt - 1] || 0;
};

const client = new DynamoDBClient({
  retryStrategy: new ConfiguredRetryStrategy(
    5, // max attempts.
    calculateRetryBackoff // backoff function.
  ),
});
```

## Waiters

DynamoDB 클라이언트에는 테이블을 생성, 수정 또는 삭제할 때 테이블 수정이 완료될 때까지 코드가 진행되기를 기다리는 데 사용할 수 있는 두 가지 유용한 웨이터 함수가 포함되어 있습니다. 예를 들어 테이블을 배포하고 `waitUntilTableExists` 함수를 호출하면 테이블이 활성 상태가 될 때까지 코드가 차단됩니다. 웨이터는 20초마다 `describe-table`을 사용하여 DynamoDB 서비스를 내부적으로 폴링합니다.

```
import {waitUntilTableExists, waitUntilTableNotExists} from "@aws-sdk/client-dynamodb";

… <create table details>

const results = await waitUntilTableExists({client: client, maxWaitTime: 180},
 {TableName: "products"});
if (results.state == 'SUCCESS') {
  return results.reason.Table
}
console.error(`${results.state} ${results.reason}`);
```

`waitUntilTableExists` 기능은 테이블 상태를 활성으로 표시하는 `describe-table` 명령을 수행할 수 있는 경우에만 제어권을 반환합니다. 이렇게 하면 `waitUntilTableExists`를 사용하여 테이블 생성뿐만 아니라 GSI 인덱스 추가와 같은 수정이 완료될 때까지 기다릴 수 있습니다. 이때 테이블이 활성 상태로 돌아가려면 적용하는 데 시간이 다소 걸릴 수 있습니다.

## 오류 처리

여기의 초기 예시에서는 모든 오류를 광범위하게 찾아냈습니다. 하지만 실제 상황에서는 다양한 오류 유형을 구분하고 보다 정확한 오류 처리를 구현하는 것이 중요합니다.

DynamoDB 오류 응답에는 오류 이름을 비롯한 메타데이터가 포함됩니다. 오류를 발견한 다음 오류 조건에 해당할 수 있는 문자열 이름을 비교하여 처리 방법을 결정할 수 있습니다. 서버 측 오류의 경우 `@aws-sdk/client-dynamodb` 패키지에서 내보낸 오류 유형과 함께 `instanceof` 연산자를 활용하여 오류 처리를 효율적으로 관리할 수 있습니다.

이러한 오류는 모든 재시도를 모두 완료한 후에만 발생한다는 점에 유의해야 합니다. 오류가 재시도되어 결국 성공적인 직접 호출이 발생하는 경우 코드의 관점에서 보면 오류가 나타나지 않고 지연 시간이 약간 길어질 뿐입니다. 재시도는 Amazon CloudWatch 차트에 제한 또는 오류 요청과 같은 실패한 요청으로 표시됩니다. 클라이언트가 최대 재시도 횟수에 도달하면 이를 포기하고 예외가 발생합니다. 이는 클라이언트가 재시도하지 않겠다고 말하는 방식입니다.

다음은 오류를 포착하고 반환된 오류 유형에 따라 조치를 취하는 스니펫입니다.

```
import {
  ResourceNotFoundException
  ProvisionedThroughputExceededException,
  DynamoDBServiceException,
} from "@aws-sdk/client-dynamodb";

try {
  await client.send(someCommand);
} catch (e) {
    if (e instanceof ResourceNotFoundException) {
      // Handle ResourceNotFoundException
    } else if (e instanceof ProvisionedThroughputExceededException) {
      // Handle ProvisionedThroughputExceededException
    } else if (e instanceof DynamoDBServiceException) {
      // Handle DynamoDBServiceException
    } else {
      // Other errors such as those from the SDK
      if (e.name === "TimeoutError") {
        // Handle SDK TimeoutError.
      } else {
        // Handle other errors.
      }
    }
}
```

DynamoDB 개발자 안내서의 일반적인 오류 문자열은 the section called “오류 처리” 섹션을 참조하세요. 특정 API 직접 호출에서 발생할 수 있는 정확한 오류는 해당 API 직접 호출에 대한 설명서(예: Query API 문서)에서 찾을 수 있습니다.

오류의 메타데이터에는 오류에 따라 추가 속성이 포함됩니다. `TimeoutError`의 경우 메타데이터에는 아래와 같이 시도한 횟수와 `totalRetryDelay`가 포함됩니다.

```
{
  "name": "TimeoutError",
  "$metadata": {
    "attempts": 3,
    "totalRetryDelay": 199
  }
}
```

자체 재시도 정책을 관리하는 경우 제한과 오류를 구분하는 것이 좋습니다.

- 제한( `ProvisionedThroughputExceededException` 또는 `ThrottlingException`으로 표시됨)은 정상 서비스에서 DynamoDB 테이블 또는 파티션의 읽기 또는 쓰기 용량을 초과했음을 알리는 것입니다. 밀리초가 지날 때마다 약간 더 많은 읽기 또는 쓰기 용량을 사용할 수 있게 되므로 빠르게(예: 50ms마다) 재시도하여 새로 확보된 용량에 액세스하려고 시도할 수 있습니다.

  제한을 사용하면 지수 백오프가 특별히 필요하지 않습니다. 제한은 DynamoDB가 반환하기에 가볍고 요청당 요금이 부과되지 않기 때문입니다. 지수 백오프는 이미 가장 오래 기다린 클라이언트 스레드에 더 긴 지연을 할당하여 통계적으로 p50과 p99를 밖으로 확장합니다.
- 오류(특히 `InternalServerError` 또는 `ServiceUnavailable`로 표시됨)는 서비스에 일시적인 문제가 있음을 나타냅니다. 전체 테이블에 문제가 있을 수도 있고, 읽거나 쓰고 있는 파티션에만 문제가 있을 수도 있습니다. 오류가 발생하면 재시도 전에 더 오래(예: 250ms 또는 500ms) 중단하고 지터를 사용하여 재시도에 시차를 줄 수 있습니다.

## 로깅

로깅을 켜면 SDK가 수행하는 작업에 대한 자세한 내용을 확인할 수 있습니다. 아래 예시와 같이 `DynamoDBClient`에서 파라미터를 설정할 수 있습니다. 콘솔에 더 많은 로그 정보가 표시되며 여기에는 상태 코드 및 사용된 용량과 같은 메타데이터가 포함됩니다. 터미널 창에서 로컬로 코드를 실행하면 로그가 해당 창에 나타납니다. AWS Lambda에서 코드를 실행하고 Amazon CloudWatch 로그를 설정하면 콘솔 출력이 해당 로그에 기록됩니다.

```
const client = new DynamoDBClient({
  logger: console
});
```

또한 내부 SDK 활동에 연결하여 특정 이벤트가 발생할 때 사용자 지정 로깅을 수행할 수 있습니다. 아래 예시는 클라이언트의 `middlewareStack`을 사용하여 SDK에서 전송되는 각 요청을 가로채고 요청이 발생하는 대로 로깅합니다.

```
const client = new DynamoDBClient({});

client.middlewareStack.add(
  (next) => async (args) => {
    console.log("Sending request from AWS SDK", { request: args.request });
    return next(args);
  },
  {
```

```
    step: "build",
    name: "log-ddb-calls",
  }
);
```

`MiddlewareStack`은 SDK 동작을 관찰하고 제어할 수 있는 강력한 후크를 제공합니다. 자세한 내용은 Introducing Middleware Stack in Modular AWS SDK for JavaScript 블로그를 참조하세요.

## 고려 사항

프로젝트에 AWS SDK for JavaScript를 구현할 때 다음과 같이 추가로 고려해야 할 요소가 있습니다.

모듈 시스템

SDK는 CommonJS와 ES(ECMAScript)라는 두 가지 모듈 시스템을 지원합니다. CommonJS는 `require` 함수를 사용하고 ES는 `import` 키워드를 사용합니다.

1. Common JS – `const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");`
2. ES(ECMAScript) – `import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";`

프로젝트 유형에 따라 사용할 모듈 시스템이 결정되며 package.json 파일의 유형 섹션에 명시되어 있습니다. 기본값은 CommonJS입니다. ES 프로젝트를 나타내려면 `"type": "module"`을 사용합니다. CommonJS 패키지 형식을 사용하는 기존 Node.JS 프로젝트가 있는 경우에도 함수 파일 이름을 .mjs 확장자로 지정하여 최신 SDK V3 Import 구문으로 함수를 추가할 수 있습니다. 이렇게 하면 코드 파일을 ES(ECMAScript)로 처리할 수 있습니다.

비동기식 운영

콜백을 사용하고 DynamoDB 작업의 결과를 처리하도록 약속하는 많은 코드 샘플을 보게 될 것입니다. 최신 JavaScript를 사용하면 이러한 복잡성이 더 이상 필요하지 않으며 개발자는 비동기식 작업에 더 간결하고 읽기 쉬운 async/await 구문을 활용할 수 있습니다.

웹 브라우저 런타임

React 또는 React Native로 빌드하는 웹 및 모바일 개발자는 프로젝트에서 SDK for JavaScript를 사용할 수 있습니다. SDK의 초기 V2에서는 웹 개발자가 https://sdk.amazonaws.com/js/에서 호스팅되는 SDK 이미지를 참조하여 전체 SDK를 브라우저에 로드해야 했습니다.

V3를 사용하면 SDK 설명서의 Getting started in a browser script 섹션에 설명된 대로 Webpack을 사용하여 필수 V3 클라이언트 모듈과 모든 필수 JavaScript 함수를 단일 JavaScript 파일로 번들링하고 이를 HTML 페이지의 <head>에 있는 스크립트 태그에 추가할 수 있습니다

DAX 데이터 영역 작업

Amazon DynamoDB Streams Accelerator(DAX) 데이터 플레인 작업은 SDK for JavaScript V3에서 지원되지 않습니다.

# AWS SDK for Java 2.x를 사용한 DynamoDB 프로그래밍

이 프로그래밍 안내서는 Java를 통해 Amazon DynamoDB를 사용하려는 프로그래머에게 지침을 제공합니다. 이 안내서는 추상화 계층, 구성 관리, 오류 처리, 재시도 정책 제어, 연결 유지 관리와 같은 다양한 개념을 다룹니다.

주제


- AWS SDK for Java 2.x 소개
- AWS SDK for Java 2.x 시작하기
- AWS SDK for Java 2.x 문서 검토
- 지원되는 인터페이스
- 추가 코드 예시
- 동기식 및 비동기식 프로그래밍
- HTTP 클라이언트
- HTTP 클라이언트 구성
- 오류 처리
- AWS 요청 ID
- 로깅
- 페이지 매김
- 데이터 클래스 주석


## AWS SDK for Java 2.x 소개

공식 AWS SDK for Java를 사용하여 Java에서 DynamoDB에 액세스할 수 있습니다. SDK for Java에는 1.x와 2.x의 두 가지 버전이 있습니다. 1.x의 경우 2024년 1월 12일에 지원 종료가 발표되었습니다. 2024년 7월 31일에 유지 관리 모드로 전환될 계획이며, 2025년 12월 31일에 지원이 종료될 예정입니다. 새 개발의 경우 2018년에 처음 릴리스된 2.x를 사용하는 것이 좋습니다. 이 안내서는 2.x만을 대상으로 하며 SDK에서 DynamoDB와 관련된 부분에만 초점을 맞춥니다.

AWS SDK에 대한 유지 관리 및 지원에 대한 자세한 내용은 AWS SDK 및 도구 참조 안내서의 AWS SDK and Tools maintenance policy 및 AWS SDKs and Tools version support matrix를 참조하세요.

AWS SDK for Java 2.x에서 1.x 코드 베이스 상당수를 다시 작성했습니다. SDK for Java 2.x는 Java 8에 도입된 비차단 I/O와 같은 최신 Java 기능을 지원합니다. 또한, SDK for Java 2.x는 플러그형 HTTP 클라이언트 구현에 대한 지원을 추가하여 보다 많은 네트워크 연결 유연성과 구성 옵션을 제공합니다.

SDK for Java 1.x와 비교할 때 SDK for Java 2.x의 눈에 띄는 변화는 새로운 패키지 이름 사용입니다. Java 1.x SDK는 `com.amazonaws` 패키지 이름을 사용하는 반면 Java 2.x SDK는 `software.amazon.awssdk` 패키지 이름을 사용합니다. 마찬가지로 Java 1.x SDK용 Maven 아티팩트는 `com.amazonaws` `groupId`를 사용하는 반면, Java 2.x SDK 아티팩트는 `software.amazon.awssdk` `groupId`를 사용합니다.

> **Important**
>
> AWS SDK for Java 1.x에는 이름이 `com.amazonaws.dynamodbv2`인 DynamoDB 패키지가 있습니다. 패키지 이름의 'v2'는 해당 패키지가 Java 2(J2SE) 용이라는 의미가 아닙니다. 'v2'는 패키지가 하위 수준 API의 원래 버전 대신 DynamoDB 하위 수준 API의 두 번째 버전을 지원함을 나타냅니다.

### Java 버전 지원

AWS SDK for Java 2.x는 장기 지원(LTS) Java 릴리스를 모두 지원합니다.

## AWS SDK for Java 2.x 시작하기

다음 자습서에서는 Apache Maven을 사용하여 SDK for Java 2.x에 대한 종속성을 정의하는 방법을 보여줍니다. 또한 이 자습서에서는 사용 가능한 DynamoDB 테이블을 나열하기 위해 DynamoDB에 연결하는 코드를 작성하는 방법도 보여줍니다. 이 안내서의 자습서는 AWS SDK for Java 2.x 개발자 안내서에 있는 Get started with the AWS SDK for Java 2.x 자습서를 기반으로 합니다. Amazon S3 대신 DynamoDB를 직접 호출하도록 이 자습서를 편집했습니다.

이 자습서를 완료하려면 다음이 필요합니다.

- 1단계: 튜토리얼 설정
- 2단계: 프로젝트 생성
- 3단계: 코드 작성
- 4단계: 애플리케이션 빌드 및 실행

## 1단계: 튜토리얼 설정

이 튜토리얼을 시작하기 전에 다음이 필요합니다.

- DynamoDB에 액세스할 수 있는 권한
- AWS 액세스 포털을 사용하여 AWS 서비스에 대한 SSO(Single Sign-On) 액세스로 구성된 Java 개발 환경

이 자습서를 설정하려면 AWS SDK for Java 2.x 개발자 안내서의 설정 개요에 있는 지침을 따르세요. Java SDK에 대한 SSO(Single Sign-On) 액세스로 개발 환경을 구성하고 활성 AWS 액세스 포털 세션이 있는 경우 이 자습서의 2단계로 계속 진행합니다.

## 2단계: 프로젝트 생성

이 자습서의 프로젝트를 생성하려면 프로젝트 구성 방법에 대한 입력을 요청하는 Maven 명령을 실행합니다. 모든 입력이 입력되고 확인되면 Maven은 `pom.xml` 파일을 생성하여 프로젝트 빌드를 완료하고 스텁 Java 파일을 생성합니다.

1. 터미널 또는 명령 프롬프트 창을 열고 원하는 디렉터리 (예: `Desktop` 또는 `Home` 폴더)로 이동합니다.
2. 터미널에서 다음 명령을 입력하고 Enter 키를 누릅니다.

```
mvn archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-app-quickstart \
   -DarchetypeVersion=2.22.0
```

3. 각 프롬프트의 두 번째 열에 나열된 값을 입력합니다.

| 프롬프트 | 입력할 값 |
| --- | --- |
| `Define value for property 'service':` | `dynamodb` |
| `Define value for property 'httpClient' :` | `apache-client` |
| `Define value for property 'nativeImage' :` | `false` |

| 프롬프트 | 입력할 값 |
| --- | --- |
| Define value for property 'credentialProvider' | identity-center |
| Define value for property 'groupId': | org.example |
| Define value for property 'artifactId': | getstarted |
| Define value for property 'version' 1.0-SNAPSHOT: | *<Enter>* |
| Define value for property 'package' org.example: | *<Enter>* |

4. 마지막 값을 입력하면 Maven에서 선택한 항목을 나열합니다. 확인하려면 Y를 입력합니다. 아니면 N을 입력한 다음 선택 사항을 다시 입력합니다.

Maven은 입력한 artifactId 값을 기반으로 이름이 getstarted로 지정된 프로젝트 폴더를 만듭니다. getstarted 폴더 안에서 검토할 수 있는 README.md라는 이름의 파일, pom.xml 파일, src 디렉터리를 찾습니다.

Maven은 다음과 같은 디렉터리 트리를 만듭니다.

```
getstarted
 ### README.md
 ### pom.xml
 ### src
     ### main
     #   ### java
     #   #   ### org
     #   #       ### example
     #   #           ### App.java
     #   #           ### DependencyFactory.java
     #   #           ### Handler.java
     #   ### resources
     #       ### simplelogger.properties
     ### test
         ### java
```

```
            ### org
                ### example
                    ### HandlerTest.java

 10 directories, 7 files
```

다음은 pom.xml 프로젝트 파일의 콘텐츠를 보여줍니다.

**pom.xml**

dependencyManagement 섹션은 AWS SDK for Java 2.x에 대한 종속성을 포함하며 dependencies 섹션에는 DynamoDB에 대한 종속성이 있습니다. 이러한 종속성을 지정하면 Maven이 관련 .jar 파일을 Java 클래스 경로에 포함하도록 강제합니다. 기본적으로 AWS SDK에는 모든 AWS 서비스에 대한 클래스가 모두 포함되어 있지는 않습니다. DynamoDB의 경우 하위 수준 인터페이스를 사용하면 dynamodb 아티팩트에 대한 종속성이 있어야 합니다. 또는 상위 수준 인터페이스를 사용하는 경우 dynamodb-enhanced 아티팩트에 종속적이어야 합니다 관련 종속성을 포함하지 않으면 코드가 컴파일되지 않습니다. 프로젝트는 maven.compiler.source 및 maven.compiler.target 속성의 1.8 값 때문에 Java 1.8을 사용합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <groupId>org.example</groupId>
     <artifactId>getstarted</artifactId>
     <version>1.0-SNAPSHOT</version>
     <packaging>jar</packaging>
     <properties>
         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
         <maven.compiler.source>1.8</maven.compiler.source>
         <maven.compiler.target>1.8</maven.compiler.target>
         <maven.shade.plugin.version>3.2.1</maven.shade.plugin.version>
         <maven.compiler.plugin.version>3.6.1</maven.compiler.plugin.version>
         <exec-maven-plugin.version>1.6.0</exec-maven-plugin.version>
         <aws.java.sdk.version>2.22.0</aws.java.sdk.version> <-------- SDK version
 picked up from archetype version.
         <slf4j.version>1.7.28</slf4j.version>
         <junit5.version>5.8.1</junit5.version>
     </properties>
```

```
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>bom</artifactId>
                <version>${aws.java.sdk.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>dynamodb</artifactId>  <-------- DynamoDB dependency
            <exclusions>
                <exclusion>
                    <groupId>software.amazon.awssdk</groupId>
                    <artifactId>netty-nio-client</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>software.amazon.awssdk</groupId>
                    <artifactId>apache-client</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>sso</artifactId> <-------- Required for identity center
authentication.
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>ssooidc</artifactId> <-------- Required for identity center
authentication.
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>apache-client</artifactId> <-------- HTTP client specified.
            <exclusions>
```

```
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Needed to adapt Apache Commons Logging used by Apache HTTP Client to
Slf4j to avoid
        ClassNotFoundException: org.apache.commons.logging.impl.LogFactoryImpl during
runtime -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Test Dependencies -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>${junit5.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>${maven.compiler.plugin.version}</version>
```

```
            </plugin>
        </plugins>
    </build>

</project>
```

## 3단계: 코드 작성

다음 코드는 Maven이 생성한 App 클래스를 보여줍니다. main 메서드는 Handler 클래스의 인스턴스를 만든 다음 해당 sendRequest 메서드를 호출하는 애플리케이션의 진입점입니다.

**App 클래스**

```
package org.example;
 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;

 public class App {
     private static final Logger logger = LoggerFactory.getLogger(App.class);

     public static void main(String... args) {
         logger.info("Application starts");

         Handler handler = new Handler();
         handler.sendRequest();

         logger.info("Application ends");
     }
 }
```

Maven에서 만든 DependencyFactory 클래스에는 DynamoDbClient 인스턴스를 빌드하고 반환하는 dynamoDbClient 팩토리 메서드가 포함되어 있습니다. DynamoDbClient 인스턴스는 Apache 기반 HTTP 클라이언트의 인스턴스를 사용합니다. 이는 Maven에서 사용할 HTTP 클라이언트를 묻는 메시지가 표시될 때 사용자가 apache-client를 지정했기 때문입니다.

다음 코드는 DependencyFactory 클래스를 보여줍니다.

DependencyFactory 클래스

```
package org.example;
```

```
import software.amazon.awssdk.http.apache.ApacheHttpClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

/**
 * The module containing all dependencies required by the {@link Handler}.
 */
public class DependencyFactory {

    private DependencyFactory() {}

    /**
     * @return an instance of DynamoDbClient
     */
    public static DynamoDbClient dynamoDbClient() {
        return DynamoDbClient.builder()
                        .httpClientBuilder(ApacheHttpClient.builder())
                        .build();
    }
}
```

Handler 클래스에는 프로그램의 기본 로직이 들어 있습니다. App 클래스에서 Handler 인스턴스가 생성되면 DependencyFactory는 DynamoDbClient 서비스 클라이언트를 제공합니다. 코드는 DynamoDbClient 인스턴스를 사용하여 DynamoDB를 직접 호출합니다.

Maven은 *TODO* 주석과 함께 다음과 같은 Handler 클래스를 생성합니다. 자습서의 다음 단계에서는 *TODO* 주석을 코드로 대체합니다.

Maven에서 생성한 **Handler** 클래스

```
package org.example;

import software.amazon.awssdk.services.dynamodb.DynamoDbClient;


public class Handler {
    private final DynamoDbClient dynamoDbClient;

    public Handler() {
        dynamoDbClient = DependencyFactory.dynamoDbClient();
    }

    public void sendRequest() {
        // TODO: invoking the API calls using dynamoDbClient.
```

```
    }
}
```

로직을 채우려면 Handler 클래스의 전체 내용을 다음 코드로 바꾸세요. sendRequest 메서드가 채워지고 필요한 임포트가 추가됩니다.

**Handler 클래스 구현됨**

다음 코드는 DynamoDbClient 인스턴스를 사용하여 기존 테이블 목록을 검색합니다. 지정된 계정 및 AWS 리전에 대한 테이블이 있는 경우 코드는 Logger 인스턴스를 사용하여 이러한 테이블의 이름을 로깅합니다.

```
package org.example;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;


public class Handler {
    private final DynamoDbClient dynamoDbClient;

    public Handler() {
        dynamoDbClient = DependencyFactory.dynamoDbClient();
    }

    public void sendRequest() {
        Logger logger = LoggerFactory.getLogger(Handler.class);

        logger.info("calling the DynamoDB API to get a list of existing tables");
        ListTablesResponse response = dynamoDbClient.listTables();

        if (!response.hasTableNames()) {
            logger.info("No existing tables found for the configured account &
region");
        } else {
            response.tableNames().forEach(tableName -> logger.info("Table: " +
tableName));
        }
    }
}
```

### 4단계: 애플리케이션 빌드 및 실행

프로젝트가 생성되고 전체 `Handler` 클래스가 포함된 후 애플리케이션을 빌드하고 실행합니다.

1. AWS IAM Identity Center 세션이 활성화되어 있는지 확인합니다. 확인하려면 AWS Command Line Interface(AWS CLI) 명령 `aws sts get-caller-identity`를 실행하고 응답을 점검하세요. 활성 세션이 없는 경우 AWS CLI를 사용하여 로그인에서 지침을 확인하세요.
2. 터미널 또는 명령 프롬프트 창을 열고 프로젝트 디렉토리 `getstarted`로 이동합니다.
3. 프로젝트를 빌드하려면 다음 명령을 실행합니다.

```
mvn clean package
```

4. 애플리케이션을 실행하려면 다음 명령을 사용합니다.

```
mvn exec:java -Dexec.mainClass="org.example.App"
```

파일을 확인한 후 객체를 삭제한 다음 버킷을 삭제합니다.

Success

Maven 프로젝트가 오류 없이 빌드되고 실행되었다면 축하합니다. SDK for Java 2.x를 사용한 첫 Java 애플리케이션 빌드에 성공했습니다.

정리

이 자습서를 진행하는 동안 만든 리소스를 정리하려면 `getstarted` 프로젝트 폴더를 삭제합니다.

## AWS SDK for Java 2.x 문서 검토

AWS SDK for Java 2.x 개발자 안내서에서는 전 AWS 서비스에 걸쳐 SDK의 모든 측면을 전체적으로 다룹니다. 다음 주제를 검토하는 것이 좋습니다.

- Migrate from version 1.x to 2.x - 1.x와 2.x의 차이점에 대한 자세한 설명이 포함되어 있습니다. 이 주제에는 두 주요 버전을 나란히 사용하는 방법에 대한 지침도 포함되어 있습니다.
- DynamoDB guide for Java 2.x SDK - 테이블 생성, 항목 조작, 항목 검색 등 기본적인 DynamoDB 작업을 수행하는 방법을 보여줍니다. 이 예에서는 하위 수준 인터페이스를 사용합니다. Java에는 지원되는 인터페이스 섹션에 설명된 대로 여러 인터페이스가 있습니다.

> **Tip**
>
> 이러한 주제를 검토한 후 AWS SDK for Java 2.x API 참조를 북마크하세요. 모든 AWS 서비스를 다루며 기본 API 참조로 사용하는 것이 좋습니다.

## 지원되는 인터페이스

AWS SDK for Java 2.x는 원하는 추상화 수준에 따라 다음 인터페이스를 지원합니다.

이 섹션의 주제

- 하위 수준 인터페이스
- 상위 수준 인터페이스
- 문서 인터페이스
- Query 예제와 인터페이스 비교

### 하위 수준 인터페이스

하위 수준 인터페이스는 기본 서비스 API에 대한 일대일 매핑을 제공합니다. 이 인터페이스를 통해 모든 DynamoDB API를 사용할 수 있습니다. 즉, 하위 수준 인터페이스가 완전한 기능을 제공할 수 있지만 사용하기가 더 복잡한 경우가 많습니다. 예를 들어, `.s()` 함수를 사용하여 문자열을 저장하고 `.n()` 함수를 사용하여 숫자를 저장합니다. 다음 PutItem 예시는 하위 수준 인터페이스를 사용하여 항목을 삽입합니다.

```
import org.slf4j.*;
import software.amazon.awssdk.http.crt.AwsCrtHttpClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;

import java.util.Map;

public class PutItem {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT = DynamoDbClient.create();
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
```

```
        PutItemResponse response = DYNAMODB_CLIENT.putItem(PutItemRequest.builder()
                .item(Map.of(
                        "pk", AttributeValue.builder().s("123").build(),
                        "sk", AttributeValue.builder().s("cart#123").build(),
                        "item_data",
 AttributeValue.builder().s("YourItemData").build(),
                        "inventory", AttributeValue.builder().n("500").build()
                        // ... more attributes ...
                ))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .tableName("YourTableName")
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

## 상위 수준 인터페이스

AWS SDK for Java 2.x의 상위 수준 인터페이스를 DynamoDB 향상된 클라이언트라고 합니다. 이 인터페이스는 보다 관용적인 코드 작성 경험을 제공합니다.

향상된 클라이언트는 클라이언트 측 데이터 클래스와 해당 데이터를 저장하도록 설계된 DynamoDB 테이블 간에 매핑하는 방법을 제공합니다. 코드에서 테이블과 해당 모델 클래스 간의 관계를 정의합니다. 그러면 SDK를 사용하여 데이터 유형 조작을 관리할 수 있습니다. 향상된 클라이언트에 대한 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 DynamoDB enhanced client API를 참조하세요.

다음 PutItem 예시에서는 상위 수준 인터페이스를 사용합니다. 이 예제에서는 `YourItem`이라는 `DynamoDbBean`이 `TableSchema`를 만들어 `putItem()` 직접 호출의 입력으로 바로 사용할 수 있도록 합니다.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName", TableSchema.fromBean(YourItem.class));
```

```
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourItem> response =
DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourItem.class)
                .item(new YourItem("123", "cart#123", "YourItemData", 500))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }

    @DynamoDbBean
    public static class YourItem {

        public YourItem() {}

        public YourItem(String pk, String sk, String itemData, int inventory) {
            this.pk = pk;
            this.sk = sk;
            this.itemData = itemData;
            this.inventory = inventory;
        }

        private String pk;
        private String sk;
        private String itemData;

        private int inventory;

        @DynamoDbPartitionKey
        public void setPk(String pk) {
            this.pk = pk;
        }

        public String getPk() {
            return pk;
        }

        @DynamoDbSortKey
        public void setSk(String sk) {
            this.sk = sk;
        }
```

```
        public String getSk() {
            return sk;
        }

        public void setItemData(String itemData) {
            this.itemData = itemData;
        }

        public String getItemData() {
            return itemData;
        }

        public void setInventory(int inventory) {
            this.inventory = inventory;
        }

        public int getInventory() {
            return inventory;
        }
    }
}
```

AWS SDK for Java 1.x에는 자체 상위 수준 인터페이스가 있으며, 이 인터페이스는 주로 기본 클래스 `DynamoDBMapper`에서 참조합니다. AWS SDK for Java 2.x는 `software.amazon.awssdk.enhanced.dynamodb`라는 별도의 패키지(및 Maven 아티팩트)에 게시됩니다. Java 2.x SDK는 주로 기본 클래스 `DynamoDbEnhancedClient`에서 참조합니다.

변경할 수 없는 데이터 클래스를 사용하는 상위 수준 인터페이스

DynamoDB 향상된 클라이언트 API의 매핑 기능은 변경 불가능한 데이터 클래스와도 함께 작동합니다. 불변 클래스에는 접근자만 포함되며 SDK가 클래스의 인스턴스를 생성하는 데 사용하는 빌더 클래스가 필요합니다. Java의 불변성은 개발자가 부작용이 없는 클래스를 만드는 데 사용할 수 있는 일반적으로 사용되는 스타일입니다. 복잡한 멀티스레드 애플리케이션에서는 이러한 클래스의 동작을 더 잘 예측할 수 있습니다. 변경 불가능한 클래스는 High-level interface example에 나온 대로 `@DynamoDbBean` 주석을 사용하는 대신 빌더 클래스를 입력으로 사용하는 `@DynamoDbImmutable` 주석을 사용합니다.

다음 예시에서는 빌더 클래스 `DynamoDbEnhancedClientImmutablePutItem`을 입력으로 사용하여 테이블 스키마를 생성합니다. 그런 다음 예시는 PutItem API 직접 호출을 위한 입력으로 해당 스키마를 제공합니다.

```
import org.slf4j.*;
```

```
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutablePutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutablePutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourImmutableItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableItem.class)
                .item(YourImmutableItem.builder()
                                    .pk("123")
                                    .sk("cart#123")
                                    .itemData("YourItemData")
                                    .inventory(500)
                                    .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

다음 예시에서는 변경 불가능한 데이터 클래스를 보여줍니다.

```
@DynamoDbImmutable(builder = YourImmutableItem.YourImmutableItemBuilder.class)
class YourImmutableItem {
    private final String pk;
    private final String sk;
    private final String itemData;
    private final int inventory;
    public YourImmutableItem(YourImmutableItemBuilder builder) {
        this.pk = builder.pk;
        this.sk = builder.sk;
        this.itemData = builder.itemData;
        this.inventory = builder.inventory;
    }
```

```
    public static YourImmutableItemBuilder builder() { return new
 YourImmutableItemBuilder(); }

    @DynamoDbPartitionKey
    public String getPk() {
        return pk;
    }

    @DynamoDbSortKey
    public String getSk() {
        return sk;
    }

    public String getItemData() {
        return itemData;
    }

    public int getInventory() {
        return inventory;
    }

    static final class YourImmutableItemBuilder {
        private String pk;
        private String sk;
        private String itemData;
        private int inventory;

        private YourImmutableItemBuilder() {}

        public YourImmutableItemBuilder pk(String pk) { this.pk = pk; return this; }
        public YourImmutableItemBuilder sk(String sk) { this.sk = sk; return this; }
        public YourImmutableItemBuilder itemData(String itemData) { this.itemData =
 itemData; return this; }
        public YourImmutableItemBuilder inventory(int inventory) { this.inventory =
 inventory; return this; }

        public YourImmutableItem build() { return new YourImmutableItem(this); }
    }
}
```

변경 불가능한 데이터 클래스와 서드 파티 보일러플레이트 생성 라이브러리를 사용하는 상위 수준 인터페이스

이전 예제에서 언급한 변경 불가능한 데이터 클래스에는 몇 가지 보일러플레이트 코드가 필요합니다. 예를 들어, `Builder` 클래스 외에 데이터 클래스의 게터 및 세터 로직이 필요합니다. Project Lombok과 같은 서드 파티 라이브러리는 이러한 유형의 보일러플레이트 코드를 생성하는 데 도움이 될 수 있습니다. 대부분의 보일러플레이트 코드를 줄이면 변경 불가능한 데이터 클래스와 AWS SDK를 사용하는 데 필요한 코드의 양을 제한할 수 있습니다. 이를 통해 코드의 생산성과 가독성이 더욱 향상됩니다. AWS SDK for Java 2.x 자세한 내용은 개발자 안내서의 Lombok과 같은 타사 라이브러리를 사용을 참조하세요.

다음 예시는 Project Lombok이 DynamoDB 향상된 클라이언트 API를 사용하는 데 필요한 코드를 간소화하는 방법을 보여줍니다.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutableLombokPutItem {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableLombokItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableLombokItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutableLombokPutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourImmutableLombokItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableLombokItem.clas
                .item(YourImmutableLombokItem.builder()
                        .pk("123")
                        .sk("cart#123")
                        .itemData("YourItemData")
                        .inventory(500)
                        .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
```

```
    }
}
```

다음 예시에서는 변경 불가능한 데이터 클래스의 변경 불가능한 데이터 객체를 보여줍니다.

```
import lombok.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;

@Builder
@DynamoDbImmutable(builder =
 YourImmutableLombokItem.YourImmutableLombokItemBuilder.class)
@Value
public class YourImmutableLombokItem {

    @Getter(onMethod_=@DynamoDbPartitionKey)
    String pk;
    @Getter(onMethod_=@DynamoDbSortKey)
    String sk;
    String itemData;
    int inventory;
}
```

`YourImmutableLombokItem` 클래스는 Project Lombok 및 AWS SDK에서 제공하는 다음과 같은 주석을 사용합니다.

- @Builder - Project Lombok에서 제공하는 데이터 클래스를 위한 복잡한 빌더 API를 생성합니다.
- @DynamoDbImmutable - `DynamoDbImmutable` 클래스를 AWS SDK에서 제공하는 DynamoDB 매핑 가능한 엔터티 주석으로 식별합니다.
- @Value - `@Data`의 변경할 수 없는 변형입니다. 기본적으로 모든 필드는 비공개 및 최종본으로 설정되고 세터가 생성되지 않습니다. Project Lombok은 이 주석을 제공합니다.

## 문서 인터페이스

AWS SDK for Java 2.x 문서 인터페이스를 사용하면 데이터 유형 설명자를 지정할 필요가 없습니다. 데이터 형식은 데이터 자체의 의미론으로 암시됩니다. 이 문서 인터페이스는 AWS SDK for Java 1.x, 문서 인터페이스와 비슷하지만, 인터페이스가 새롭게 디자인되었습니다.

다음 Document interface example는 Document 인터페이스를 사용하여 표현된 `PutItem` 직접 호출을 보여줍니다. 이 예시에서는 EnhancedDocument도 사용합니다. 향상된 문서 API를 사용하여 DynamoDB 테이블에 대해 명령을 실행하려면 먼저 테이블을 문서 테이블 스키마와 연결하여

DynamoDBTable 리소스 개체를 생성해야 합니다. 문서 테이블 스키마 빌더에는 기본 인덱스 키와 속성 변환기 제공자가 필요합니다.

AttributeConverterProvider.defaultProvider()를 사용하여 기본 유형의 문서 속성을 변환할 수 있습니다. 사용자 지정 AttributeConverterProvider 구현으로 전체 기본 동작을 변경할 수 있습니다. 단일 속성의 변환기를 변경할 수도 있습니다. AWS SDK 및 도구 참조 안내서에는 사용자 지정 변환기를 사용하는 방법에 대한 자세한 내용과 예제가 나와 있습니다. 기본적인 용도는 기본 변환기를 사용할 수 없는 도메인 클래스의 속성을 위한 것입니다. 사용자 지정 변환기를 사용하면 DynamoDB에 쓰거나 읽는 데 필요한 정보를 SDK에 제공할 수 있습니다.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedDocumentClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()

 .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk", AttributeValueType.S)
                            .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                            .build());

    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientPutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<EnhancedDocument> response =
 DYNAMODB_TABLE.putItemWithResponse(
                        PutItemEnhancedRequest.builder(EnhancedDocument.class)
                                .item(
                                    EnhancedDocument.builder()

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                                            .putString("pk", "123")
                                            .putString("sk", "cart#123")
```

```
                                                .putString("item_data", "YourItemData")
                                                .putNumber("inventory", 500)
                                                .build())
                                    .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                                    .build());
            LOGGER.info("PutItem call consumed [" +
     response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
        }

    }
```

다음 유틸리티 메서드를 사용하여 JSON 문서를 기본 Amazon DynamoDB 데이터 유형으로, 또 그 반대로 변환할 수 있습니다.

- `EnhancedDocument.fromJson(String json)` – JSON 문자열에서 새로운 EnhancedDocument 인스턴스를 생성합니다.
- `EnhancedDocument.toJson()` – 다른 JSON 개체처럼 애플리케이션에서 사용할 수 있도록 문서의 JSON 문자열 표현을 생성합니다.

## **Query** 예제와 인터페이스 비교

이 섹션에서는 다양한 인터페이스를 사용하여 동일한 `Query` 직접 호출을 표현한 것을 보여줍니다. 이러한 쿼리의 결과를 세밀하게 조정하려면 다음 사항에 유의하세요.

- DynamoDB는 하나의 특정 파티션 키 값을 대상으로 하므로, 파티션 키를 완전히 지정해야 합니다.
- 정렬 키에는 카트 항목만 이 쿼리의 대상으로 지정되도록 `begins_with`를 사용하는 키 조건 표현식이 있습니다.
- 쿼리를 최대 100개의 반환 항목으로 제한하는 데 `limit()`을 사용합니다.
- `scanIndexForward`를 false로 설정합니다. 결과는 UTF-8 바이트 순으로 반환되며, 이는 일반적으로 숫자가 가장 작은 카트 항목이 먼저 반환됨을 의미합니다. `scanIndexForward`를 false로 설정하면 순서가 반대가 되고 숫자가 가장 큰 카트 항목이 먼저 반환됩니다.
- 필터를 적용하여 기준과 일치하지 않는 모든 결과를 제거합니다. 필터링되는 데이터는 항목이 필터와 일치하는지와 관계없이 읽기 용량을 소비합니다.

### Example 하위 수준 인터페이스를 사용한 **Query**

다음 예제에서는 `keyConditionExpression`를 사용하여 이름이 `YourTableName`인 테이블을 쿼리합니다. 이를 통해 쿼리를 특정 파티션 키 값 및 특정 접두사 값으로 시작하는 정렬 키 값으로 제

한합니다. 이러한 키 조건은 DynamoDB에서 읽는 데이터의 양을 제한합니다. 마지막으로 쿼리는 `filterExpression`을 사용하여 DynamoDB에서 검색한 데이터에 필터를 적용합니다.

```
import org.slf4j.*;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;

import java.util.Map;

public class Query {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT =
 DynamoDbClient.builder().build();
    private static final Logger LOGGER = LoggerFactory.getLogger(Query.class);

    private static void query() {
        QueryResponse response = DYNAMODB_CLIENT.query(QueryRequest.builder()
                .expressionAttributeNames(Map.of("#name", "name"))
                .expressionAttributeValues(Map.of(
                    ":pk_val", AttributeValue.fromS("id#1"),
                    ":sk_val", AttributeValue.fromS("cart#"),
                    ":name_val", AttributeValue.fromS("SomeName")))
                .filterExpression("#name = :name_val")
                .keyConditionExpression("pk = :pk_val AND begins_with(sk, :sk_val)")
                .limit(100)
                .scanIndexForward(false)
                .tableName("YourTableName")
                .build());

        LOGGER.info("nr of items: " + response.count());
        LOGGER.info("First item pk: " + response.items().get(0).get("pk"));
        LOGGER.info("First item sk: " + response.items().get(0).get("sk"));
    }
}
```

### Example 문서 인터페이스를 사용한 **`Query`**

다음 예제에서는 문서 인터페이스를 사용하여 이름이 `YourTableName`인 테이블을 쿼리합니다.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
import software.amazon.awssdk.enhanced.dynamodb.model.*;

import java.util.Map;

public class DynamoDbEnhancedDocumentClientQuery {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()
                    .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk",
 AttributeValueType.S)
                    .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                    .build());
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientQuery.class);

    private void query() {
        PageIterable<EnhancedDocument> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
                        .expressionValues(Map.of(":name_val",
 AttributeValue.fromS("SomeName")))
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
                        .sortValue("cart#")
                        .build()))
                .scanIndexForward(false)
                .build());

        LOGGER.info("nr of items: " + response.items().stream().count());
```

```
        LOGGER.info("First item pk: " +
 response.items().iterator().next().getString("pk"));
        LOGGER.info("First item sk: " +
 response.items().iterator().next().getString("sk"));

    }
}
```

Example 상위 수준 인터페이스를 사용한 **Query**

다음 예시는 DynamoDB 향상된 클라이언트 API를 사용하여 이름이 YourTableName인 테이블을 쿼리합니다.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;

import java.util.Map;

public class DynamoDbEnhancedClientQuery {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromBean(DynamoDbEnhancedClientQuery.YourItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientQuery.class);

    private void query() {
        PageIterable<YourItem> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
                        .expressionValues(Map.of(":name_val",
 AttributeValue.fromS("SomeName")))
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
```

```
                    .sortValue("cart#")
                    .build()))
            .scanIndexForward(false)
            .build());

    LOGGER.info("nr of items: " + response.items().stream().count());
    LOGGER.info("First item pk: " + response.items().iterator().next().getPk());
    LOGGER.info("First item sk: " + response.items().iterator().next().getSk());
}

@DynamoDbBean
public static class YourItem {

    public YourItem() {}

    public YourItem(String pk, String sk, String name) {
        this.pk = pk;
        this.sk = sk;
        this.name = name;
    }

    private String pk;
    private String sk;
    private String name;

    @DynamoDbPartitionKey
    public void setPk(String pk) {
        this.pk = pk;
    }

    public String getPk() {
        return pk;
    }

    @DynamoDbSortKey
    public void setSk(String sk) {
        this.sk = sk;
    }

    public String getSk() {
        return sk;
    }

    public void setName(String name) {
```

```
            this.name = name;
        }

        public String getName() {
            return name;
        }
    }
}
```

변경할 수 없는 데이터 클래스를 사용하는 상위 수준 인터페이스

상위 수준의 변경 불가능한 데이터 클래스를 사용하여 Query를 수행하는 경우 엔터티 클래스 YourItem 또는 YourImmutableItem의 구성을 제외하면 코드는 상위 수준 인터페이스 예제와 동일합니다. 자세한 내용은 PutItem 예시를 참조하세요.

변경 불가능한 데이터 클래스와 서드 파티 보일러플레이트 생성 라이브러리를 사용하는 상위 수준 인터페이스

상위 수준의 변경 불가능한 데이터 클래스를 사용하여 Query를 수행하는 경우 엔터티 클래스 YourItem 또는 YourImmutableLombokItem의 구성을 제외하면 코드는 상위 수준 인터페이스 예제와 동일합니다. 자세한 내용은 PutItem 예시를 참조하세요.

## 추가 코드 예시

SDK for Java 2.x와 함께 DynamoDB를 사용하는 방법에 대한 추가 예제는 다음 코드 예제 리포지토리를 참조하세요.

- 공식 AWS 단일 작업 코드 예시
- 커뮤니티에서 유지 관리하는 단일 작업 코드 예시
- 공식 AWS 시나리오 지향 코드 예시

## 동기식 및 비동기식 프로그래밍

AWS SDK for Java 2.x는 DynamoDB와 같은 AWS 서비스에 동기식 클라이언트와 비동기식 클라이언트를 모두 제공합니다.

DynamoDbClient 및 DynamoDbEnhancedClient 클래스는 클라이언트가 서비스로부터 응답을 받을 때까지 스레드의 실행을 차단하는 동기식 메서드를 제공합니다. 이 클라이언트는 비동기식 작업이 필요 없는 경우 DynamoDB와 상호 작용하는 가장 간단한 방법입니다.

DynamoDbAsyncClient 및 DynamoDbEnhancedAsyncClient 클래스는 즉시 반환하는 비동기식 메서드를 제공하며, 응답을 기다리지 않고 제어 권한을 직접 호출하는 스레드에 넘겨줍니다. 비차단 클라이언트는 몇 개의 스레드에서 높은 동시성을 사용하여 최소한의 컴퓨팅 리소스로 I/O 요청을 효율적으로 처리할 수 있다는 이점이 있습니다. 이를 통해 처리량과 응답성이 향상됩니다.

AWS SDK for Java 2.x는 비차단 I/O에 대한 기본 지원을 사용합니다. AWS SDK for Java 1.x는 비차단 I/O를 시뮬레이션해야 했습니다.

동기식 메서드는 응답이 제공되기 전에 반환하므로, 준비되었을 때 응답을 가져올 방법이 필요합니다. AWS SDK for Java의 비동기식 메서드는 미래의 비동기식 작업 결과를 포함하는 CompletableFuture 객체를 반환합니다. 이러한 CompletableFuture 개체에서 get() 또는 join()을 직접 호출하면 결과가 나올 때까지 코드가 차단됩니다. 요청과 동시에 직접 호출을 수행하면 일반 동기식 직접 호출과 동작이 비슷합니다.

비동기 프로그래밍에 대한 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 Use asynchronous programming을 참조하세요.

## HTTP 클라이언트

모든 클라이언트를 지원하기 위해 AWS 서비스와의 통신을 처리하는 HTTP 클라이언트가 있습니다. 애플리케이션에 가장 적합한 특성을 가진 클라이언트를 선택하여 대체 HTTP 클라이언트를 연결할 수 있습니다. 어떤 것은 더 가볍고 어떤 것은 더 많은 구성 옵션을 제공합니다.

어떤 HTTP 클라이언트는 동기식 사용만 지원하는 반면 어떤 HTTP 클라이언트는 비동기식 사용만 지원합니다. 워크로드에 적합한 HTTP 클라이언트를 선택하는 데 도움이 되는 흐름도는 AWS SDK for Java 2.x 개발자 안내서의 HTTP 클라이언트 권장 사항에서 참조하세요.

다음 목록은 가능한 HTTP 클라이언트 중 일부를 보여줍니다.

주제

- Apache 기반 HTTP 클라이언트
- URLConnection 기반 HTTP 클라이언트
- Netty 기반 HTTP 클라이언트
- AWS CRT 기반 HTTP 클라이언트

### Apache 기반 HTTP 클라이언트

이 `ApacheHttpClient` 클래스는 동기식 서비스 클라이언트를 지원합니다. 동기식 사용의 기본 HTTP 클라이언트입니다. `ApacheHttpClient` 클래스 구성에 대한 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 Configure the Apache-based HTTP client에서 참조하세요.

### **`URLConnection`** 기반 HTTP 클라이언트

`UrlConnectionHttpClient` 클래스는 동기식 클라이언트를 위한 또 다른 옵션입니다. Apache 기반 HTTP 클라이언트보다 로드 속도가 빠르지만 기능이 더 적습니다. `UrlConnectionHttpClient` 클래스 구성에 대한 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 Configure the URLConnection-based HTTP client에서 참조하세요.

### Netty 기반 HTTP 클라이언트

이 `NettyNioAsyncHttpClient` 클래스는 비동기식 클라이언트를 지원합니다. 비동기식으로 사용 시 기본으로 선택됩니다. `NettyNioAsyncHttpClient` 클래스 구성에 대한 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 Configure the Netty-based HTTP client에서 참조하세요.

### AWS CRT 기반 HTTP 클라이언트

AWS Common Runtime(CRT) 라이브러리의 최신 `AwsCrtHttpClient` 및 `AwsCrtAsyncHttpClient` 클래스는 동기식 및 비동기식 클라이언트를 지원하는 더 많은 옵션입니다. 다른 HTTP 클라이언트와 비교하여 AWS CRT는 다음을 제공합니다.

- 더 빠른 SDK 시작 시간
- 더 작은 메모리 공간
- 대기 시간 단축
- 연결 상태 관리
- DNS 로드 밸런싱

`AwsCrtHttpClient` 및 `AwsCrtAsyncHttpClient` 클래스 구성에 대한 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 Configure the AWS CRT-based HTTP clients에서 참조하세요.

AWS CRT 기반 HTTP 클라이언트는 기존 애플리케이션의 하위 호환성을 깨뜨릴 수 있으므로, 기본값이 아닙니다. 하지만 DynamoDB의 경우 동기식 및 비동기식 사용에 AWS CRT 기반 HTTP 클라이언트를 사용하는 것이 좋습니다.

AWS CRT 기반 HTTP 클라이언트에 대한 소개는 AWS 개발자 도구 블로그의 Announcing availability of the AWS CRT HTTP Client in the AWS SDK for Java 2.x에서 참조하세요.

## HTTP 클라이언트 구성

클라이언트를 구성할 때 다음과 같은 다양한 구성 옵션을 제공할 수 있습니다.

- API 직접 호출의 다양한 측면에 대한 제한 시간 설정.
- TCP 연결 유지 활성화.
- 오류 발생 시 재시도 정책 제어.
- 실행 인터셉터 인스턴스가 수정할 수 있는 실행 속성 지정. 실행 인터셉터는 API 요청 및 응답의 실행을 가로채는 코드를 작성할 수 있습니다. 이를 통해 지표를 게시하고 진행 중인 요청을 수정하는 등의 작업을 수행할 수 있습니다.
- HTTP 헤더 추가 또는 조작.
- 클라이언트 측 성능 지표 추적 활성화. 이 기능을 사용하면 애플리케이션의 서비스 클라이언트에 대한 지표를 수집하고 Amazon CloudWatch에서 출력을 분석할 수 있습니다.
- 비동기식 재시도 및 제한 시간 작업과 같은 일정 예약 작업에 사용할 대체 실행자 서비스 지정.

서비스 클라이언트 `Builder` 클래스에 `ClientOverrideConfiguration` 객체를 제공하여 구성을 제어합니다. 다음 섹션의 일부 코드 예시에서 이를 확인할 수 있습니다.

`ClientOverrideConfiguration`은 표준 구성 선택 사항을 제공합니다. 다양한 플러그 가능 HTTP 클라이언트에는 구현별 구성 기능도 있습니다.

이 섹션의 주제

- 제한 시간 구성
- RetryMode
- DefaultsMode
- 연결 유지 구성

### 제한 시간 구성

클라이언트 구성을 조정하여 서비스 직접 호출과 관련된 다양한 제한 시간을 제어할 수 있습니다. DynamoDB는 다른 AWS 서비스에 비해 지연 시간이 더 짧습니다. 따라서 네트워킹 문제가 발생할 경우 빠르게 실패할 수 있도록 이러한 속성을 조정하여 제한 시간 값을 낮추는 것이 좋습니다.

DynamoDB 클라이언트에서 `ClientOverrideConfiguration`을 사용하거나 기본 HTTP 클라이언트 구현에서 세부 구성 옵션을 변경하여 지연 시간 관련 동작을 사용자 지정할 수 있습니다.

`ClientOverrideConfiguration`을 사용하여 다음과 같은 영향력 있는 속성을 구성할 수 있습니다.

- `apiCallAttemptTimeout` – 포기하고 제한 시간이 초과되기 전에 한 번의 HTTP 요청 시도가 완료되기까지 기다리는 시간입니다.
- `apiCallTimeout` – 클라이언트가 API 직접 호출을 완전히 실행해야 하는 시간입니다. 여기에는 재시도를 포함한 모든 HTTP 요청으로 구성된 요청 핸들러 실행이 포함됩니다.

AWS SDK for Java 2.x는 연결 제한 시간 및 소켓 제한 시간 등의 일부 제한 시간 옵션에 기본값을 제공합니다. SDK는 API 직접 호출 제한 시간 또는 개별 API 직접 호출 시도 제한 시간에 기본값을 제공하지 않습니다. 이러한 제한 시간이 `ClientOverrideConfiguration`에 설정되지 않은 경우 SDK는 전체 API 직접 호출 제한 시간 대신 소켓 제한 시간 값을 사용합니다. 소켓 제한 시간의 기본값은 30초입니다.

### RetryMode

제한 시간 구성과 관련하여 고려해야 하는 또 다른 구성은 `RetryMode` 구성 객체입니다. 이 구성 객체에는 재시도 동작 모음이 포함되어 있습니다.

SDK for Java 2.x는 다음 재시도 모드를 지원합니다.

- `legacy` – 명시적으로 변경하지 않는 경우 기본 재시도 모드입니다. 이 재시도 모드는 Java SDK에만 해당됩니다. 최대 3회 재시도 또는 DynamoDB와 같은 서비스의 경우 최대 8회 재시도할 수 있습니다.
- `standard` - 다른 AWS SDK와 더 일관적이기 때문에 '표준'이라는 이름이 지정되었습니다. 이 모드는 첫 번째 재시도를 위해 0ms에서 1,000ms 사이의 임의의 시간 동안 기다립니다. 다시 재시도해야 하는 경우 이 모드는 0ms에서 1,000ms 사이의 또 다른 임의의 시간을 선택하여 2를 곱합니다. 추가 재시도가 필요한 경우 같은 범위에서 임의로 선택한 시간에 4를 곱하는 식으로 반복합니다. 각 대기 시간은 20초로 제한됩니다. 이 모드는 `legacy` 모드보다 더 많이 감지된 장애 조건에 대해 재시도를 수행합니다. DynamoDB의 경우 numRetries로 재정의하지 않는 한 모두 합쳐 최대 3회까지 시도합니다.
- `adaptive` – `standard` 모드를 기반으로 하며 AWS 요청 비율을 동적으로 제한하여 성공률을 극대화합니다. 이렇게 하면 요청 지연 시간이 길어질 수 있습니다. 예측 가능한 지연 시간이 중요한 경우에는 적응형 재시도 모드를 사용하지 않는 것이 좋습니다.

이러한 재시도 모드의 확장된 정의는 AWS SDK 및 도구 참조 안내서의 Retry behavior에서 확인할 수 있습니다.

재시도 정책

모든 RetryMode 구성에는 하나 이상의 RetryCondition 구성을 기반으로 구축된 RetryPolicy가 있습니다. TokenBucketRetryCondition은 DynamoDB SDK 클라이언트 구현의 재시도 동작에 특히 중요합니다. 이 조건은 토큰 버킷 알고리즘을 사용하여 SDK의 재시도 횟수를 제한합니다. 선택한 재시도 모드에 따라 제한 예외가 TokenBucket에서 토큰을 뺄 수도 있고 빼지 않을 수도 있습니다.

클라이언트에서 제한 예외 또는 일시적 서버 오류와 같은 재시도 가능한 오류가 발생하면 SDK는 요청을 자동으로 재시도합니다. 재시도 횟수와 속도를 제어할 수 있습니다.

클라이언트를 구성할 때 다음 파라미터를 지원하는 RetryPolicy를 제공할 수 있습니다.

- numRetries – 요청이 실패한 것으로 간주되기 전에 적용해야 하는 최대 재시도 횟수입니다. 사용하는 재시도 모드와 관계없이 기본값은 8입니다.

  > **Warning**
  >
  > 이 기본값을 변경하려면 신중하게 고려하시기 바랍니다.

- backoffStrategy - 재시도에 적용할 BackoffStrategy으로, FullJitterBackoffStrategy이 기본 전략입니다. 이 전략은 현재 재시도 횟수, 기본 지연 및 최대 백오프 시간을 기준으로 추가 재시도 사이에 기하급수적 지연을 수행합니다. 그런 다음 지터를 추가하여 약간의 무작위성을 제공합니다. 기하급수적 지연에 사용되는 기본 지연은 재시도 모드와 관계없이 25ms입니다.
- retryCondition – RetryCondition은 요청을 재시도할지 여부를 결정합니다. 기본적으로 재시도 가능하다고 판단되는 특정 HTTP 상태 코드 및 예외 집합을 재시도합니다. 대부분의 경우 기본 구성이면 충분합니다.

다음 코드는 대체 재시도 정책을 제공합니다. 총 5번의 재시도(총 6번의 요청)를 지정합니다. 첫 번째 재시도는 약 100ms 지연 후에 이루어져야 하며, 각 추가 재시도는 최대 1초 지연까지 기하급수적으로 2배 증가해야 합니다.

```
DynamoDbClient client = DynamoDbClient.builder()
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .retryPolicy(RetryPolicy.builder()
            .backoffStrategy(FullJitterBackoffStrategy.builder()
```

```
                .baseDelay(Duration.ofMillis(100))
                .maxBackoffTime(Duration.ofSeconds(1))
                .build())
            .numRetries(5)
            .build())
        .build())
    .build();
```

## DefaultsMode

`ClientOverrideConfiguration` 및 `RetryMode`가 관리하지 않는 제한 시간 속성은 일반적으로 `DefaultsMode`를 지정하여 암묵적으로 구성됩니다.

AWS SDK for Java 2.x(버전 2.17.102 이상)에 `DefaultsMode`에 대한 지원이 도입되었습니다. 이 기능은 HTTP 통신 설정, 재시도 동작, 서비스 리전 엔드포인트 설정 및 잠재적인 모든 SDK 관련 구성과 같은 일반적인 구성 가능 설정에 대한 기본값 집합을 제공합니다. 이 기능을 사용하면 일반 사용 시나리오에 맞게 조정된 새 구성 기본값을 얻을 수 있습니다.

기본 모드는 모든 AWS SDK에서 표준화되어 있습니다. SDK for Java 2.x는 다음과 같은 기본 모드를 지원합니다.

- `legacy` - AWS SDK에 따라 달라지고 `DefaultsMode`가 설정되기 전에 존재했던 기본 설정을 제공합니다.
- `standard` - 대부분의 시나리오에 최적화되지 않은 기본 설정을 제공합니다.
- `in-region` - 표준 모드를 기반으로 구축하며 동일한 AWS 리전에서 AWS 서비스를 직접 호출하는 애플리케이션에 맞게 조정된 설정을 포함합니다.
- `cross-region` - 표준 모드를 기반으로 구축하며 동일한 다른 리전에서 AWS 서비스를 직접 호출하는 애플리케이션에 제한 시간이 긴 설정을 포함합니다.
- `mobile` - 표준 모드를 기반으로 구축하며 지연 시간이 긴 모바일 애플리케이션에 맞게 조정된 제한 시간이 긴 설정을 포함합니다.
- `auto`— 표준 모드를 기반으로 구축하며 실험적 기능을 포함합니다. SDK는 런타임 환경을 검색하여 적절한 설정을 자동으로 결정합니다. 자동 감지는 휴리스틱 기반이며 정확도가 100%는 아닙니다. 런타임 환경을 확인할 수 없는 경우 표준 모드가 사용됩니다. 자동 탐지는 인스턴스 메타데이터와 사용자 데이터를 쿼리할 수 있으며, 이로 인해 지연이 발생할 수 있습니다. 시작 지연 시간이 애플리케이션에 중요한 경우에는 명시적 지연 시간을 `DefaultsMode`을 대신 선택하는 것이 좋습니다.

다음과 같은 방법으로 기본 모드를 구성할 수 있습니다.

- `AwsClientBuilder.Builder#defaultsMode(DefaultsMode)`를 통해 클라이언트에게 직접 전달합니다.
- `defaults_mode` 프로필 파일 속성을 통해 구성 프로필에서 구성합니다.
- `aws.defaultsMode` 시스템 속성을 통해 전역적으로 구성합니다.
- `AWS_DEFAULTS_MODE` 환경 변수를 통해 전역적으로 구성합니다.

> **Note**
>
> `legacy` 외의 모든 모드에서는 모범 사례가 발전함에 따라 벤딩 기본값이 변경될 수 있습니다. 따라서 `legacy` 이외의 모드를 사용하는 경우 SDK를 업그레이드할 때 테스트를 수행하는 것이 좋습니다.

AWS SDK 및 도구 참조 안내서의 스마트 구성 기본값은 다양한 기본 모드의 구성 속성 및 기본값 목록을 제공합니다.

애플리케이션의 특성과 애플리케이션이 상호 작용하는 AWS 서비스에 따라 기본 모드 값을 선택합니다.

이러한 값은 다양한 AWS 서비스 선택지를 염두에 두고 구성되었습니다. DynamoDB 테이블과 애플리케이션이 모두 한 리전에 배포되는 일반적인 DynamoDB 배포의 경우 `standard` 기본 모드 중에서 `in-region` 기본 모드가 가장 적합합니다.

Example 지연 시간이 짧은 직접 호출을 위해 조정된 DynamoDB SDK 클라이언트 구성

다음 예시는 지연 시간이 짧을 것으로 예상되는 DynamoDB 직접 호출의 제한 시간을 더 낮은 값으로 조정합니다.

```
DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.builder()
    .defaultsMode(DefaultsMode.IN_REGION)
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder())
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(3))
        .apiCallAttemptTimeout(Duration.ofMillis(500))
        .build())
    .build();
```

개별 HTTP 클라이언트 구현을 통해 제한 시간 및 연결 사용 동작을 훨씬 더 세밀하게 제어할 수 있습니다. 예를 들어, AWS CRT 기반 클라이언트의 경우 클라이언트가 사용된 연결의 상태를 능동적으로

모니터링할 수 있도록 ConnectionHealthConfiguration을 활성화할 수 있습니다. 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 AWS CRT 기반 HTTP 클라이언트의 고급 구성을 참조하세요.

### 연결 유지 구성

연결 유지를 활성화하면 연결을 재사용하여 지연 시간을 줄일 수 있습니다. 연결 유지에는 HTTP 연결 유지와 TCP 연결 유지라는 두 가지 종류가 있습니다.

- HTTP 연결 유지는 클라이언트와 서버 간의 HTTPS 연결을 유지하려고 시도하므로 이후 요청에서 해당 연결을 재사용할 수 있습니다. 이렇게 하면 이후 요청 시 무거운 HTTPS 인증을 건너뛸 수 있습니다. HTTP 연결 유지는 모든 클라이언트에서 기본적으로 활성화되어 있습니다.
- TCP 연결 유지는 기본 운영 체제에 소켓 연결을 통해 작은 패킷을 전송하도록 요청하여 소켓의 연결이 유지되어 있는지 확인하고 연결 해제가 발생하면 즉시 감지하도록 합니다. 이렇게 하면 나중에 요청할 때 연결이 해제된 소켓을 사용하느라 시간을 허비하지 않아도 됩니다. TCP 연결 유지는 모든 클라이언트에서 기본적으로 비활성화되어 있습니다. 다음 코드 예제에서는 각 HTTP 클라이언트에서 이를 활성화하는 방법을 보여줍니다. CRT 기반이 아닌 모든 HTTP 클라이언트에 대해 활성화된 경우 실제 연결 유지 메커니즘은 운영 체제에 따라 달라집니다. 따라서 운영 체제를 통해 추가 TCP 연결 유지 값(예: 제한 시간 및 패킷 수)을 구성해야 합니다. Linux 또는 macOS의 sysctl을 사용하거나 Windows의 레지스트리 값을 사용하여 이 작업을 수행할 수 있습니다.

Example Apache 기반 HTTP 클라이언트에서 TCP 연결 유지를 활성화하는 방법

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(ApacheHttpClient.builder().tcpKeepAlive(true))
    .build();
```

**URLConnection** 기반 HTTP 클라이언트

URLConnection 기반 HTTP 클라이언트 HttpURLConnection을 사용하는 동기식 클라이언트에는 연결 유지를 활성화하는 메커니즘이 없습니다.

Example Netty 기반 HTTP 클라이언트에서 TCP 연결 유지 활성화

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Example AWS CRT 기반 HTTP 클라이언트에서 TCP 연결 유지 활성화

AWS CRT 기반 HTTP 클라이언트를 사용하여 TCP 연결 유지를 활성화하고 기간을 제어할 수 있습니다.

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(AwsCrtHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

비동기식 DynamoDB 클라이언트를 사용하는 경우 다음 코드와 같이 TCP 연결 유지를 활성화할 수 있습니다.

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

## 오류 처리

예외 처리와 관련하여 AWS SDK for Java 2.x는 런타임 (확인되지 않은) 예외를 사용합니다.

모든 SDK 예외를 포함하는 기본 예외는 Java 확인되지 않은 `RuntimeException`에서 확장된 `SdkServiceException`입니다. 이를 포착하면 SDK에서 발생하는 모든 예외를 포착할 수 있습니다.

`SdkServiceException`에는 `AwsServiceException`이라는 하위 클래스가 있습니다. 이 하위 클래스는 AWS 서비스와 통신할 때 문제가 발생했음을 나타냅니다. `DynamoDbException`이라는 하위 클래스가 있는데, 이는 DynamoDB와의 통신에 문제가 있음을 나타냅니다. 이것을 포착하면 DynamoDB와 관련된 모든 예외를 포착할 수 있지만, 다른 SDK 예외는 포착하지 못합니다.

`DynamoDbException` 아래에는 더 구체적인 예외 유형이 있습니다. 이러한 예외 유형 중 일부는 `TableAlreadyExistsException`과 같은 컨트롤 플레인 작업에 적용됩니다. 다른 예외 유형은 데이터 영역 작업에 적용됩니다. 다음은 일반적인 데이터 영역 예외의 예입니다.

- `ConditionalCheckFailedException` – 요청에서 false로 평가된 조건을 지정했습니다. 예를 들어 어떤 항목에서 조건부 업데이트 수행을 시도했지만 속성의 실제 값이 조건에서 예상되는 값과 일치하지 않았을 수 있습니다. 이러한 방식으로 실패한 요청은 재시도되지 않습니다.

다른 상황에서는 구체적인 예외가 정의되어 있지 않습니다. 예를 들어 요청이 제한되면 구체적인 `ProvisionedThroughputExceededException`이 발생하고 다른 경우에는 보다 일반적인 `DynamoDbException`이 발생할 수 있습니다. 어느 경우에나 `isThrottlingException()`이 `true`를 반환하는지 확인하여 제한으로 인해 예외가 발생했는지 확인할 수 있습니다.

애플리케이션 요구 사항에 따라 모든 `AwsServiceException` 또는 `DynamoDbException` 인스턴스를 포착할 수 있습니다. 그러나 상황에 따라 다른 동작이 필요한 경우가 많습니다. 상태 확인 실패를 처리하는 논리는 제한을 처리하는 것과 다릅니다. 처리할 예외적인 경로를 정의하고 대체 경로를 테스트해 보세요. 이를 통해 모든 관련 시나리오를 처리할 수 있습니다.

발생할 수 있는 일반적인 오류 목록은 DynamoDB 관련 오류 처리 섹션을 참조하세요. Amazon DynamoDB API 참조의 Common Errors도 참조하세요. 또한, API 참조는 각 API 작업(예: `Query` 작업)에서 발생할 수 있는 정확한 오류를 제공합니다. 예외 처리에 대한 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 Exception handling for the AWS SDK for Java 2.x를 참조하세요.

## AWS 요청 ID

각 요청에는 요청 ID가 포함되어 있으며, 이 ID는 AWS Support과 협력하여 문제를 진단하는 경우 유용하게 사용할 수 있습니다. `SdkServiceException`에서 파생된 각 예외에는 요청 ID를 검색하는 데 사용할 수 있는 `requestId()` 메서드가 있습니다.

## 로깅

SDK에서 제공하는 로깅을 사용하면 클라이언트 라이브러리에서 중요한 메시지를 포착하고 심층적으로 디버깅하는 데 모두 유용할 수 있습니다. 로거는 계층적이며 SDK는 `software.amazon.awssdk`를 루트 로거로 사용합니다. 수준은 `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `ALL`, `OFF` 중 하나로 구성할 수 있습니다. 구성된 수준은 해당 로거에 적용되며 로거 계층 구조까지 내려갑니다.

AWS SDK for Java 2.x에서는 로깅에 Simple Logging Façade for Java(SLF4J)를 사용합니다. 이는 다른 로거 주변의 추상화 계층 역할을 하며, 이를 사용하여 원하는 로거를 연결할 수 있습니다. 로거 연결에 대한 지침은 SLF4J 사용 설명서를 참조하세요.

각 로거에는 특정한 동작이 있습니다. 기본적으로 Log4j 2.x 로거는 로그 이벤트를 `System.out`에 추가하고 기본값은 `ERROR` 로그 수준에 추가하는 `ConsoleAppender`를 생성합니다.

SLF4J 내에 포함된 SimpleLogger 로거는 기본적으로 `System.err`을 출력하며 `INFO` 로그 수준을 기본값으로 사용합니다.

모든 프로덕션 배포에서 출력 수량을 제한하면서 SDK 클라이언트 라이브러리에서 중요한 메시지를 포착하려면 `software.amazon.awssdk` 수준을 `WARN`으로 설정하는 것이 좋습니다.

SLF4J가 클래스 경로에서 지원되는 로거를 찾을 수 없는 경우(SLF4J 바인딩 없음) 기본적으로 무작업 구현으로 설정됩니다. 이 구현으로 인해 SLF4J가 클래스 경로에서 로거 구현을 찾을 수 없다는 내용의 메시지가 `System.err`에 로깅됩니다. 이러한 상황을 방지하려면 로거 구현을 추가해야 합니다. `org.slf4j.slf4j-simple` 또는 `org.apache.logging.log4j.log4j-slf4j2-imp`와 같은 아티팩트에 Apache Maven `pom.xml`의 종속성을 추가하면 됩니다.

애플리케이션 구성에 로깅 종속성을 추가하는 것을 포함하여 SDK에서 로깅을 구성하는 방법에 대한 자세한 내용은 AWS SDK for Java 개발자 안내서의 Logging with the SDK for Java 2.x를 참조하세요.

`Log4j2.xml` 파일의 다음 구성은 Apache Log4j 2 로거를 사용하는 경우 로깅 동작을 조정하는 방법을 보여줍니다. 이 구성은 루트 로거 수준을 `WARN`으로 설정합니다. 계층의 모든 로거는 `software.amazon.awssdk` 로거를 포함하여 이 로그 수준을 상속합니다.

기본적으로 출력은 `System.out`으로 이동합니다. 다음 예시에서는 여전히 기본 출력 Log4j 어펜더를 재정의하여 맞춤형 Log4j `PatternLayout`을 적용합니다.

**`Log4j2.xml` 구성 파일 예**

다음 구성은 모든 로거 계층 구조에 대해 `ERROR` 및 `WARN` 수준의 메시지를 콘솔에 로깅합니다.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
    <Root level="WARN">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

## AWS 요청 ID 로깅

문제가 발생한 경우 예외 내에서 요청 ID를 찾을 수 있습니다. 하지만 예외를 생성하지 않는 요청에 대한 요청 ID를 원하면 로깅을 사용하면 됩니다.

`software.amazon.awssdk.request` 로거는 DEBUG 수준에서 요청 ID를 출력합니다. 다음 예시는 이전 configuration example를 확장하여 루트 로거 수준을 ERROR로, `software.amazon.awssdk`를 WARN 수준으로, `software.amazon.awssdk.request`를 DEBUG 수준으로 유지합니다. 이러한 수준을 설정하면 요청 ID 및 기타 요청 관련 세부 정보(예: 엔드포인트 및 상태 코드)를 파악하는 데 도움이 됩니다.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
    <Root level="ERROR">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
    <Logger name="software.amazon.awssdk" level="WARN" />
    <Logger name="software.amazon.awssdk.request" level="DEBUG" />
  </Loggers>
</Configuration>
```

다음은 로그 출력의 예입니다:

```
2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Sending Request:
 DefaultSdkHttpFullRequest(httpMethod=POST, protocol=https, host=dynamodb.us-
east-1.amazonaws.com, encodedPath=/, headers=[amz-sdk-invocation-id, Content-Length,
 Content-Type, User-Agent, X-Amz-Target], queryParameters=[])
 2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Received
 successful response: 200, Request ID:
 QS9DUMME2NHEDH8TGT9N5V53OJVV4KQNSO5AEMVJF66Q9ASUAAJG, Extended Request ID: not
 available
```

# 페이지 매김

Query 및 Scan과 같은 일부 요청의 경우 단일 요청에서 반환되는 데이터 크기가 제한되므로, 후속 페이지를 가져오려면 반복적으로 요청해야 합니다.

Limit 파라미터를 사용하여 각 페이지에 대해 읽을 항목의 최대 수를 제어할 수 있습니다. 예를 들어, Limit 파라미터를 사용하여 마지막 10개 항목만 검색할 수 있습니다. 이 제한은 필터링이 적용되기 전에 테이블에서 읽을 항목 수를 지정합니다. 필터링 후 정확히 10개를 원한다고 해서 지정할 수 있는 방법은 없습니다. 필터링 전의 개수를 제어하고 실제로 10개 항목을 검색한 경우에만 클라이언트 측에서 확인할 수 있습니다. 제한과 관계없이 응답의 최대 크기는 항상 1MB입니다.

API 응답에 LastEvaluatedKey가 포함될 수 있습니다. 이는 개수 제한 또는 크기 제한에 도달하여 응답이 종료되었음을 나타냅니다. 이 키는 해당 응답에 대해 마지막으로 평가된 키입니다. API와 직접 상호 작용하면 이 LastEvaluatedKey를 검색하고 ExclusiveStartKey로 후속 직접 호출에 전달하여 해당 시작 지점부터 다음 청크를 읽을 수 있습니다. LastEvaluatedKey가 반환되지 않으면 Query 또는 Scan API 직접 호출과 일치하는 항목이 더 이상 없는 것입니다.

다음 예시에서는 하위 수준 인터페이스를 사용하여 keyConditionExpression 파라미터에 따라 항목을 100개로 제한합니다.

```
QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
        .expressionAttributeValues(Map.of(
                ":pk_val", AttributeValue.fromS("123"),
                ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName(TABLE_NAME);

while (true) {
    QueryResponse queryResponse = DYNAMODB_CLIENT.query(queryRequestBuilder.build());

    queryResponse.items().forEach(item -> {
        LOGGER.info("item PK: [" + item.get("pk") + "] and SK: [" + item.get("sk") +
 "]");
    });

    if (!queryResponse.hasLastEvaluatedKey()) {
        break;
    }
    queryRequestBuilder.exclusiveStartKey(queryResponse.lastEvaluatedKey());
}
```

AWS SDK for Java 2.x는 자동으로 다음 결과 페이지를 얻기 위해 여러 서비스 직접 호출을 수행하는 자동 페이지 매김 메서드를 제공함으로써 DynamoDB와의 상호 작용을 단순화할 수 있습니다. 이렇게 하면 코드가 단순해지지만, 페이지를 수동으로 읽으면서 유지할 수 있는 리소스 사용에 대한 일부 제어 기능이 제거됩니다.

DynamoDB 클라이언트에서 사용할 수 있는 `QueryPaginator` 및 `ScanPaginator`와 같은 `Iterable` 메서드를 사용하여 SDK가 페이지 매김을 처리합니다. 이러한 메서드의 반환 유형은 모든 페이지를 반복하는 데 사용할 수 있는 사용자 지정 반복자입니다. SDK는 내부적으로 서비스 직접 호출을 처리합니다. Java Stream API를 사용하여 다음 예제와 같이 `QueryPaginator`의 결과를 처리할 수 있습니다.

```
QueryPublisher queryPublisher =
    DYNAMODB_CLIENT.queryPaginator(QueryRequest.builder()
        .expressionAttributeValues(Map.of(
            ":pk_val", AttributeValue.fromS("123"),
            ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName("YourTableName")
        .build());

queryPublisher.items().subscribe(item ->
    System.out.println(item.get("itemData"))).join();
```

## 데이터 클래스 주석

Java SDK는 데이터 클래스의 속성에 적용할 수 있는 여러 주석을 제공합니다. 이러한 주석은 SDK가 속성과 상호 작용하는 방식에 영향을 줍니다. 주석을 추가하면 속성이 암시적 원자성 카운터 역할을 하도록 하거나, 자동 생성된 타임스탬프 값을 유지하거나, 항목 버전 번호를 추적할 수 있습니다. 자세한 내용은 Data class annotations를 참조하세요.

# DynamoDB 관련 오류 처리

이 단원에서는 런타임 오류와 그러한 오류를 처리하는 방법을 설명합니다. 또한 Amazon DynamoDB 관련 오류 메시지 및 코드를 설명합니다. 모든 AWS 서비스에 적용되는 일반적인 오류 목록은 액세스 관리를 참조하세요.

주제

- 오류 구성 요소

- 트랜잭션 오류
- 오류 메시지 및 코드
- 애플리케이션에서의 오류 처리
- 오류 재시도 횟수 및 지수 백오프
- 일괄 작업 및 오류 처리


## 오류 구성 요소

프로그램이 요청을 보내면 DynamoDB는 요청 처리를 시도합니다. 요청이 성공하면 DynamoDB는 요청된 작업의 결과와 함께 HTTP 성공 상태 코드(`200 OK`)를 반환합니다.

요청이 실패하면 DynamoDB는 오류를 반환합니다. 오류에는 세 가지 구성 요소가 있습니다.

- HTTP 상태 코드(예: `400`).
- 예외 이름(예: `ResourceNotFoundException`).
- 오류 메시지(예: `Requested resource not found: Table: tablename not found`).

AWS SDK는 사용자가 적절한 조치를 취할 수 있도록 애플리케이션에 오류를 전파합니다. 예를 들어 Java 프로그램에서는 `ResourceNotFoundException`을 처리하기 위해 `try-catch` 로직을 작성할 수 있습니다.

AWS SDK를 사용하지 않는 경우 DynamoDB에서 하위 수준 응답의 콘텐츠를 구문 분석해야 합니다. 다음은 이러한 응답의 예입니다.

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: LDM6CJP8RMQ1FHKSC1RBVJFPNVV4KQNSO5AEMF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 240
Date: Thu, 15 Mar 2012 23:56:23 GMT

{"__type":"com.amazonaws.dynamodb.v20120810#ResourceNotFoundException",
"message":"Requested resource not found: Table: tablename not found"}
```

## 트랜잭션 오류

트랜잭션 오류에 대한 자세한 내용은 DynamoDB의 트랜잭션 충돌 처리 섹션을 참조하세요.

# 오류 메시지 및 코드

다음은 DynamoDB가 반환하는 예외 목록을 HTTP 상태 코드를 기준으로 그룹화한 목록입니다. 재시도 가능?이 예라면 동일한 요청을 다시 제출할 수 있습니다. 재시도 가능?이 아니요라면 새로운 요청을 제출하기 전에 클라이언트 측에서 문제를 해결해야 합니다.

## HTTP 상태 코드 400

HTTP `400` 상태 코드는 인증 실패, 필수 파라미터 누락, 테이블의 할당 처리량 초과 등 요청에 문제가 있음을 의미합니다. 요청을 다시 제출하기 전에 애플리케이션의 문제를 해결해야 합니다.

### AccessDeniedException

메시지: 액세스가 거부되었습니다.

클라이언트가 요청에 정확히 서명하지 않았습니다. AWS SDK를 사용 중인 경우 요청에 자동으로 서명됩니다. 그렇지 않은 경우 AWS 일반 참조의 서명 버전 4 서명 프로세스로 이동하세요.

재시도 가능? 아니요

### ConditionalCheckFailedException

메시지: 조건부 요청이 실패했습니다.

false로 평가된 조건을 지정했습니다. 예를 들어 어떤 항목에서 조건부 업데이트 수행을 시도했지만 속성의 실제 값이 조건에서 예상되는 값과 일치하지 않았을 수 있습니다.

재시도 가능? 아니요

### IncompleteSignatureException

메시지: 요청 서명이 AWS 표준을 준수하지 않습니다.

요청 서명에 일부 필요한 구성 요소가 빠져있습니다. AWS SDK를 사용 중인 경우 요청에 자동으로 서명됩니다. 그렇지 않은 경우 AWS 일반 참조의 서명 버전 4 서명 프로세스로 이동하세요.

재시도 가능? 아니요

### ItemCollectionSizeLimitExceededException

메시지: 그룹 크기 제한이 초과됐습니다.

로컬 보조 인덱스가 포함된 테이블에서 동일한 파티션 키 값의 항목 그룹이 최대 크기 제한 10GB를 초과하였습니다. 항목 그룹에 대한 자세한 내용은 로컬 보조 인덱스의 항목 컬렉션 단원을 참조하세요.

재시도 가능? 예

LimitExceededException

메시지: 임의 구독자의 작업이 너무 많습니다.

동시 제어 플레인 작업이 너무 많습니다. `CREATING`, `DELETING` 또는 `UPDATING` 상태의 누적 테이블 및 인덱스 수는 500을 초과할 수 없습니다.

재시도 가능? 예

MissingAuthenticationTokenException

메시지: 요청에는 유효한(등록된) AWS 액세스 키 ID가 포함되어야 합니다.

요청이 필요한 인증 헤더를 포함하지 않았거나 잘못된 형식입니다. DynamoDB 하위 수준 API을(를) 참조하세요.

재시도 가능? 아니요

ProvisionedThroughputExceededException

메시지: 테이블 또는 하나 이상의 글로벌 보조 인덱스에게 허용되는 최대 프로비저닝된 처리량을 초과하였습니다. 프로비저닝된 처리량과 소비한 처리량의 성능 지표를 보려면 Amazon CloudWatch 콘솔을 여세요.

예: 요청량이 너무 높습니다. DynamoDB용 AWS SDK가 이 예외를 수신하는 요청을 자동으로 재시도합니다. 재시도 대기열이 너무 많아 완료하지 못하는 경우를 제외하고 결국 요청이 성공합니다. 오류 재시도 횟수 및 지수 백오프를 사용하여 요청 빈도를 줄입니다.

재시도 가능? 예

ReplicatedWriteConflictException

메시지: One or more items in this request are being modified by a request in another Region.

예: 현재 다른 리전의 요청에 의해 수정 중인 다중 리전 강력한 일관성(MRSC) 글로벌 테이블의 항목에 대해 쓰기 작업이 요청되었습니다.

재시도 가능? 예

RequestLimitExceeded

메시지: 처리량이 계정에 대한 현재 처리량 한도를 초과합니다. 한도 증가를 요청하려면 AWS Support(https://aws.amazon.com/support)에 문의하세요.

예: 온디맨드 요청량이 허용된 계정 처리량을 초과했으며 더 이상 테이블의 크기를 조정할 수 없습니다.

재시도 가능? 예

ResourceInUseException

메시지: 변경하려는 리소스가 현재 사용 중입니다.

예: 기존 테이블을 다시 생성하려고 하거나, 또는 현재 `CREATING` 상태인 테이블을 삭제하려고 했습니다.

재시도 가능? 아니요

ResourceNotFoundException

메시지: 요청한 리소스를 찾을 수 없습니다.

예제: 요청한 테이블이 존재하지 않거나 너무 일찍 `CREATING` 상태가 되었습니다.

재시도 가능? 아니요

ThrottlingException

메시지: 요청량이 허용 처리량을 초과하였습니다.

이 예외는 THROTTLING_EXCEPTION 상태 코드와 함께 AmazonServiceException 응답으로 반환됩니다. 제어 플레인 API 작업을 너무 빠르게 수행하는 경우 이 예외가 반환될 수 있습니다.

요청 속도가 너무 높으면 온디맨드 모드를 사용하는 테이블의 경우 모든 데이터 플레인 API 작업에 대해 이 예외가 반환될 수 있습니다. 온디맨드 규모 조정에 대한 자세한 내용은 초기 처리량 및 규모 조정 속성 섹션을 참조하세요.

재시도 가능? 예

UnrecognizedClientException

메시지: 액세스 키 ID 또는 보안 토큰이 잘못되었습니다.

요청 서명이 잘못되었습니다. 가장 가능성이 높은 원인은 잘못된 AWS 액세스 키 ID 또는 보안 키입니다.

재시도 가능? 예

ValidationException

메시지: 발생하는 구체적 오류에 따라 다릅니다

이 오류는 필수 파라미터의 누락, 값의 범위 이탈 또는 일치하지 않는 데이터 형식 등 몇 가지 이유로 발생할 수 있습니다. 오류 메시지에는 요청에서 오류를 일으킨 특정 부분에 대한 정보가 포함됩니다.

재시도 가능? 아니요

## HTTP 상태 코드 5xx

HTTP 5xx 상태 코드는 AWS가 해결해야 하는 문제를 나타냅니다. 이것은 일시적 오류일 수 있으므로 성공할 때까지 요청을 다시 시도할 수 있습니다. 그렇지 않은 경우 AWS Service Health Dashboard에서 서비스 운영 문제가 있는지 확인하세요.

자세한 내용은 Amazon DynamoDB에서 HTTP 5xx 오류를 해결하려면 어떻게 해야 하나요?를 참조하세요.

InternalServerError (HTTP 500)

DynamoDB에서 요청을 처리할 수 없습니다.

재시도 가능? 예

> **Note**
>
> 항목 작업 시 내부 서버 오류가 발생할 수 있습니다. 이러한 오류는 테이블 수명 동안 예상됩니다. 실패한 요청은 즉시 다시 시도할 수 있습니다.
> 쓰기 작업에 대한 상태 코드 500을 수신하면 작업이 성공했거나 실패했을 수 있습니다. 쓰기 작업이 `TransactWriteItem` 요청이면 작업을 다시 시도해도 좋습니다. 쓰기 작업이 `PutItem`, `UpdateItem` 또는 `DeleteItem` 같은 단일 항목 쓰기 요청인 경우 작업을 다시 시도하기 전에 애플리케이션이 항목의 상태를 읽어야 합니다. 그리고 DynamoDB 조건 표현식 CLI 예제를 사용하여 이전 작업의 성공 여부와 관계없이 재시도한 후에도 항목이 올바른 상태로 유지되도록 해야 합니다. 멱등성이 쓰기 작업의 요구 사항인 경우 동일한 작업을 수행하려는 여러 시도를 명확하게 하기 위해 `ClientRequestToken`을 자동으로 지정하여 멱등원 요청을 지원하는 `TransactWriteItem`을 사용하세요.

ServiceUnavailable (HTTP 503)

DynamoDB를 현재 사용할 수 없습니다. (이것은 일시적 상태여야 합니다.)

재시도 가능? 예

## 애플리케이션에서의 오류 처리

애플리케이션의 원활한 실행을 위해서는 오류를 발견하고 대응할 수 있는 로직을 추가해야 합니다. 일반적인 접근법에는 `try-catch` 블록 또는 `if-then` 문의 사용이 포함됩니다.

AWS SDK는 자체적으로 재시도 및 오류 검사를 수행합니다. AWS SDK 사용 중에 오류가 발생하면 해당 오류 코드와 설명이 문제 해결에 도움이 될 수 있습니다.

또한 응답에서 `Request ID`도 확인해야 합니다. 문제 진단을 위해 AWS Support와 협력해야 하는 경우 `Request ID`가 도움이 될 수 있습니다.

## 오류 재시도 횟수 및 지수 백오프

DNS 서버, 스위치, 로드 밸런서 등 수많은 네트워크 구성 요소는 요청이 이루어지는 모든 단계에서 오류를 일으킬 수 있습니다. 네트워크 환경에서는 클라이언트 애플리케이션의 재시도 기술이 이러한 오류 응답을 처리하는 데 가장 많이 사용되고 있습니다. 이 기술은 애플리케이션의 안정성을 개선합니다.

AWS SDK는 모두 재시도 로직을 자동으로 구현합니다. 재시도 파라미터를 필요에 따라 수정할 수 있습니다. 예를 들어 오류가 발생할 때 재시도가 허용되지 않는 fail-fast 전략을 필요로 하는 Java 애플

리케이션을 고려해 보세요. AWS SDK for Java의 경우, `ClientConfiguration` 클래스를 사용하고 `maxErrorRetry` 값으로 `0`을 입력하면 재시도가 비활성화됩니다. 자세한 내용은 해당 프로그래밍 언어의 AWS SDK 설명서를 참조하세요.

AWS SDK를 사용하지 않을 때는 서버 오류(5xx)가 수신된 최초 요청을 재시도해야 합니다. 하지만 클라이언트 오류(4xx, `ThrottlingException` 또는 `ProvisionedThroughputExceededException` 제외)는 요청 자체를 수정하여 문제를 해결해야만 재시도가 가능합니다.

간단한 재시도 방법 외에도 각각의 AWS SDK는 흐름 제어가 탁월한 지수 백오프 알고리즘을 구현합니다. 지수 백오프의 기본 개념은 오류 응답이 연이어 나올 때마다 재시도 간 대기 시간을 점진적으로 늘린다는 것입니다. 예를 들어 첫 번째 재시도 전에는 최대 50밀리초, 두 번째 재시도 전에는 최대 100밀리초, 그리고 세 번째 전에는 최대 200밀리초를 기다리는 방식입니다. 하지만 1분 후에도 요청이 실패하면 요청량이 아니라 할당 처리량을 초과하는 요청 크기가 원인일 수도 있습니다. 따라서 약 1분 정도에서 멈추도록 최대 재시도 횟수를 설정하세요. 요청이 실패하면 프로비저닝된 처리량 옵션을 살펴보는 것이 좋습니다.

> **Note**
>
> AWS SDK는 자동 재시도 로직과 지수 백오프를 구현합니다.

대부분의 지수 백오프 알고리즘은 지터(임의 지연)를 사용하여 연쇄 충돌을 방지합니다. 이 경우에는 이런 충돌을 방지하는 것이 아니므로 이 난수를 사용할 필요가 없습니다. 하지만 동시 클라이언트를 사용하는 경우에는 지터가 보다 빠른 요청 성공에 도움이 될 수 있습니다. 자세한 내용은 지수 백오프 및 지터 블로그 게시물을 참조하세요.

## 일괄 작업 및 오류 처리

DynamoDB 하위 수준 API는 읽기와 쓰기에 대한 배치 작업을 지원합니다. `BatchGetItem`은 하나 이상의 테이블에서 항목을 읽고 `BatchWriteItem`은 하나 이상의 테이블에 항목을 추가하거나 삭제합니다. 이 배치 작업은 다른 배치가 아닌 DynamoDB 작업 주위의 래퍼로 구현됩니다. 즉, `BatchGetItem`은 각 항목마다 한 번씩 `GetItem`을 일괄 방식으로 불러옵니다. 마찬가지로 `BatchWriteItem`은 각 항목마다 상황에 맞게 `DeleteItem` 또는 `PutItem`을 일괄 방식으로 불러옵니다.

배치 작업은 처리 도중 개별 요청의 오류를 허용할 수 있습니다. 예를 들어 `BatchGetItem` 요청으로 항목 5개를 읽어온다고 가정하겠습니다. 이때 기본 `GetItem` 요청 일부가 실패하더라도

전체 BatchGetItem 작업이 실패하지는 않습니다. 그러나 5개 읽기 작업 모두 실패하면 전체 BatchGetItem이 실패합니다.

배치 작업은 실패한 요청 각각에 대한 정보를 반환하므로 사용자가 문제를 진단하고 작업을 재시도할 수 있습니다. BatchGetItem의 경우 해당하는 테이블 및 기본 키가 응답의 UnprocessedKeys 값으로 반환됩니다. BatchWriteItem에서도 비슷한 정보가 UnprocessedItems로 반환됩니다.

읽기 또는 쓰기 작업이 실패할 수 있는 가장 큰 원인은 병목 현상입니다. BatchGetItem일 때는 일괄 요청에 포함된 하나 이상의 테이블에 작업을 지원할 만큼 충분한 읽기 용량이 할당되지 않은 경우입니다. 그리고 BatchWriteItem일 때는 하나 이상의 테이블에 충분한 쓰기 용량이 프로비저닝되지 않은 경우입니다.

DynamoDB가 미처리 항목을 반환하면 해당 항목에 대해 배치 작업을 재시도해야 합니다. 하지만 지수 백오프 알고리즘 사용을 강력히 권장합니다. 배치 작업을 바로 재시도하면 각 테이블의 병목 현상으로 인해 원인이 된 읽기 또는 쓰기 요청이 다시 실패할 수 있습니다. 하지만 지수 백오프를 사용하여 배치 작업을 지연시키면 일괄에 포함된 각 요청 작업이 성공할 가능성이 훨씬 높습니다.

## AWS SDK와 함께 DynamoDB 사용

다양한 프로그래밍 언어에 대해 AWS 소프트웨어 개발 키트(SDK)을 사용할 수 있습니다. 각 SDK는 개발자가 선호하는 언어로 애플리케이션을 쉽게 구축할 수 있도록 하는 API, 코드 예시 및 설명서를 제공합니다.

| SDK 설명서 | 코드 예시 |
| --- | --- |
| AWS SDK for C++ | AWS SDK for C++ 코드 예시 |
| AWS CLI | AWS CLI 코드 예시 |
| AWS SDK for Go | AWS SDK for Go 코드 예시 |
| AWS SDK for Java | AWS SDK for Java 코드 예시 |
| AWS SDK for JavaScript | AWS SDK for JavaScript 코드 예시 |
| AWS SDK for Kotlin | AWS SDK for Kotlin 코드 예시 |
| AWS SDK for .NET | AWS SDK for .NET 코드 예시 |
| AWS SDK for PHP | AWS SDK for PHP 코드 예시 |

| SDK 설명서 | 코드 예시 |
| --- | --- |
| AWS Tools for PowerShell | AWS Tools for PowerShell 코드 예시 |
| AWS SDK for Python (Boto3) | AWS SDK for Python (Boto3) 코드 예시 |
| AWS SDK for Ruby | AWS SDK for Ruby 코드 예시 |
| AWS SDK for Rust | AWS SDK for Rust 코드 예시 |
| AWS SDK for SAP ABAP | AWS SDK for SAP ABAP 코드 예시 |
| AWS SDK for Swift | AWS SDK for Swift 코드 예시 |

DynamoDB에 대한 구체적인 예는 AWS SDK를 사용한 DynamoDB용 코드 예제 섹션을 참조하세요.

가용성 예제

필요한 예제를 찾을 수 없습니까? 이 페이지 하단의 피드백 제공 링크를 사용하여 코드 예시를 요청하세요.

# 테이블, 항목, 쿼리, 스캔 및 인덱스 작업

이 섹션에는 Amazon DynamoDB에서의 테이블, 항목, 및 쿼리 작업에 대한 세부 정보를 제공합니다.

주제


- DynamoDB의 테이블 및 데이터 작업
- 글로벌 테이블 - 다중 활성, 다중 리전 복제
- DynamoDB의 항목 및 속성 작업
- DynamoDB에서 보조 인덱스를 사용하여 데이터 액세스 개선
- DynamoDB Transactions를 사용하여 복잡한 워크플로 관리
- Amazon DynamoDB를 사용하여 변경 데이터 캡처


## DynamoDB의 테이블 및 데이터 작업

이 단원에서는 AWS Command Line Interface(AWS CLI) 및 AWS SDK를 사용하여 Amazon DynamoDB에서 테이블을 생성하고, 업데이트하고, 삭제하는 방법에 대해 설명합니다.

> **Note**
>
> AWS Management Console을 사용하여 이와 동일한 작업을 수행할 수도 있습니다. 자세한 내용은 콘솔 사용 섹션을 참조하세요.

또한 이 단원에서는 DynamoDB Auto Scaling을 사용하거나 프로비저닝된 처리량을 수동으로 설정하여 처리 용량에 대한 추가 정보도 제공합니다.

주제


- DynamoDB 테이블에 대한 기본 작업
- DynamoDB에서 테이블 클래스 선택 시 고려 사항
- DynamoDB의 리소스에 태그 및 레이블 추가


### DynamoDB 테이블에 대한 기본 작업

다른 데이터베이스 시스템과 마찬가지로 Amazon DynamoDB는 데이터를 테이블에 저장합니다. 몇 가지 기본 작업을 사용하여 테이블을 관리할 수 있습니다.

주제

- 테이블 생성
- 테이블 설명
- 테이블 업데이트
- 테이블 삭제
- 삭제 보호 기능 사용
- 테이블 이름 나열
- 프로비저닝된 처리량 할당량 설명

## 테이블 생성

`CreateTable` 작업을 사용하여 Amazon DynamoDB에서 테이블을 생성합니다. 테이블을 생성하려면 다음 정보를 제공해야 합니다.

- 테이블 이름 이름은 DynamoDB 이름 지정 규칙에 맞아야 하고, 현재의 AWS 계정과 리전에서 고유해야 합니다. 예를 들어 미국 동부(버지니아 북부)에서 `People` 테이블을 생성하고 유럽(아일랜드)에서 또 하나의 `People` 테이블을 생성할 수 있습니다. 하지만 이 두 테이블은 서로 완전히 다릅니다. 자세한 내용은 Amazon DynamoDB에서 지원되는 데이터 형식 및 이름 지정 규칙 섹션을 참조하세요.
- 기본 키. 속성 한 개(파티션 키) 또는 두 개(파티션 키와 정렬 키)로 기본 키를 구성할 수 있습니다. `HASH`(파티션 키) 및 `RANGE`(정렬 키)에 대해 속성 이름과 데이터 유형, 그리고 각 속성의 역할을 입력해야 합니다. 자세한 내용은 프라이머리 키 섹션을 참조하세요.
- 처리량 설정(프로비저닝된 테이블의 경우) 프로비저닝된 모드를 사용 중인 경우 테이블의 초기 읽기 및 쓰기 처리량 설정을 지정해야 합니다. 나중에 이러한 설정을 수정하거나 DynamoDB Auto Scaling을 활성화하여 설정을 자동으로 관리할 수 있습니다. 자세한 내용은 DynamoDB 프로비저닝된 용량 모드 및 DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리(을)를 참조하세요.

예 1: 온디맨드 테이블 생성

온디맨드 모드를 사용하여 동일한 `Music` 테이블을 생성하려면

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
```

```
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode=PAY_PER_REQUEST
```

다음과 같이 `CreateTable` 작업은 이 테이블의 메타데이터를 반환합니다.

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 0,
            "ReadCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        },
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
```

```
        "CreationDateTime": 1542397468.348
    }
}
```

> **Important**
>
> 온디맨드 테이블에서 `DescribeTable`을 호출할 경우 읽기 용량 단위와 쓰기 용량 단위는 0으로 설정됩니다.

예 2: 프로비저닝된 테이블 생성

다음 AWS CLI 예제에서는 테이블(`Music`)을 생성하는 방법을 보여 줍니다. 기본 키는 `Artist`(파티션 키)와 `SongTitle`(정렬 키)로 구성되며, 각각의 데이터 형식은 `String`입니다. 이 테이블의 최대 처리량은 읽기 용량 단위 10개, 쓰기 용량 단위 5개입니다.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5
```

다음과 같이 `CreateTable` 작업은 이 테이블의 메타데이터를 반환합니다.

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
```

```
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
    }
}
```

`TableStatus` 요소는 테이블의 현재 상태(`CREATING`)를 나타냅니다. `ReadCapacityUnits` 및 `WriteCapacityUnits`에 지정한 값에 따라 테이블을 생성하는 데 시간이 걸릴 수 있습니다. 이 값이 크면 DynamoDB에서 테이블에 더 많은 리소스를 할당해야 합니다.

### 예 3: DynamoDB Standard-Infrequent Access 테이블 클래스를 사용하여 테이블 생성

DynamoDB Standard-Infrequent Access 테이블 클래스를 사용하여 동일한 `Music` 테이블을 생성하려면

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
```

```
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

다음과 같이 CreateTable 작업은 이 테이블의 메타데이터를 반환합니다.

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableClassSummary": {
            "LastUpdateDateTime": 1542397215.37,
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
```

```
    }
}
```

## 테이블 설명

테이블에 대한 세부 정보를 보려면 DescribeTable 작업을 사용합니다. 테이블 이름을 입력해야 합니다. DescribeTable 출력은 CreateTable 출력의 형식과 동일합니다. 여기에는 테이블이 생성된 타임스탬프, 해당 키 스키마, 해당 프로비저닝된 처리량 설정, 해당 예상 크기 및 보조 인덱스(있는 경우)가 포함됩니다.

> **Important**
>
> 온디맨드 테이블에서 DescribeTable을 호출할 경우 읽기 용량 단위와 쓰기 용량 단위는 0으로 설정됩니다.

Example

```
aws dynamodb describe-table --table-name Music
```

TableStatus가 CREATING에서 ACTIVE로 변경되면 테이블을 사용할 준비가 된 것입니다.

> **Note**
>
> CreateTable 요청 직후에 DescribeTable 요청을 실행하면 DynamoDB는 오류(ResourceNotFoundException)를 반환할 수 있습니다. 이는 DescribeTable이 eventually consistent query를 사용하여 당장은 테이블 메타데이터를 이용할 수 없기 때문입니다. 이때는 몇 초 기다린 후 DescribeTable 요청을 다시 실행하십시오.
> 요금을 청구하기 위해 DynamoDB 스토리지 비용에는 100바이트의 항목별 오버헤드가 포함됩니다. (자세한 내용은 DynamoDB 요금을 참조하세요.) 이렇게 항목별로 추가된 100바이트는 용량 단위 계산이나 DescribeTable 작업에 사용되지 않습니다.

## 테이블 업데이트

UpdateTable 작업을 통해 다음 중 하나를 수행할 수 있습니다.

- 테이블의 프로비저닝된 처리량 설정을 수정합니다(프로비저닝된 모드 테이블의 경우).

- 테이블의 읽기/쓰기 용량 모드를 변경합니다.
- 테이블에서 전역 보조 인덱스를 조작합니다(DynamoDB에서 글로벌 보조 인덱스 사용 참조).
- 테이블에서 DynamoDB Streams를 활성화하거나 비활성화합니다(DynamoDB Streams에 대한 변경 데이터 캡처 참조).

Example

다음 AWS CLI 예제는 테이블의 프로비저닝된 처리량 설정을 수정하는 방법을 보여줍니다.

```
aws dynamodb update-table --table-name Music \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10
```

> **Note**
>
> `UpdateTable` 요청을 실행하면 테이블 상태가 `AVAILABLE`에서 `UPDATING`으로 바뀝니다. `UPDATING` 상태라고 해도 테이블 사용에는 제한이 없습니다. 이 프로세스가 끝나면 테이블 상태가 다시 `UPDATING`에서 `AVAILABLE`로 바뀝니다.

Example

다음 AWS CLI 예제는 테이블의 읽기/쓰기 용량 모드를 온디맨드 모드로 수정하는 방법을 보여줍니다.

```
aws dynamodb update-table --table-name Music \
    --billing-mode PAY_PER_REQUEST
```

## 테이블 삭제

`DeleteTable` 작업으로 미사용 테이블을 제거할 수 있습니다. 테이블 삭제 작업은 취소할 수 없습니다. AWS Management Console을 사용하여 테이블을 삭제하려면 the section called “6단계: (선택 사항) 정리” 섹션을 참조하세요.

Example

다음 AWS CLI 예제는 대기열을 삭제하는 방법을 보여줍니다.

```
aws dynamodb delete-table --table-name Music
```

`DeleteTable` 요청을 실행하면 테이블의 상태가 `ACTIVE`에서 `DELETING`으로 바뀝니다. 사용하는 리소스(예: 테이블에 저장된 데이터, 테이블의 스트림 또는 인덱스 등)에 따라 테이블을 삭제하는 데 시간이 걸릴 수 있습니다.

`DeleteTable` 작업이 종료되면 DynamoDB에는 해당 테이블이 더 이상 존재하지 않습니다.

## 삭제 보호 기능 사용

삭제 보호 속성을 사용하여 테이블이 실수로 삭제되지 않도록 보호할 수 있습니다. 테이블에 이 속성을 활성화하면 관리자가 일반적인 테이블 관리 작업을 수행하는 동안 테이블이 실수로 삭제되는 것을 방지할 수 있습니다. 이렇게 하면 정상적인 비즈니스 운영이 중단되는 것을 방지하는 데 도움이 됩니다.

테이블 소유자 또는 권한이 부여된 관리자가 각 테이블의 삭제 보호 속성을 제어합니다. 모든 테이블의 삭제 보호 속성은 기본적으로 해제되어 있습니다. 여기에는 글로벌 복제본과 백업에서 복원된 테이블이 포함됩니다. 테이블에 대한 삭제 보호가 비활성화되면 Identity and Access Management(IAM) 정책에 의해 승인된 모든 사용자가 테이블을 삭제할 수 있습니다. 삭제 보호가 활성화된 테이블은 그 누구도 삭제할 수 없습니다.

이 설정을 변경하려면 테이블의 추가 설정으로 이동하여 삭제 보호 패널로 이동한 다음, 삭제 보호 활성화를 선택합니다.

삭제 보호 속성은 DynamoDB 콘솔, API, CLI/SDK 및 AWS CloudFormation에서 지원됩니다. `CreateTable` API는 테이블 생성 시 삭제 보호 속성을 지원하고, `UpdateTable` API는 기존 테이블의 삭제 보호 속성 변경을 지원합니다.

> **Note**
>
> - AWS 계정이 삭제되면 테이블을 포함한 해당 계정의 모든 데이터도 90일 이내에 삭제됩니다.
> - DynamoDB가 테이블을 암호화하는 데 사용된 고객 관리형 키에 액세스할 수 없는 경우에도 테이블은 여전히 아카이브됩니다. 아카이브에는 테이블을 백업하고 원본을 삭제하는 작업이 포함됩니다.

## 테이블 이름 나열

`ListTables` 작업은 현재 AWS 계정 및 리전의 DynamoDB 테이블 이름을 반환합니다.

Example

다음 AWS CLI 예제는 DynamoDB 테이블 이름을 나열하는 방법을 보여 줍니다.

```
aws dynamodb list-tables
```

### 프로비저닝된 처리량 할당량 설명

`DescribeLimits` 작업은 현재 AWS 계정 및 리전의 현재 읽기 및 쓰기 용량 할당량을 반환합니다.

Example

다음 AWS CLI 예제는 현재 프로비저닝된 처리량 할당량을 설명하는 방법을 보여줍니다.

```
aws dynamodb describe-limits
```

출력을 통해 현재 AWS 계정 및 리전에 대한 읽기 및 쓰기 용량 단위의 상한 할당량을 확인할 수 있습니다.

이러한 할당량과 할당량 증가를 요청하는 방법에 대한 자세한 내용은 처리량 기본 할당량 단원을 참조하십시오.

## DynamoDB에서 테이블 클래스 선택 시 고려 사항

DynamoDB는 비용을 최적화할 수 있도록 설계된 두 가지 테이블 클래스를 제공합니다. DynamoDB Standard 테이블 클래스가 기본값이며 대다수의 워크로드에 권장됩니다. DynamoDB Standard-Infrequent Access(DynamoDB Standard-IA) 테이블 클래스는 스토리지 비용이 많이 드는 테이블에 최적화되어 있습니다. 예를 들어 애플리케이션 로그, 이전 소셜 미디어 게시물, 전자 상거래 주문 내역 및 과거 게임 성과 같이 자주 액세스하지 않는 데이터를 저장하는 테이블은 Standard-IA 테이블 클래스에 적합합니다.

모든 DynamoDB 테이블은 테이블 클래스와 연결됩니다. 테이블과 연결된 모든 보조 인덱스는 동일한 테이블 클래스를 사용합니다. 테이블을 생성할 때 테이블 클래스를 설정하고(기본적으로 DynamoDB Standard), AWS Management Console, AWS CLI 또는 AWS SDK를 사용하여 기존 테이블의 테이블 클래스를 업데이트할 수 있습니다. DynamoDB는 또한 단일 리전 테이블(글로벌 테이블이 아닌 테이블)에 대해 AWS CloudFormation을 사용하여 테이블 클래스를 관리하는 것을 지원합니다. 각 테이블 클래스는 데이터 스토리지와 읽기 및 쓰기 요청에 대해 서로 다른 요금이 적용됩니다. 테이블의 테이블 클래스를 선택할 경우 다음 사항에 유의하세요.

- DynamoDB Standard 테이블 클래스는 DynamoDB Standard-IA보다 낮은 처리량 비용을 제공하며 처리량이 가장 중요한 테이블에서는 가장 비용 효율적인 옵션입니다.
- DynamoDB Standard-IA 테이블 클래스는 DynamoDB Standard보다 저렴한 스토리지 비용을 제공하며 스토리지 비용이 가장 큰 테이블에 가장 비용 효율적인 옵션입니다. 스토리지가 DynamoDB Standard 테이블 클래스를 사용하는 테이블의 처리량(읽기 및 쓰기) 비용의 50%를 초과하는 경우 DynamoDB Standard-IA 테이블 클래스를 사용하면 총 테이블 비용을 절감할 수 있습니다.
- DynamoDB Standard-IA 테이블은 DynamoDB Standard 테이블과 성능, 내구성 및 가용성이 동일합니다.
- DynamoDB Standard와 DynamoDB Standard-IA 테이블 클래스 간을 전환할 때 애플리케이션 코드를 변경할 필요가 없습니다. 테이블에서 사용하는 테이블 클래스와 관계없이 동일한 DynamoDB API와 서비스 엔드포인트를 사용합니다.
- DynamoDB Standard-IA 테이블은 자동 크기 조정, 온디맨드 모드, 유지 시간(TTL), 온디맨드 백업, PITR(특정 시점으로 복구) 및 글로벌 보조 인덱스와 같은 기존 DynamoDB 기능과 모두 호환됩니다.

테이블의 가장 비용 효율적인 테이블 클래스는 테이블의 예상 스토리지 및 처리량 사용 패턴에 따라 달라집니다. AWS 비용 및 사용 보고서와 AWS Cost Explorer를 사용하여 테이블의 과거 스토리지 및 처리량 비용 및 사용량을 확인할 수 있습니다. 이 기록 데이터를 사용하여 테이블에 가장 비용 효율적인 테이블 클래스를 결정할 수 있습니다. AWS 비용 및 사용 보고서와 AWS Cost Explorer 사용에 대한 자세한 내용은 AWS Billing and Cost Management 설명서를 참조하세요. 테이블 클래스 요금 세부 정보는 Amazon DynamoDB 요금을 참조하세요.

> **Note**
>
> 테이블 클래스 업데이트는 백그라운드 프로세스입니다. 테이블 클래스 업데이트 중에도 여전히 테이블에 정상적으로 액세스할 수 있습니다. 테이블 클래스를 업데이트하는 시간은 테이블 트래픽, 스토리지 크기 및 기타 관련 변수에 따라 다릅니다. 30일 후행 기간에는 테이블에 대한 테이블 클래스 업데이트가 2개 이상 허용되지 않습니다.

## DynamoDB의 리소스에 태그 및 레이블 추가

Amazon DynamoDB 리소스에 태그를 사용하여 레이블을 지정할 수 있습니다. 태그를 사용하면 용도, 소유자, 환경 또는 다른 기준 등 다양한 방식으로 리소스를 분류할 수 있습니다. 태그를 사용하면 다음이 가능합니다.

- 지정한 태그를 기반으로 리소스를 신속하게 식별합니다.

- AWS 청구서를 태그별로 구분할 수 있습니다.

> **Note**
>
> 태그가 지정된 테이블과 관련된 로컬 보조 인덱스(LSI) 및 글로벌 보조 인덱스(GSI)는 동일한 태그로 레이블이 자동 지정됩니다. 현재 DynamoDB Streams 사용량에 태그를 지정할 수 없습니다.

태그 지정은 Amazon EC2, Amazon S3, DynamoDB 등의 AWS 서비스에서 지원됩니다. 효율적으로 태그를 지정하면 특정 태그와 연결하여 서비스 전체에 대해 생성된 보고서를 통해 비용을 분석할 수 있습니다.

태그 지정을 시작하려면 다음을 수행합니다.

1. DynamoDB의 태그 지정 제한 사항를 파악합니다.
2. DynamoDB에서 리소스 태그 지정를 사용하여 태그를 생성합니다.
3. DynamoDB 태그를 사용하여 비용 할당 보고서 생성를 사용하여 활성 태그당 AWS 비용을 추적합니다.

끝으로, 최적화된 태깅 전략을 따르는 것이 좋습니다. 자세한 내용은 AWS 태깅 전략을 참조하세요.

## DynamoDB의 태그 지정 제한 사항

각 태그는 사용자가 정의하는 키와 값으로 구성됩니다. 다음과 같은 제한 사항이 있습니다.

- 각 DynamoDB 테이블은 동일한 키에 대해 한 가지 태그만을 사용합니다. 기존 태그(동일한 키)를 추가하는 경우 기존 태그 값이 새 값으로 업데이트됩니다.
- 태그 키와 값은 대소문자를 구분합니다.
- 최대 키 길이는 유니코드 문자 128자입니다.
- 최대 값 길이는 유니코드 문자 256자입니다.
- 허용되는 문자는 문자, 공백, 숫자, 특수 문자(`+ - = . _ : /`)입니다.
- 리소스당 최대 태그 수는 50개입니다.
- 테이블의 모든 태그에 지원되는 최대 크기는 10KB입니다.
- AWS 할당 태그 이름과 값에는 사용자가 할당할 수 없는 `aws:` 접두사가 자동으로 할당됩니다. AWS 할당 태그 이름은 태그 제한인 50개 또는 최대 크기 제한인 10KB에 포함되지 않습니다. 비용 할당 보고서에는 사용자가 지정한 태그 이름인 `user:` 접두사가 포함됩니다.

- 태그를 소급해서 적용할 수 없습니다.

## DynamoDB에서 리소스 태그 지정

Amazon DynamoDB 콘솔 또는 AWS Command Line Interface(AWS CLI)를 사용하여 태그를 추가, 나열, 편집 또는 삭제할 수 있습니다. 이러한 사용자 정의 태그를 활성화하면 AWS 결제 및 비용 관리 콘솔에 표시되어 비용 할당을 추적할 수 있습니다. 자세한 내용은 DynamoDB 태그를 사용하여 비용 할당 보고서 생성 섹션을 참조하세요.

일괄 편집을 위해 AWS Management Console에서 Tag Editor를 사용할 수 있습니다. 자세한 내용은 Tag Editor 작업을 참조하세요.

대신 DynamoDB API를 사용하려면 Amazon DynamoDB API 참조의 다음 작업을 참조하세요.

- TagResource
- UntagResource
- ListTagsOfResource

주제

- 태그별로 필터링할 사용 권한 설정
- 신규 또는 기존 테이블에 태그 추가(AWS Management Console)
- 신규 또는 기존 테이블에 태그 추가(AWS CLI)

### 태그별로 필터링할 사용 권한 설정

태그를 사용하여 DynamoDB 콘솔에서 테이블 목록을 필터링하려면 사용자의 정책에 다음 작업에 대한 액세스 권한이 포함되어 있는지 확인합니다.

- `tag:GetTagKeys`
- `tag:GetTagValues`

아래 단계에 따라 사용자에게 새 IAM 정책을 연결하여 이러한 작업에 액세스할 수 있습니다.

1. 관리자로 IAM 콘솔로 이동합니다.
2. 왼쪽 탐색 창에서 “정책”을 선택합니다.
3. “정책 생성”을 선택합니다.

4. 다음 정책을 JSON 편집기에 붙여넣습니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        }
    ]
}
```

5. 마법사를 완료하고 정책에 이름을 할당합니다(예: `TagKeysAndValuesReadAccess`).
6. 왼쪽 탐색 메뉴에서 “사용자”를 선택합니다.
7. 목록에서 DynamoDB 콘솔에 액세스하는 데 일반적으로 사용하는 사용자를 선택합니다.
8. “Add permissions(권한 추가)”를 선택합니다.
9. “Attach existing policies directly(기존 정책 직접 연결)”를 선택합니다.
10. 목록에서 이전에 생성한 정책을 선택합니다.
11. 마법사를 완료합니다.

### 신규 또는 기존 테이블에 태그 추가(AWS Management Console)

DynamoDB 콘솔을 사용하여 생성한 신규 테이블에 태그를 추가하거나 기존 테이블에 태그를 추가, 편집 또는 삭제할 수 있습니다.

#### 생성 시 리소스에 태그를 지정하려면(콘솔)

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 탐색 창에서 테이블을 선택한 다음 테이블 생성을 선택합니다.
3. Create DynamoDB table(DynamoDB 테이블 만들기) 페이지에서 이름과 기본 키를 입력합니다. 태그(Tags) 섹션에서 새 태그 추가(Add new tag)를 선택하고 사용하려는 태그를 입력합니다.

태그 구조에 대한 자세한 내용은 DynamoDB의 태그 지정 제한 사항 섹션을 참조하세요.

테이블 생성에 대한 자세한 내용은 DynamoDB 테이블에 대한 기본 작업 단원을 참조하세요.

기존 리소스에 태그를 지정하려면(콘솔)

https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.

1. 탐색 창에서 테이블을 선택합니다.
2. 목록에서 테이블을 선택한 다음 추가 설정(Additional settings) 탭을 선택합니다. 페이지 하단의 태그(Tags) 섹션에서 태그를 추가, 편집 또는 삭제할 수 있습니다.

신규 또는 기존 테이블에 태그 추가(AWS CLI)

다음 예제에서는 AWS CLI를 사용하여 테이블 및 인덱스 생성 시 태그를 지정하고 기존 리소스에 태그를 지정하는 방법을 보여줍니다.

생성 시 리소스에 태그를 지정하려면(AWS CLI)

- 다음 예제에서는 새 `Movies` 테이블을 생성하고 값이 `blueTeam`인 `Owner` 태그를 추가합니다.

```
aws dynamodb create-table \
    --table-name Movies \
    --attribute-definitions AttributeName=Title,AttributeType=S \
    --key-schema AttributeName=Title,KeyType=HASH \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

기존 리소스에 태그를 지정하려면(AWS CLI)

- 다음 예제에서는 `Movies` 테이블에 대해 값이 `blueTeam`인 `Owner` 태그를 추가합니다.

```
aws dynamodb tag-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies \
    --tags Key=Owner,Value=blueTeam
```

테이블에 대한 태그를 모두 나열하려면(AWS CLI)

- 다음 예제는 `Movies` 테이블과 연결된 모든 태그를 나열합니다.

```
aws dynamodb list-tags-of-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies
```

## DynamoDB 태그를 사용하여 비용 할당 보고서 생성

AWS에서는 태그를 사용하여 비용 할당 보고서에서 리소스 비용을 구성합니다. AWS에서는 두 가지 비용 할당 태그 유형을 제공합니다.

- AWS 생성 태그 AWS는 사용자를 위해 태그를 정의, 생성 및 적용합니다.
- 사용자 정의 태그로, 사용자가 태그를 정의, 생성 또는 적용합니다.

두 유형의 태그 모두 개별적으로 활성화해야만 Cost Explorer나 비용 할당 보고서에 표시됩니다.

AWS 생성 태그 활성화 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/billing/home#/에서 Billing and Cost Management 콘솔을 엽니다.
2. 탐색 창에서 비용 할당 태그를 선택합니다.
3. AWS-Generated Cost Allocation Tags(AWS 생성 비용 할당 태그)에서 Activate(활성화)를 선택합니다.

사용자 정의 태그 활성화 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/billing/home#/에서 Billing and Cost Management 콘솔을 엽니다.
2. 탐색 창에서 비용 할당 태그를 선택합니다.
3. 사용자 정의 비용 할당 태그에서 활성화를 선택합니다.

태그를 생성하여 활성화한 후 AWS에서는 사용 내역 및 비용별로 집계한 태그를 사용하여 비용 할당 보고서를 만듭니다. 비용 할당 보고서에는 각 결제 기간의 모든 AWS 비용이 포함되어 있습니다. 보고서에 태그가 지정된 리소스와 태그가 지정되지 않은 리소스가 모두 포함되어 있어 리소스에 대한 요금을 알아보기 쉽게 정리할 수 있습니다.

> **Note**
>
> 현재 DynamoDB에서 이전한 모든 데이터에 대해 비용 할당 보고서의 태그로 구분되지는 않습니다.

자세한 내용은 비용 할당 태그 사용을 참조하세요.

# 글로벌 테이블 - 다중 활성, 다중 리전 복제

Amazon DynamoDB 글로벌 테이블은 완전관리형 다중 리전 및 다중 활성 데이터베이스 기능으로, 전 세계적으로 확장되는 애플리케이션을 위해 사용하기 쉬운 데이터 복제와 빠른 로컬 읽기 및 쓰기 성능을 제공합니다.

글로벌 테이블은 자체 복제 솔루션을 구축하고 유지 관리할 필요 없이 AWS 리전 간에 DynamoDB 테이블 데이터를 자동으로 복제합니다. 테이블 복제본에 대해 AWS 리전을 선택하면 글로벌 테이블이 모든 복제를 자동으로 처리합니다. 글로벌 테이블은 DynamoDB가 제공되는 모든 리전에서 사용할 수 있습니다.

글로벌 테이블은 다음과 같은 이점을 제공합니다.

- 선택한 AWS 리전에서 DynamoDB 테이블 데이터를 자동으로 복제하여 사용자에게 더 가까운 데이터 찾기
- 리전 격리 또는 성능 저하 중에 애플리케이션 가용성 향상
- 애플리케이션의 비즈니스 로직에 집중할 수 있도록 업데이트 충돌 해결 제거

글로벌 테이블은 다중 리전 배포를 통해 비즈니스 연속성과 고가용성이 필요한 애플리케이션에 적합합니다. 모든 글로벌 테이블 복제본은 읽기 및 쓰기를 제공할 수 있습니다. 목표 복구 시점(RPO)이 낮거나 0인 애플리케이션은 리전에서 애플리케이션 처리가 중단되는 경우 트래픽을 다른 리전으로 이동하여 높은 복원력을 달성할 수 있습니다.

AWS Management Console을 사용하여 글로벌 테이블을 구성할 수 있습니다. 글로벌 테이블은 기존 DynamoDB API를 사용하여 테이블에 데이터를 읽고 쓸 수 있으므로 애플리케이션을 변경할 필요가 없습니다. 선결제 비용이나 약정 없이 프로비저닝하거나 사용한 리소스에 대해서만 비용을 지불하면 됩니다. 글로벌 테이블을 시작하려면 자습서: 글로벌 테이블 만들기 섹션을 참조하세요.

주제

- DynamoDB 글로벌 테이블 작동 방식

- 자습서: 글로벌 테이블 만들기
- DynamoDB 글로벌 테이블 보안
- 글로벌 테이블에 대한 Amazon DynamoDB 요금 청구 이해
- DynamoDB 글로벌 테이블 버전
- 글로벌 테이블에 대한 모범 사례

## DynamoDB 글로벌 테이블 작동 방식

다음 섹션에서는 Amazon DynamoDB의 글로벌 테이블 동작과 개념을 설명합니다.

### 개념

글로벌 테이블은 AWS 리전 간에 테이블 데이터를 복제하는 DynamoDB 기능입니다.

복제 테이블(또는 복제본)은 글로벌 테이블의 일부로 기능하는 DynamoDB 테이블입니다. 글로벌 테이블은 서로 다른 AWS 리전에 있는 두 개 이상의 복제본 테이블로 구성됩니다. 각 글로벌 테이블은 AWS 리전당 한 개의 복제본만 가질 수 있습니다. 글로벌 테이블의 모든 복제본은 동일한 테이블 이름, 프라이머리 키 스키마 및 항목 데이터를 공유합니다.

애플리케이션이 한 리전의 복제본에 데이터를 쓰면 DynamoDB는 글로벌 테이블의 다른 모든 복제본으로 해당 쓰기를 자동으로 복제합니다. 전역 테이블을 시작하는 방법에 대한 자세한 내용은 자습서: 글로벌 테이블 만들기를 참조하십시오.

### 버전

DynamoDB 글로벌 테이블에는 버전 2019.11.21(현재)과 버전 2017.11.29(레거시)의 두 가지 버전이 있습니다. 가능하면 버전 2019.11.21(현재)을 사용해야 합니다. 이 설명서 섹션의 정보는 버전 2019.11.21(현재)에 대한 것입니다. 자세한 내용은 글로벌 테이블 버전 확인 섹션을 참조하세요.

### 가용성

글로벌 테이블은 다중 리전 고가용성 아키텍처를 더 쉽게 구현하여 비즈니스 연속성을 개선하는 데 도움이 됩니다. 단일 AWS 리전의 워크로드가 손상된 경우, 애플리케이션 트래픽을 다른 리전으로 이동하고 동일한 글로벌 테이블의 다른 복제본 테이블에 읽기 및 쓰기를 수행할 수 있습니다.

글로벌 테이블의 각 복제본 테이블은 단일 리전 DynamoDB 테이블과 동일한 내구성과 가용성을 제공합니다. 글로벌 테이블은 단일 리전 테이블의 99.99%에 비해 99.999%의 가용성 서비스 수준 계약(SLA)을 제공합니다.

## 일관성 모델

글로벌 테이블을 만들 때 일관성 모드를 구성할 수 있습니다. 글로벌 테이블은 다중 리전 최종 일관성(MREC)과 다중 리전 강력한 일관성(MRSC)이라는 두 가지 일관성 모드를 지원합니다.

글로벌 테이블을 만들 때 일관성 모드를 지정하지 않으면 글로벌 테이블은 기본적으로 다중 리전 최종 일관성(MREC)으로 설정됩니다. 글로벌 테이블에는 서로 다른 일관성 모드로 구성된 복제본이 포함될 수 없습니다. 글로벌 테이블을 만든 후에는 일관성 모드를 변경할 수 없습니다.

### 다중 리전 최종 일관성(MREC)

다중 리전 최종 일관성(MREC)은 글로벌 테이블의 기본 일관성 모드입니다. MREC 글로벌 테이블 복제본의 항목에 대한 변경 사항은 일반적으로 1초 이내에 다른 모든 복제본에 동기식으로 복제됩니다. 드물지만 MREC 글로벌 테이블의 복제본이 격리되거나 손상된 경우, 복제본이 정상이 되면 다른 리전에 아직 복제되지 않은 모든 데이터가 복제됩니다.

여러 리전에서 동일한 항목이 동시에 수정될 경우, DynamoDB는 각 항목 단위로 내부 타임스탬프가 가장 최신인 수정 내용을 사용하여 충돌을 해결합니다. 이 방식을 '최종 쓰기 우선' 충돌 해결 방식이라고 합니다. 항목은 결국 모든 복제본에서 마지막 쓰기에 의해 만들어진 버전으로 수렴됩니다.

강력하게 일관된 읽기 작업은 항목이 읽기가 발생한 리전에서 마지막으로 업데이트된 경우 항목의 최신 버전을 반환하지만, 항목이 다른 리전에서 마지막으로 업데이트된 경우 오래된 데이터를 반환할 수 있습니다. 조건부 쓰기는 리전에 있는 항목의 버전과 비교하여 조건 표현식을 평가합니다.

기존 DynamoDB 테이블에 복제본을 추가하여 MREC 글로벌 테이블을 만듭니다. 복제본을 추가해도 기존 단일 리전 DynamoDB 테이블 또는 글로벌 테이블 복제본에는 성능 영향이 없습니다. MREC 글로벌 테이블에 복제본을 추가하여 데이터가 복제되는 리전 수를 확장하거나, 더 이상 필요하지 않은 경우 MREC 글로벌 테이블에서 복제본을 제거할 수 있습니다. MREC 글로벌 테이블에는 DynamoDB를 사용할 수 있는 모든 리전에 복제본이 있을 수 있으며, AWS 파티션에 있는 리전 수만큼의 복제본이 있을 수 있습니다.

### 다중 리전 강력한 일관성(MRSC)

글로벌 테이블을 만들 때 다중 리전 강력한 일관성(MRSC) 모드를 구성할 수 있습니다. MRSC 글로벌 테이블 복제본의 항목 변경 사항은 쓰기 작업이 성공적인 응답을 반환하기 전에 하나 이상의 다른 리전에 동기식으로 복제됩니다. 기존 단일 리전 테이블을 MRSC 글로벌 테이블로 변환할 때는 적절한 초기화 및 복제 설정을 위해 변환이 완료될 때까지 테이블이 비어 있어야 합니다.

모든 MRSC 복제본에서 강력하게 일관된 읽기 작업은 항상 항목의 최신 버전을 반환합니다. 조건부 쓰기는 항상 항목의 최신 버전을 기준으로 조건 표현식을 평가합니다.

다른 리전에서 이미 수정 중인 항목을 수정하려고 하면 `ReplicatedWriteConflictException`과 함께 쓰기 작업이 실패합니다. `ReplicatedWriteConflictException`과 함께 실패한 쓰기는 재시도할 수 있으며 항목이 다른 리전에서 더 이상 수정되고 있지 않는 경우 성공합니다.

복제본 세 개 또는 복제본 두 개와 감시자 하나로 MRSC 글로벌 테이블을 구성할 수 있습니다. 감시자는 글로벌 테이블 복제본에 작성된 데이터를 포함하는 MRSC 글로벌 테이블의 구성 요소이며, MRSC의 가용성 아키텍처를 지원하면서 전체 복제본에 대한 선택적 대안을 제공합니다. 감시자에 대한 읽기 또는 쓰기 작업은 수행할 수 없습니다. 감시자는 두 복제본과 다른 리전에 있습니다.

MRSC 글로벌 테이블을 만들 때는 복제본과 감시자 배포 모두에 대한 리전을 MRSC 테이블을 만드는 시점에 선택합니다. MRSC 글로벌 테이블에 `DescribeTable` API의 출력에서 구성된 감시자가 있는지 여부와 어느 리전에 있는지 확인할 수 있습니다. 감시자는 DynamoDB에서 소유하고 관리하며, 감시자는 구성된 리전의 AWS 계정에 표시되지 않습니다.

MRSC 글로벌 테이블은 정확히 3개의 리전에 배포되어야 합니다. 데이터가 없는 기존 DynamoDB 테이블에 복제본 1개와 감시자 또는 복제본 2개를 추가하여 MRSC 글로벌 테이블을 만듭니다. 기존 MRSC 글로벌 테이블에는 복제본을 추가할 수 없습니다. MRSC 글로벌 테이블에서 단일 복제본 또는 감시자를 삭제할 수 없습니다. MRSC 글로벌 테이블에서 복제본 2개를 삭제하거나 복제본 1개와 감시자 1개를 삭제하여 나머지 복제본을 단일 리전 DynamoDB 테이블로 변환할 수 있습니다.

MRSC 글로벌 테이블에는 다음 고려 사항이 적용됩니다.

- 단일 리전 테이블을 MRSC 글로벌 테이블로 변환할 때는 테이블이 비어 있어야 합니다. 단일 리전 테이블을 기존 항목이 있는 MRSC 글로벌 테이블로 변환하는 것은 지원되지 않습니다. 변환 프로세스 중에 테이블에 데이터가 작성되는 일이 없도록 합니다.
- MRSC 글로벌 테이블은 다음 리전 세트에서 사용할 수 있습니다.
  - 미국 리전 세트: 미국 동부(버지니아 북부), 미국 동부(오하이오), 미국 서부(오리건)
  - EU 리전 세트: 유럽(아일랜드), 유럽(런던), 유럽(파리), 유럽(프랑크푸르트)
  - 아시아 태평양 리전 세트: 아시아 태평양(도쿄), 아시아 태평양(서울) 및 아시아 태평양(오사카).
- MRSC 글로벌 테이블은 여러 리전 세트에 걸칠 수 없습니다(예: MRSC 글로벌 테이블에는 미국 및 EU 리전 세트 모두의 복제본이 포함될 수 없음).
- MRSC 글로벌 테이블에는 Time to Live(TTL)가 지원되지 않습니다.
- MRSC 글로벌 테이블에는 로컬 보조 인덱스(LSI)가 지원되지 않습니다.
- CloudWatch Contributor Insights 정보는 작업이 발생한 리전에 대해서만 보고됩니다.

## 일관성 모드 선택

다중 리전 일관성 모드를 선택하는 주요 기준은 애플리케이션이 지연 시간이 짧은 쓰기와 강력하게 일관된 읽기를 우선시하는지 아니면 글로벌 강력한 일관성을 우선시하는지 여부입니다.

MREC 글로벌 테이블은 MRSC 글로벌 테이블에 비해 쓰기 및 강력하게 일관된 읽기 지연 시간이 짧습니다. MREC 글로벌 테이블은 복제본 간의 복제 지연과 동일한 목표 복구 시점(RPO)을 지원할 수 있으며, 일반적으로 복제본 리전에 따라 몇 초 정도 소요됩니다.

다음과 같은 경우 MREC 모드를 사용해야 합니다.

- 애플리케이션이 강력하게 일관된 읽기 작업에서 반환된 오래된 데이터를 허용할 수 있습니다(다른 리전에서 데이터가 업데이트된 경우).
- 다중 리전 읽기 일관성보다 짧은 쓰기 및 강력하게 일관된 읽기 지연 시간을 우선시합니다.
- 다중 리전 고가용성 전략은 0보다 큰 RPO를 허용할 수 있습니다.

MRSC 글로벌 테이블은 MREC 글로벌 테이블에 비해 쓰기 및 강력하게 일관된 읽기 지연 시간이 깁니다. MRSC 글로벌 테이블은 0의 목표 복구 시점(RPO)을 지원합니다.

다음과 같은 경우 MRSC 모드를 사용해야 합니다.

- 여러 리전 간에 강력하게 일관된 읽기가 필요합니다.
- 짧은 쓰기 지연 시간보다 글로벌 읽기 일관성을 우선시합니다.
- 다중 리전 고가용성 전략에 0의 RPO가 필요합니다.

## 전역 테이블 모니터링

다중 리전 최종 일관성(MREC)을 위해 구성된 글로벌 테이블은 `ReplicationLatency` 지표를 CloudWatch에 게시합니다. 이 지표는 하나의 복제본 테이블에 항목이 쓰인 시점부터 해당 항목이 글로벌 테이블의 다른 복제본에 나타날 때까지 걸린 시간을 추적합니다. `ReplicationLatency`는 밀리초 단위로 표현되며 글로벌 테이블의 모든 소스 및 대상 리전 쌍마다 내보내집니다.

일반적인 `ReplicationLatency` 값은 선택한 AWS 리전 간의 거리와 워크로드 유형 및 처리량과 같은 기타 변수에 따라 달라집니다. 예를 들어 미국 서부(캘리포니아 북부)(us-west-1) 리전의 소스 복제본은 아프리카(케이프타운)(af-south-1) 리전에 비해 미국 서부(오리건)(us-west-2) 리전에 대한 `ReplicationLatency`가 낮습니다.

ReplicationLatency 값이 상승하면 한 복제본의 업데이트 내용이 다른 복제본 테이블로 시기 적절하게 전파되지 않는다는 것을 나타낼 수 있습니다. 이 경우 애플리케이션의 읽기 및 쓰기 작업을 다른 AWS 리전으로 일시적으로 리디렉션할 수 있습니다.

다중 리전 강력한 일관성(MRSC)을 위해 구성된 글로벌 테이블은 ReplicationLatency 지표를 게시하지 않습니다.

### 결함 주입 테스트

MREC 글로벌 테이블은 AWS 결함 주입 서비스(AWS FIS)와 통합되어 글로벌 테이블 워크로드에 대한 결함 주입 실험을 수행합니다. 이렇게 하면 선택한 복제본과의 복제를 일시 중지하여 시뮬레이션된 리전 격리에 대한 애플리케이션의 응답을 테스트할 수 있습니다. 자세한 내용은 글로벌 테이블 복제 일시 중지를 참조하세요.

### TTL(Time To Live)

MREC를 위해 구성된 글로벌 테이블은 Time To Live(TTL) 삭제 구성을 지원합니다. 글로벌 테이블의 모든 복제본에 대해 TTL 설정이 자동으로 동기화됩니다. TTL이 리전의 복제본에서 항목을 삭제하면 삭제가 글로벌 테이블의 다른 모든 복제본에 복제됩니다. TTL은 쓰기 용량을 소비하지 않으므로 삭제가 발생한 리전에서 TTL 삭제에 대한 요금이 부과되지 않습니다. 그러나 글로벌 테이블에 복제본이 있는 다른 리전에서 복제된 삭제에 대해서는 요금이 부과됩니다.

TTL 삭제 복제는 삭제가 복제되는 복제본의 쓰기 용량을 사용합니다. 프로비저닝된 용량에 대해 구성된 복제본은 쓰기 처리량과 TTL 삭제 처리량의 조합이 프로비저닝된 쓰기 용량보다 높은 경우 요청을 스로틀링할 수 있습니다.

다중 리전 강력한 일관성(MRSC)을 위해 구성된 글로벌 테이블은 Time To Live(TTL) 삭제 구성을 지원하지 않습니다.

### 스트림

다중 리전 최종 일관성(MREC)을 위해 구성된 글로벌 테이블은 복제본 테이블의 DynamoDB 스트림에서 변경 사항을 읽고 해당 변경 사항을 다른 모든 복제본 테이블에 적용하여 변경 사항을 복제합니다. 따라서 스트림은 MREC 글로벌 테이블의 모든 복제본에서 기본적으로 활성화되며 해당 복제본에서 비활성화할 수 없습니다. MREC 복제 프로세스는 단기간에 여러 변경 사항을 복제된 단일 쓰기로 결합하여 각 복제본의 스트림에 서로 약간 다른 레코드가 포함될 수 있습니다. MREC 복제본의 스트림 레코드는 항상 항목별로 정렬되지만, 항목 간 정렬은 복제본마다 다를 수 있습니다.

다중 리전 강력한 일관성(MRSC)에 대해 구성된 글로벌 테이블은 복제에 DynamoDB 스트림을 사용하지 않으므로 스트림은 MRSC 복제본에서 기본적으로 활성화되지 않습니다. MRSC 복제본에서 스트림

을 활성화할 수 있습니다. MRSC 복제본의 스트림 레코드는 스트림 레코드 순서를 포함하여 모든 복제본에서 동일합니다.

특정 리전에서 발생했지만 글로벌 테이블의 다른 리전에서는 발생하지 않은 변경 사항에 대해 스트림 레코드를 처리하는 애플리케이션을 구축하려면 해당 항목에 대한 변경 사항이 발생한 리전을 정의하는 각 항목에 속성을 추가할 수 있습니다. Lambda 이벤트 필터를 사용하여 특정 리전의 변경 사항에 대해서만 Lambda 함수를 간접적으로 호출하는 등, 다른 리전에서 발생한 변경 사항에 대해 스트림 레코드를 필터링하려면 이 속성을 사용합니다.

### 트랜잭션

MREC를 위해 구성된 글로벌 테이블에서 DynamoDB 트랜잭션 작업(`TransactWriteItems` 및 `TransactGetItems`)은 작업이 간접적으로 호출된 리전 내에서만 원자성이 보장됩니다. 트랜잭션 쓰기는 리전 간에 하나의 단위로 복제되지 않습니다. 즉, 트랜잭션의 일부 쓰기만 특정 시점에 다른 복제본의 읽기 작업에 의해 반환될 수 있습니다.

예를 들어 미국 동부(오하이오) 및 미국 서부(오리건) 리전에 복제본이 있는 글로벌 테이블이 있고 미국 동부(오하이오)에서 `TransactWriteItems` 작업을 수행할 경우, 변경 내용이 복제될 때 미국 서부(오리건)에서 부분적으로 완료된 트랜잭션을 관찰할 수 있습니다. 변경 내용은 소스 리전에서 커밋된 이후에만 다른 리전에 복제됩니다.

다중 리전 강력한 일관성(MRSC)을 위해 구성된 글로벌 테이블은 트랜잭션 작업을 지원하지 않으며, 이러한 작업이 MRSC 복제본에서 간접적으로 호출되는 경우 오류를 반환합니다.

### 읽기 및 쓰기 처리량

복제는 쓰기 용량을 사용합니다. 프로비저닝된 용량에 대해 구성된 복제본은 쓰기 처리량과 복제 처리량의 조합이 프로비저닝된 쓰기 용량보다 높은 경우 요청을 스로틀링할 수 있습니다. 온디맨드 모드의 쓰기 용량은 글로벌 테이블의 모든 복제본에 대해 동기화됩니다. 프로비저닝된 용량에 대해 구성된 글로벌 테이블은 복제본 간의 오토 스케일링 설정을 동기화합니다. 실제 프로비저닝된 쓰기 용량 설정은 사용된 쓰기 처리량에 따라 복제본 간에 다를 수 있습니다.

글로벌 테이블의 각 복제본에 대한 읽기 용량 설정을 독립적으로 구성할 수 있습니다. 글로벌 테이블에 복제본을 추가할 때 재정의 값이 지정되지 않은 한 소스 테이블 또는 복제본의 읽기 용량이 초기 값으로 사용됩니다.

### 설정 동기화

DynamoDB 글로벌 테이블의 설정은 테이블 동작 및 복제의 다양한 측면을 제어하는 구성 파라미터입니다. 이러한 설정은 DynamoDB 컨트롤 플레인 API를 통해 관리되며, 글로벌 테이블을 만들거나 수정

할 때 구성할 수 있습니다. 글로벌 테이블은 모든 복제본에서 특정 설정을 자동으로 동기화하여 일관성을 유지하면서 리전별 최적화에 유연성을 부여합니다. 동기화되는 설정과 동작 방식을 이해하면 글로벌 테이블을 효과적으로 구성하는 데 도움이 됩니다. 설정은 복제본 간에 동기화되는 방식에 따라 세 가지 주요 범주로 나뉩니다.

다음 설정은 글로벌 테이블의 복제본 간에 항상 동기화됩니다.

- 용량 모드(프로비저닝된 용량 또는 온디맨드)
- 테이블 프로비저닝된 쓰기 용량
- 테이블 쓰기 오토 스케일링
- 글로벌 보조 인덱스(GSI) 정의
- GSI 프로비저닝된 쓰기 용량
- GSI 쓰기 오토 스케일링
- 서버 측 암호화(SSE) 유형
- MREC 모드의 스트림 정의
- TTL(Time To Live)
- 웜 처리량
- 온디맨드 최대 쓰기 처리량

다음 설정은 복제본 간에 동기화되지만 복제본별로 재정의할 수 있습니다.

- 테이블 프로비저닝된 읽기 용량
- 테이블 읽기 오토 스케일링
- GSI 프로비저닝된 읽기 용량
- GSI 읽기 오토 스케일링
- 테이블 클래스
- 온디맨드 최대 읽기 처리량

> **Note**
>
> 다른 복제본에서 설정을 수정하면 재정의 가능한 설정 값이 변경됩니다. 예를 들어, 미국 동부(버지니아 북부) 및 미국 서부(오리건)에 복제본이 있는 MREC 글로벌 테이블이 있습니다. 미국 동부(버지니아 북부) 복제본은 프로비저닝된 읽기 처리량이 200RCU로 설정되어 있습니다. 미국 서부(오리건)의 복제본에는 프로비저닝된 읽기 처리량 재정의가 100RCU로 설정되어 있

습니다. 미국 동부(버지니아 북부) 복제본의 프로비저닝된 읽기 처리량 설정을 200RCU에서 300RCU로 업데이트하면 미국 서부(오리건)의 복제본에도 새로 프로비저닝된 읽기 처리량 값이 적용됩니다. 이렇게 하면 미국 서부(오리건) 복제본의 프로비저닝된 읽기 처리량 설정이 재정의된 값인 100RCU에서 새 값인 300RCU로 변경됩니다.

다음 설정은 복제본 간에 동기화되지 않습니다.

- 삭제 방지
- 시점 복구
- Tags
- 테이블 CloudWatch Contributor Insights 활성화
- GSI CloudWatch Contributor Insights 활성화
- Kinesis Data Streams 정의
- 리소스 정책
- MRSC 모드의 스트림 정의

다른 모든 설정은 복제본 간에 동기화되지 않습니다.

### DynamoDB Accelerator(DAX)

글로벌 테이블 복제본에 대한 쓰기는 DynamoDB Accelerator(DAX)를 우회하여 DynamoDB를 직접 업데이트합니다. 따라서 쓰기가 DAX 캐시를 업데이트하지 않으므로 DAX 캐시가 오래된 상태가 될 수 있습니다. 글로벌 테이블 복제본에 대해 구성된 DAX 캐시는 캐시 TTL이 만료될 때만 새로 고쳐집니다.

### 글로벌 테이블 관리에 대한 고려 사항

새 글로벌 테이블 복제본을 추가하는 데 사용된 테이블은 새 복제본이 만들어진 후 24시간이 경과할 때까지 삭제할 수 없습니다.

글로벌 테이블 복제본이 포함된 AWS 리전을 비활성화하면 리전이 비활성화된 후 20시간이 지나면 해당 복제본이 단일 리전 테이블로 영구적으로 변환됩니다.

## 자습서: 글로벌 테이블 만들기

이 섹션에서는 선호하는 일관성 모드에 맞게 구성된 DynamoDB 글로벌 테이블을 만들기 위한 단계별 지침을 제공합니다. 애플리케이션의 요구 사항에 따라 다중 리전 최종 일관성(MREC) 또는 다중 리전 강력한 일관성(MRSC) 모드를 선택합니다.

MREC 글로벌 테이블은 AWS 리전 간에 최종 일관성과 함께 더 짧은 쓰기 지연 시간을 제공합니다. MRSC 글로벌 테이블은 MREC보다 약간 긴 쓰기 지연 시간으로 리전 간에 강력하게 일관된 읽기를 제공합니다. 데이터 일관성, 지연 시간 및 가용성에 대한 애플리케이션의 요구 사항에 가장 적합한 일관성 모드를 선택합니다.

주제

- MREC용으로 구성된 글로벌 테이블 만들기
- MRSC용으로 구성된 글로벌 테이블 만들기

## MREC용으로 구성된 글로벌 테이블 만들기

이 섹션에서는 다중 리전 최종 일관성(MREC) 모드를 사용하여 글로벌 테이블을 만드는 방법을 보여줍니다. MREC는 글로벌 테이블의 기본 일관성 모드이며 AWS 리전 간에 비동기식 복제를 통해 지연 시간이 짧은 쓰기를 제공합니다. 한 리전의 항목에 대한 변경 사항은 일반적으로 1초 이내에 다른 모든 리전에 복제됩니다. 따라서 MREC는 낮은 쓰기 지연 시간을 우선시하고 리전마다 약간 다른 버전의 데이터를 반환할 수 있는 짧은 기간을 허용할 수 있는 애플리케이션에 이상적입니다.

DynamoDB를 사용할 수 있는 모든 AWS 리전에서 복제본이 있는 MREC 글로벌 테이블을 만들고 언제든지 복제본을 추가하거나 제거할 수 있습니다. 다음 예제에서는 여러 리전에 복제본이 있는 MREC 글로벌 테이블을 만드는 방법을 보여줍니다.

DynamoDB 콘솔을 사용하여 MREC 글로벌 테이블 만들기

다음 단계에 따라 AWS Management Console을 사용하여 전역 테이블을 생성합니다. 다음 예제에서는 미국 및 유럽의 복제본 테이블로 전역 테이블을 만듭니다.

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 이 예제에서는 탐색 모음의 리전 선택기에서 미국 동부(오하이오)를 선택합니다.
3. 콘솔 왼쪽의 탐색 창에서 테이블을 선택합니다.
4. Create Table(테이블 생성)을 선택합니다.
5. 테이블 생성 페이지에서 다음을 수행합니다.
    a. 테이블 이름에 **`Music`**을(를) 입력합니다.
    b. 파티션 키(Partition key)에 **`Artist`**를 입력합니다.
    c. 정렬 키에는 **`SongTitle`**을 입력합니다.

d. 기타 기본 설정을 유지하고 테이블 생성을 선택합니다.

   이 최신 테이블은 새로운 전역 테이블에서 첫 번째 복제본 테이블 역할을 합니다. 이는 나중에 추가하는 다른 복제본 테이블의 프로토타입입니다.

6. 테이블이 활성화되면 다음을 수행합니다.

   a. 테이블 목록에서 Music 테이블을 선택합니다.

   b. 전역 테이블 탭을 선택합니다.

   c. 복제본 생성을 선택합니다.

7. 사용 가능한 복제 리전 드롭다운 목록에서 미국 서부(오리건) us-west-2를 선택합니다.

   콘솔은 선택한 리전에 동일한 이름의 테이블이 존재하지 않는지 확인합니다. 이름이 동일한 테이블이 있는 경우 해당 리전에서 새 복제본 테이블을 생성하려면 먼저 기존 테이블을 삭제해야 합니다.

8. 복제본 생성을 선택합니다. 그러면 미국 서부(오레곤) us-west-2 리전에서 테이블 생성 프로세스가 시작됩니다.

   음악 테이블의 전역 테이블 탭 및 다른 복제본 테이블의 전역 테이블 탭은 테이블이 여러 리전에서 복제되었음을 나타냅니다.

9. 이전 단계를 반복하여 다른 리전을 추가하되 유럽(프랑크푸르트) eu-central-1을 리전으로 선택합니다.

10. 복제를 테스트하려면 다음을 수행합니다.

    a. 미국 동부(오하이오) 리전에서 AWS Management Console을 계속 사용해야 합니다.

    b. 테이블 항목 탐색을 선택합니다.

    c. 항목 생성을 선택합니다.

    d. Artist에 **`item_1`**을 입력하고 SongTitle에는 **`Song Value 1`**을 입력합니다.

    e. 항목 생성을 선택합니다.

11. 다른 리전으로 전환하여 복제를 확인합니다.

    a. 오른쪽 상단의 리전 선택기에서 유럽(프랑크푸르트)을 선택합니다.

    b. Music 테이블에 생성한 항목이 포함되어 있는지 확인합니다.

    c. 미국 서부(오리건)에 대해 확인을 반복합니다.

AWS CLI 또는 Java를 사용하여 MREC 글로벌 테이블 만들기

CLI

다음 코드 예제에서는 최종 일관성이 있는 다중 리전 복제(MREC)를 사용하여 DynamoDB 글로벌 테이블을 관리하는 방법을 보여줍니다.

- 다중 리전 복제(MREC)를 사용하여 테이블을 만듭니다.
- 복제본 테이블에서 항목을 넣고 가져옵니다.
- 복제본을 하나씩 제거합니다.
- 테이블을 삭제하여 정리합니다.

Bash 스크립트와 함께 AWS CLI사용

다중 리전 복제를 사용하여 테이블을 생성합니다.

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
```

```
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2
```

다중 리전 테이블을 설명합니다.

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

복제본 테이블에 항목을 넣습니다.

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

복제본 테이블에서 항목을 가져옵니다.

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
```

```
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

복제본을 제거합니다.

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

테이블을 삭제하여 정리합니다.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateTable

Java

다음 코드 예제에서는 다중 리전 간 복제본을 사용하여 DynamoDB 글로벌 테이블을 만들고 관리하는 방법을 보여줍니다.

- 글로벌 보조 인덱스와 DynamoDB Streams가 있는 테이블을 만듭니다.
- 다른 리전에 복제본을 추가하여 글로벌 테이블을 만듭니다.
- 글로벌 테이블에서 복제본을 제거합니다.
- 테스트 항목을 추가하여 리전 간 복제를 확인합니다.
- 글로벌 테이블 구성 및 복제본 상태를 설명합니다.

SDK for Java 2.x

AWS SDK for Java 2.x를 사용하여 글로벌 보조 인덱스와 DynamoDB Streams가 있는 테이블을 만듭니다.

```
    public static CreateTableResponse createTableWithGSI(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

        if (dynamoDbClient == null) {
```

```
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (indexName == null || indexName.trim().isEmpty()) {
            throw new IllegalArgumentException("Index name cannot be null or
empty");
        }

        try {
            LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

            CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(
                    AttributeDefinition.builder()
                        .attributeName("Artist")
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                    .indexName(indexName)
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(
```

```
Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 테이블이 활성화될 때까지 기다립니다.

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
```

```
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 글로벌 테이블을 생성하거나 확장하기 위한 복제본을 추가합니다.

```
    public static UpdateTableResponse addReplica(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final String indexName,
        final Long readCapacity) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (indexName == null || indexName.trim().isEmpty()) {
            throw new IllegalArgumentException("Index name cannot be null or
empty");
        }
        if (readCapacity == null || readCapacity <= 0) {
            throw new IllegalArgumentException("Read capacity must be a positive
number");
        }
```

```
        try {
            LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

            // Create a ReplicationGroupUpdate for adding a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .create(builder -> builder.regionName(replicaRegion.id())
                    .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                        .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
                            .readCapacityUnits(readCapacity)
                            .build())
                        .build())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 글로벌 테이블에서 복제본을 제거합니다.

```
    public static UpdateTableResponse removeReplica(
        final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {
```

```
        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }

        try {
            LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);

            // Create a ReplicationGroupUpdate for removing a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .delete(builder ->
builder.regionName(replicaRegion.id()).build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 테스트 항목을 추가함으로써 복제를 확인합니다.

```
   public static PutItemResponse putTestItem(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (artist == null || artist.trim().isEmpty()) {
           throw new IllegalArgumentException("Artist cannot be null or empty");
       }
       if (songTitle == null || songTitle.trim().isEmpty()) {
           throw new IllegalArgumentException("Song title cannot be null or
empty");
       }

       try {
           LOGGER.info("Adding test item to table: " + tableName);

           Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
           item.put(
               "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                   .s(artist)
                   .build());
           item.put(
               "SongTitle",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                   .s(songTitle)
                   .build());

           PutItemRequest putItemRequest =
               PutItemRequest.builder().tableName(tableName).item(item).build();

           PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
```

```
            LOGGER.info("Test item added successfully");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 글로벌 테이블 구성 및 복제본을 설명합니다.

```
    public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
```

```
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용한 글로벌 테이블 작업의 전체 예제입니다.

```
    public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {

        String tableName = "Music";
        String indexName = "SongTitleIndex";
        Long readCapacity = 15L;

        // Create DynamoDB client for the source region
        try (DynamoDbClient dynamoDbClient =
            DynamoDbClient.builder().region(sourceRegion).build()) {

            try {
                // Step 1: Create the initial table with GSI and streams
                LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
                createTableWithGSI(dynamoDbClient, tableName, indexName);

                // Step 2: Wait for table to become active
                LOGGER.info("Step 2: Waiting for table to become active");
                waitForTableActive(dynamoDbClient, tableName);

                // Step 3: Add replica in destination region
```

```
                LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
                addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

                // Step 4: Wait a moment for replica creation to start
                Thread.sleep(5000);

                // Step 5: Describe table to view replica information
                LOGGER.info("Step 5: Describing table to view replicas");
                describeTable(dynamoDbClient, tableName);

                // Step 6: Add a test item to verify replication
                LOGGER.info("Step 6: Adding test item to verify replication");
                putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");

                LOGGER.info("Global table setup completed successfully!");
                LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());

                // Step 7: Remove replica and clean up table
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
                throw e;
            }
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - CreateTable

- DescribeTable
- PutItem
- UpdateTable

## MRSC용으로 구성된 글로벌 테이블 만들기

이 섹션에서는 다중 리전 강력한 일관성(MRSC) 글로벌 테이블을 만드는 방법을 보여줍니다. MRSC 글로벌 테이블은 리전 간에 항목 변경 사항을 동기식으로 복제하므로 모든 복제본에서 강력하게 일관된 읽기 작업을 통해 항상 항목의 최신 버전을 반환합니다. 단일 리전 테이블을 MRSC 글로벌 테이블로 변환할 때는 테이블이 비어 있어야 합니다. 단일 리전 테이블을 기존 항목이 있는 MRSC 글로벌 테이블로 변환하는 것은 지원되지 않습니다. 변환 프로세스 중에 테이블에 데이터가 작성되는 일이 없도록 합니다.

복제본 세 개 또는 복제본 두 개와 감시자 하나로 MRSC 글로벌 테이블을 구성할 수 있습니다. MRSC 글로벌 테이블을 만들 때 복제본과 선택적 감시자가 배포되는 리전을 선택합니다. 다음 예제에서는 미국 동부(버지니아 북부) 및 미국 동부(오하이오) 리전에 복제본이 있고 미국 서부(오리건) 리전에 감시자가 있는 MRSC 글로벌 테이블을 만듭니다.

DynamoDB 콘솔을 사용하여 MRSC 글로벌 테이블 만들기

다음 단계에 따라 AWS Management Console을 사용하여 MRSC 글로벌 테이블을 만듭니다.

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 탐색 모음의 리전 선택기에서 MRSC를 갖춘 글로벌 테이블이 지원되는 리전(예: **us-east-2**)을 선택합니다.
3. 탐색 창에서 테이블을 선택합니다.
4. 테이블 생성을 선택합니다.
5. 테이블 생성 페이지에서 다음을 수행합니다.
   a. 테이블 이름에 **Music**을(를) 입력합니다.
   b. 파티션 키에 **Artist**를 입력하고 기본 문자열 유형을 유지합니다.
   c. 정렬 키에 **SongTitle**을 입력하고 기본 문자열 유형을 유지합니다.
   d. 기타 기본 설정을 유지하고 테이블 생성을 선택합니다.

      이 최신 테이블은 새로운 전역 테이블에서 첫 번째 복제본 테이블 역할을 합니다. 이는 나중에 추가하는 다른 복제본 테이블의 프로토타입입니다.

6. 테이블이 활성화될 때까지 기다린 다음 테이블 목록에서 선택합니다.
7. 글로벌 테이블 탭을 선택한 다음 복제본 생성을 선택합니다.
8. 복제본 생성 페이지에서 다음을 수행합니다.
    a. 다중 리전 일관성에서 강력한 일관성을 선택합니다.
    b. 복제 리전 1에서 **US East (N. Virginia) us-east-1**을 선택합니다.
    c. 복제 리전 2에서 **US West (Oregon) us-west-2**를 선택합니다.
    d. 미국 서부(오리건) 리전에서 감시자로 구성을 선택합니다.
    e. 복제본 생성을 선택합니다.
9. 복제본 및 감시자 만들기 프로세스가 완료될 때까지 기다립니다. 테이블을 사용할 준비가 되면 복제본 상태가 활성으로 표시됩니다.

### AWS CLI 또는 Java를 사용하여 MRSC 글로벌 테이블 만들기

시작하기 전에 IAM 위탁자에게 감시자 리전이 있는 MRSC 글로벌 테이블을 만드는 데 필요한 권한이 있는지 확인합니다. IAM 위탁자는 `CreateGlobalTableWitness`, `CreateTable` 및 `CreateTableReplica`를 간접적으로 호출할 수 있는 권한이 있어야 합니다.

다음은 미국 동부(버지니아 북부)의 복제본과 미국 서부(오리건)의 감시자 리전을 사용하여 DynamoDB 테이블(`MusicMRSC`)을 성공적으로 만들기 위한 샘플 IAM 정책입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateTable",
                "dynamodb:CreateTable",
                "dynamodb:CreateTableReplica",
                "dynamodb:CreateGlobalTableWitness"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789123:table/MusicMRSC",
                "arn:aws:dynamodb:us-east-1:123456789123:table/MusicMRSC"
            ]
```

```
        }
    ]
}
```

다음 코드 예제에서는 다중 리전 강력한 일관성(MRSC)을 갖춘 DynamoDB 글로벌 테이블을 만들고 관리하는 방법을 보여줍니다.

- 다중 리전 강력한 일관성이 있는 테이블을 만듭니다.
- MRSC 구성 및 복제본 상태를 확인합니다.
- 즉각적인 읽기를 통해 리전 간 강력한 일관성을 테스트합니다.
- MRSC 보장을 사용하여 조건부 쓰기를 수행합니다.
- MRSC 글로벌 테이블 리소스를 정리합니다.

Bash

Bash 스크립트와 함께 AWS CLI사용

다중 리전 강력한 일관성이 있는 테이블을 만듭니다.

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2

# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
```

```
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

MRSC 구성 및 복제본 상태를 확인합니다.

```
# Verify the global table configuration and MRSC setting
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

리전 간 즉각적인 읽기를 통해 강력한 일관성을 테스트합니다.

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
# No wait time needed - MRSC provides immediate consistency
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

MRSC 보장을 사용하여 조건부 쓰기를 수행합니다.

```
# Perform a conditional update from a different region
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
```

```
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

MRSC 글로벌 테이블 리소스를 정리합니다.

```
# Remove replica tables (must be done before deleting the primary table)
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용하여 MRSC 변환이 준비된 리전 테이블을 만듭니다.

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
                       .attributeType(ScalarAttributeType.S)
                       .build())
               .keySchema(
                   KeySchemaElement.builder()
                       .attributeName("Artist")
                       .keyType(KeyType.HASH)
                       .build(),
                   KeySchemaElement.builder()
                       .attributeName("SongTitle")
                       .keyType(KeyType.RANGE)
                       .build())
               .billingMode(BillingMode.PAY_PER_REQUEST)
               .build();
```

```
            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 리전 테이블을 복제본 및 감시자를 갖춘 MRSC로 변환합니다.

```
    public static UpdateTableResponse convertToMRSCWithWitness(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
```

```
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 글로벌 테이블 구성을 설명합니다.

```
   public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Describing MRSC global table: " + tableName);

           DescribeTableRequest request =
               DescribeTableRequest.builder().tableName(tableName).build();

           DescribeTableResponse response =
dynamoDbClient.describeTable(request);

           LOGGER.info("Table status: " + response.table().tableStatus());
           LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

           if (response.table().replicas() != null
               && !response.table().replicas().isEmpty()) {
               LOGGER.info("Number of replicas: " +
response.table().replicas().size());
               response.table()
                   .replicas()
                   .forEach(replica -> LOGGER.info(
                       "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
           }

           if (response.table().globalTableWitnesses() != null
               && !response.table().globalTableWitnesses().isEmpty()) {
               LOGGER.info("Number of witnesses: "
                   + response.table().globalTableWitnesses().size());
```

```
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 강력한 일관성을 확인하기 위한 테스트 항목을 추가합니다.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String album,
        final String year) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to MRSC global table: " + tableName);

            Map<String, AttributeValue> item = new HashMap<>();
            item.put("Artist", AttributeValue.builder().s(artist).build());
            item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            if (album != null && !album.trim().isEmpty()) {
                item.put("Album", AttributeValue.builder().s(album).build());
            }
            if (year != null && !year.trim().isEmpty()) {
                item.put("Year", AttributeValue.builder().n(year).build());
            }

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully with strong consistency");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 복제본에서 일관된 읽기로 항목을 읽습니다.

```
   public static GetItemResponse getItemWithConsistentRead(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (artist == null || artist.trim().isEmpty()) {
           throw new IllegalArgumentException("Artist cannot be null or empty");
       }
       if (songTitle == null || songTitle.trim().isEmpty()) {
           throw new IllegalArgumentException("Song title cannot be null or
empty");
       }

       try {
           LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);

           Map<String, AttributeValue> key = new HashMap<>();
           key.put("Artist", AttributeValue.builder().s(artist).build());
           key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

           GetItemRequest getItemRequest = GetItemRequest.builder()
               .tableName(tableName)
               .key(key)
               .consistentRead(true)
               .build();

           GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

           if (response.hasItem()) {
               LOGGER.info("Item found with strong consistency - no wait time
needed");
           } else {
               LOGGER.info("Item not found");
           }
```

```
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 보장이 있는 조건부 업데이트를 수행합니다.

```
    public static UpdateItemResponse performConditionalUpdate(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
            throw new IllegalArgumentException("Rating cannot be null or empty");
        }

        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);
```

```
            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");

            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");

            return response;

        } catch (ConditionalCheckFailedException e) {
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 복제본과 감시자가 활성화될 때까지 기다립니다.

```
   public static void waitForMRSCReplicasActive(
       final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
       throws InterruptedException {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (maxWaitTimeSeconds <= 0) {
           throw new IllegalArgumentException("Max wait time must be positive");
       }

       try {
           LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);

           final long startTime = System.currentTimeMillis();
           final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
           int backoffSeconds = 5; // Start with 5 second intervals
           final int maxBackoffSeconds = 30; // Cap at 30 seconds

           while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
               DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

               boolean allActive = true;
               StringBuilder statusReport = new StringBuilder();

               if (response.table().multiRegionConsistency() == null
                   || !MultiRegionConsistency.STRONG
                       .toString()

.equals(response.table().multiRegionConsistency().toString())) {
                   allActive = false;
                   statusReport
                       .append("MultiRegionConsistency: ")
                       .append(response.table().multiRegionConsistency())
                       .append(" ");
```

```
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }

                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }

                // Check witness status
                if (response.table().globalTableWitnesses() != null) {
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
```

```
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }

                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }

            throw DynamoDbException.builder()
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 복제본 및 감시자를 정리합니다.

```
    public static UpdateTableResponse cleanupMRSCReplicas(
        final DynamoDbClient dynamoDbClient,
```

```
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);

            // Remove replica using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .delete(DeleteReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Remove witness
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();
```

```
            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 워크플로 데모를 완료합니다.

```
    public static void demonstrateCompleteMRSCWorkflow(
        final DynamoDbClient primaryClient,
        final DynamoDbClient replicaClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion)
        throws InterruptedException {

        if (primaryClient == null) {
            throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
        }
        if (replicaClient == null) {
            throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
```

```
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

            // Step 1: Create an empty single-Region table
            LOGGER.info("Step 1: Creating empty single-Region table");
            createRegionalTable(primaryClient, tableName);

            // Use the existing GlobalTableOperations method for basic table
waiting
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 2: Convert to MRSC with replica and witness
            LOGGER.info("Step 2: Converting to MRSC with replica and witness");
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);

            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");

            // Add test item to primary region
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
```

```
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");

            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
            LOGGER.info("Waiting for replica cleanup to complete...");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }
```

```
            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

## DynamoDB 글로벌 테이블 보안

글로벌 테이블 복제본은 DynamoDB 테이블이므로 AWS Identity and Access Management(IAM) ID 정책 및 리소스 기반 정책을 포함하여 단일 리전 테이블에 대해 수행하는 복제본 액세스 제어와 동일한 방법을 사용합니다.

# 글로벌 테이블이 AWS IAM을 사용하는 방법

## 복제 서비스 연결 역할

글로벌 테이블을 처음으로 만드는 경우, Amazon DynamoDB는 AWS Identity and Access Management(IAM) 서비스 연결 역할(SLR)을 자동으로 만듭니다. 이름이 `AWSServiceRoleForDynamoDBReplication`인 SLR은 DynamoDB가 사용자를 대신하여 글로벌 테이블에 대한 교차 리전 복제를 관리하도록 허용합니다.

복제본에 리소스 기반 정책을 적용할 때는 `AWSServiceRoleForDynamoDBReplication` 정책에 정의된 SLR 위탁자에 대한 권한을 거부하지 마세요. 이렇게 하면 복제가 중단됩니다. 필요한 SLR 권한을 거부하면 영향을 받는 복제본과의 복제가 중지되고 복제본 테이블 상태가 `REPLICATION_NOT_AUTHORIZED`로 변경됩니다.

- 다중 리전 최종 일관성(MREC)으로 구성된 글로벌 테이블의 복제본이 20시간 이상 `REPLICATION_NOT_AUTHORIZED` 상태로 유지되는 경우 복제본은 단일 리전 DynamoDB 테이블로 되돌릴 수 없이 변환됩니다.
- 다중 리전 강력한 일관성(MRSC)을 위해 구성된 글로벌 테이블 복제본은 쓰기 및 강력하게 일관된 읽기 작업을 위해 `AccessDeniedException`을 반환합니다. MRSC 글로벌 테이블의 복제본이 7일 이상 `REPLICATION_NOT_AUTHORIZED` 상태를 유지하면 복제본에 영구적으로 액세스할 수 없게 되고, 쓰기 및 강력하게 일관된 읽기 작업이 오류와 함께 계속 실패합니다. 복제본 삭제와 같은 일부 관리 작업은 성공합니다.

## IAM 서비스 연결 역할 오토 스케일링

프로비저닝된 용량 모드에 대한 글로벌 테이블을 구성할 때 글로벌 테이블에 대한 오토 스케일링도 구성해야 합니다. DynamoDB 오토 스케일링은 AWS Application Auto Scaling Service를 사용하여 글로벌 테이블 복제본에서 프로비저닝된 처리량 용량을 동적으로 조정합니다. Application Auto Scaling 서비스는 `AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`이라는 서비스 연결 역할(SLR)을 만들어 프로비저닝된 테이블 용량을 관리하고 CloudWatch 경보를 만듭니다. 복제본에 리소스 기반 정책을 적용할 때는 `AWSServiceRoleForApplicationAutoScaling_DynamoDBTable` 정책에 정의된 Application Auto Scaling Service SLR 위탁자에 대한 권한을 거부하지 마세요. 이렇게 하면 오토 스케일링 기능이 중단됩니다.

서비스 연결 역할에 대한 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 사용을 참조하세요

### 글로벌 테이블 업그레이드에 필요한 권한

복제본을 만들려면 새 복제본을 추가할 테이블 또는 복제본에 대해 다음 권한이 있어야 합니다.

- dynamodb:UpdateTable

복제본을 만들려면 새 복제본이 만들어질 각 리전에서 다음 권한이 있어야 합니다.

- dynamodb:CreateTable
- dynamodb:CreateTableReplica

감시자를 만들려면 새 감시자가 만들어질 리전에서 다음 권한이 있어야 합니다.

- dynamodb:CreateGlobalTableWitness

복제본을 삭제하려면 복제본에 대해 다음 권한이 있어야 합니다.

- dynamodb:DeleteTable
- dynamodb:DeleteTableReplica

감시자를 삭제하려면 감시자에 대해 다음 권한이 있어야 합니다.

- dynamodb:DeleteGlobalTableWitness

UpdateTableReplicaAutoScaling API를 사용하여 복제본 오토 스케일링 정책을 업데이트하려면 복제본이 포함된 모든 리전에서 다음 권한이 있어야 합니다.

- application-autoscaling:DeleteScalingPolicy
- application-autoscaling:DeleteScheduledAction
- application-autoscaling:DeregisterScalableTarget
- application-autoscaling:DescribeScalableTargets
- application-autoscaling:DescribeScalingActivities
- application-autoscaling:DescribeScalingPolicies
- application-autoscaling:DescribeScheduledActions
- application-autoscaling:PutScalingPolicy

- `application-autoscaling:PutScheduledAction`
- `application-autoscaling:RegisterScalableTarget`

`UpdateTimeToLive` API를 사용하여 Time to Live(TTL) 설정을 업데이트하려면 모든 복제본에 대해 다음 권한이 있어야 합니다.

- `dynamodb:UpdateTimeToLive`

예제 IAM 정책

예: 글로벌 테이블 관리

다음 IAM 정책은 세 리전의 'users' 테이블에 대한 글로벌 테이블 복제본 및 감시자를 만들고 삭제할 수 있는 권한을 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowManagingUsersGlobalTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:CreateTableReplica",
                "dynamodb:CreateGlobalTableWitness",
                "dynamodb:DeleteTable",
                "dynamodb:DeleteTableReplica",
                "dynamodb:DeleteGlobalTableWitness",
                "dynamodb:UpdateTable",
                "dynamodb:UpdateTimeToLive"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/users",
                "arn:aws:dynamodb:us-east-2:123456789012:table/users",
                "arn:aws:dynamodb:us-west-2:123456789012:table/users"
            ]
        }
    ]
```

```
}
```

예: 와일드카드 거부 정책에서 필수 SLR 권한 제외

다음 조건의 IAM 정책은 DynamoDB 복제 SLR 및 AWS Auto Scaling SLR에 필요한 권한에 영향을 주지 않습니다. 이 조건은 의도치 않게 복제 또는 오토 스케일링을 중단하지 않도록 광범위하게 제한적인 정책에 추가할 수 있습니다.

```
"Condition": {
    "StringNotEquals": {
        "aws:PrincipalArn": [
            "arn:aws:iam::YOUR_ACCOUNT_ID:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication",
            "arn:aws:iam::YOUR_ACCOUNT_ID:role/aws-service-role/dynamodb.application-
autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable"
        ]
    }
}
```

## 글로벌 테이블이 AWS KMS를 사용하는 방법

모든 DynamoDB 테이블과 마찬가지로 글로벌 테이블 복제본은 항상 AWS Key Management Service(AWS KMS)에 저장된 암호화 키를 사용하여 저장 데이터를 암호화합니다.

글로벌 테이블의 모든 복제본은 동일한 유형의 KMS 키(AWS 소유 키, AWS 관리형 키 또는 고객 관리형 키)로 구성되어야 합니다.

> **Important**
>
> 복제본을 삭제하려면 DynamoDB에서 복제본의 암호화 키에 액세스해야 합니다. 복제본을 삭제하기 때문에 복제본을 암호화하는 데 사용되는 고객 관리형 키를 비활성화하거나 삭제하려면 먼저 복제본을 삭제하고, 나머지 복제본 중 하나의 테이블 상태가 `ACTIVE`로 변경될 때까지 기다린 다음, 키를 비활성화하거나 삭제해야 합니다.

다중 리전 최종 일관성(MREC)을 위해 구성된 글로벌 테이블의 경우 복제본을 암호화하는 데 사용되는 고객 관리형 키에 대한 DynamoDB의 액세스를 비활성화하거나 취소하면 복제본과의 복제가 중지되고 복제본 상태가 `INACCESSIBLE_ENCRYPTION_CREDENTIALS`로 변경됩니다. MREC 글로벌 테이블의

복제본이 20시간 이상 INACCESSIBLE_ENCRYPTION_CREDENTIALS 상태로 유지되는 경우 복제본은 단일 리전 DynamoDB 테이블로 되돌릴 수 없이 변환됩니다.

다중 리전 강력한 일관성(MRSC)으로 구성된 글로벌 테이블의 경우 복제본을 암호화하는 데 사용되는 고객 관리형 키에 대한 DynamoDB의 액세스를 비활성화하거나 취소하면 복제본과의 복제가 중지되고, 복제본에 대한 쓰기 또는 강력하게 일관된 읽기 수행을 시도했을 때 오류가 반환되며, 복제본 상태가 INACCESSIBLE_ENCRYPTION_CREDENTIALS로 변경됩니다. MRSC 글로벌 테이블의 복제본이 7일 이상 INACCESSIBLE_ENCRYPTION_CREDENTIALS 상태로 유지되는 경우 취소된 특정 권한에 따라 복제본이 아카이브되거나 영구적으로 액세스할 수 없게 됩니다.

# 글로벌 테이블에 대한 Amazon DynamoDB 요금 청구 이해

이 안내서에서는 글로벌 테이블에 대한 DynamoDB 청구가 어떻게 작동하는지 설명하고 글로벌 테이블 비용에 기여하는 구성 요소를 실제 예제와 함께 식별합니다.

Amazon DynamoDB 글로벌 테이블은 완전 관리형 서버리스 다중 리전 및 다중 활성 데이터베이스입니다. 글로벌 테이블은 99.999%의 가용성, 향상된 애플리케이션 복원력, 향상된 비즈니스 연속성을 제공하도록 설계되었습니다. 글로벌 테이블은 선택한 AWS 리전 간에 DynamoDB 테이블을 자동으로 복제하므로 빠른 로컬 읽기 및 쓰기 성능을 달성할 수 있습니다.

## 작동 방법

글로벌 테이블의 청구 모델은 단일 리전 DynamoDB 테이블과 다릅니다. 단일 리전 DynamoDB 테이블에 대한 쓰기 작업은 다음 단위를 사용하여 청구됩니다.

- 온디맨드 용량 모드의 쓰기 요청 단위(WRU)로, 최대 1KB의 각 쓰기에 대해 WRU 1개가 청구됩니다.
- 프로비저닝된 용량 모드를 위한 쓰기 용량 단위(WCU)로, WCU 1개는 최대 1KB에 대해 초당 쓰기 1회를 제공합니다.

기존 단일 리전 테이블에 복제본 테이블을 추가하여 전역 테이블을 만들면 해당 단일 리전 테이블이 복제본 테이블이 됩니다. 즉, 테이블에 대한 쓰기 비용을 청구하는 데 사용되는 단위도 변경됩니다. 복제본 테이블에 대한 쓰기 작업은 다음 단위를 사용하여 청구됩니다.

- 온디맨드 용량 모드의 복제된 쓰기 요청 단위(rWRU)로, 최대 1KB의 각 쓰기에 대해 복제본 테이블당 rWRU 1개가 청구됩니다.
- 프로비저닝된 용량 모드를 위한 복제된 쓰기 용량 단위(rWCU)로, 복제본 테이블당 WCU 1개는 최대 1KB에 대해 초당 쓰기 1회를 제공합니다.

글로벌 보조 인덱스(GSI)에 대한 업데이트는 GSI의 기본 테이블이 복제본 테이블인 경우에도 단일 리전 DynamoDB 테이블과 동일한 단위를 사용하여 청구됩니다. GSI에 대한 업데이트 작업은 다음 단위를 사용하여 청구됩니다.

- 온디맨드 용량 모드의 쓰기 요청 단위(WRU)로, 최대 1KB의 각 쓰기에 대해 WRU 1개가 청구됩니다.
- 프로비저닝된 용량 모드를 위한 쓰기 용량 단위(WCU)로, WCU 1개는 최대 1KB에 대해 초당 쓰기 1회를 제공합니다.

복제된 쓰기 단위(rWCU 및 rWRU)는 단일 리전 쓰기 단위(WCU 및 WRU)와 가격이 동일합니다. 데이터가 리전 간에 복제되므로 글로벌 테이블에는 교차 리전 데이터 전송 요금이 적용됩니다. 복제본 테이블이 포함된 모든 리전에서 글로벌 테이블에 대해 복제된 쓰기(rWCU 또는 rWRU) 요금이 발생합니다.

단일 리전 테이블과 복제본 테이블에서의 읽기 작업은 다음 단위를 사용합니다.

- 온디맨드 용량 모드에 대한 읽기 요청 단위(RRU)로, 최대 4KB의 강력하게 일관된 각 읽기에 대해 RRU 1개가 청구됩니다.
- 프로비저닝된 테이블에 대한 읽기 용량 단위(RCU)로, RCU 1개는 최대 4KB에 대해 초당 강력하게 일관된 읽기 1회를 제공합니다.

## 일관성 모드 및 청구

쓰기 작업에 대한 요금을 청구하는 데 사용되는 복제된 쓰기 단위(rWCU 및 rWRU)는 다중 리전 강력한 일관성(MRSC) 모드와 다중 리전 최종 일관성(MREC) 모드 모두에서 동일합니다. 감시자가 구성된 다중 리전 강력한 일관성(MRSC) 모드의 글로벌 테이블은 감시자 리전에 대한 복제 시 복제된 쓰기 단위 비용(rWCU 및 rWRU)이 발생하지 않으며, 감시자에 대한 저장 비용도 발생하지 않습니다. 감시자로의 복제를 위한 AWS 리전 간 데이터 전송 요금이 적용됩니다.

## DynamoDB 글로벌 테이블의 요금 청구 예시

수일간의 예제 시나리오를 살펴보면서 글로벌 테이블 쓰기 요청 청구가 실제로 어떻게 작동하는지 살펴보겠습니다(이 예제에서는 쓰기 요청만 고려하며, 예제에서 발생하는 테이블 복원 및 교차 리전 데이터 전송 요금은 포함하지 않음).

1일 차 - 단일 글로벌 테이블: us-west-2 리전에 Table_A라는 단일 리전 온디맨드 DynamoDB 테이블이 있습니다. Table_A에 1KB 항목 100개를 씁니다. 이러한 단일 리전 쓰기 작업의 경우 1KB 쓰기당 쓰기 요청 단위(WRU) 1의 요금이 청구됩니다. 1일 차 요금은 다음과 같습니다.

- 단일 리전 쓰기에 대해 us-west-2 리전의 WRU 100개

1일차에 요금이 청구된 총 요청 단위: 100WRU.

2일 차 - 글로벌 테이블 만들기: us-east-2 리전에서 Table_A에 대한 복제본을 추가하여 글로벌 테이블을 만듭니다. Table_A는 이제 두 개의 복제본 테이블(us-west-2 리전에 하나, us-east-2 리전에 하나)이 있는 글로벌 테이블입니다. us-west-2 리전의 복제본 테이블에 150개의 1KB 항목을 씁니다. 2일 차 요금은 다음과 같습니다.

- 복제된 쓰기에 대해 us-west-2 리전의 rWRU 150개
- 복제된 쓰기에 대해 us-east-2 리전의 rWRU 150개

2일차에 요금이 청구된 총 요청 단위: 300rWRU.

3일 차 - 글로벌 보조 인덱스 추가: 기본(복제본) 테이블의 모든 속성을 프로젝션하는 us-east-2 리전의 복제본 테이블에 글로벌 보조 인덱스(GSI)를 추가합니다. 글로벌 테이블은 us-west-2 리전의 복제본 테이블에 GSI를 자동으로 만듭니다. us-west-2 리전의 복제본 테이블에 200개의 새 1KB 레코드를 씁니다. 3일 차 요금은 다음과 같습니다.

- • 복제된 쓰기에 대해 us-west-2 리전의 rWRU 200개
- • GSI 업데이트에 대해 us-west-2 리전의 WRU 200개
- • 복제된 쓰기에 대해 us-east-2 리전의 rWRU 200개
- • GSI 업데이트에 대해 us-east-2 리전의 WRU 200개

3일 차에 요금이 청구된 총 쓰기 요청 단위: 400WRU 및 400rWRU.

총 3일 동안 500WRU(1일 차에 100WRU + 3일 차에 400WRU) 및 700rWRU(2일 차에 300rWRU + 3일 차에 400rWRU)의 쓰기 단위 요금이 청구됩니다.

요약하면 복제본 테이블 쓰기 작업은 복제본 테이블이 포함된 모든 리전에서 복제된 쓰기 단위로 요금이 청구됩니다. 글로벌 보조 인덱스가 있는 경우 GSI가 포함된 모든 리전(글로벌 테이블에서는 복제본 테이블이 포함된 모든 리전)의 GSI 업데이트에 대해 쓰기 단위 요금이 청구됩니다.

## DynamoDB 글로벌 테이블 버전

DynamoDB 글로벌 테이블에는 글로벌 테이블 버전 2019.11.21(현재)과 글로벌 테이블 버전 2017.11.29(레거시)의 두 가지 버전이 있습니다. 글로벌 테이블 버전 2019.11.21(현재)을 사용하는 것

이 좋습니다. 버전 2017.11.29(레거시)에 비해 사용이 더 쉽고, 더 많은 리전에서 지원되며, 대부분의 워크로드에 대해 비용이 저렴하기 때문입니다.

## 글로벌 테이블 버전 확인

### AWS CLI를 사용하여 버전 확인

#### 버전 2019.11.21(현재) 글로벌 테이블 복제본 식별

테이블이 글로벌 테이블 버전 2019.11.21(현재) 복제본인지 확인하려면 테이블에 대한 `describe-table` 명령을 간접적으로 호출합니다. 출력에 값이 '2019.11.21'인 `GlobalTableVersion` 속성이 포함된 경우 테이블은 버전 2019.11.21(현재) 글로벌 테이블 복제본입니다.

`describe-table`에 대한 CLI 명령의 예:

```
aws dynamodb describe-table \
--table-name users \
--region us-east-2
```

(약식) 출력에는 값이 '2019.11.21'인 `GlobalTableVersion` 속성이 포함되어 있으므로 이 테이블은 버전 2019.11.21(현재) 글로벌 테이블 복제본입니다.

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "id",
                "AttributeType": "S"
            },
            {
                "AttributeName": "name",
                "AttributeType": "S"
            }
        ],
        "TableName": "users",
        ...
        "GlobalTableVersion": "2019.11.21",
        "Replicas": [
            {
                "RegionName": "us-west-2",
                "ReplicaStatus": "ACTIVE",
            }
        ],
```

```
        ...
    }
}
```

버전 2017.11.29(레거시) 글로벌 테이블 복제본 식별

글로벌 테이블 버전 2017.11.29(레거시)는 글로벌 테이블 관리를 위한 전용 명령 세트를 사용합니다. 테이블이 글로벌 테이블 버전 2017.11.29(레거시) 복제본인지 확인하려면 테이블에 대한 `describe-global-table` 명령을 간접적으로 호출합니다. 성공적인 응답을 받으면 테이블은 버전 2017.11.29(레거시) 글로벌 테이블 복제본입니다. `describe-global-table` 명령이 `GlobalTableNotFoundException` 오류를 반환하면 테이블은 버전 2017.11.29(레거시) 복제본이 아닙니다.

`describe-global-table`에 대한 CLI 명령의 예:

```
aws dynamodb describe-global-table \
--table-name users \
--region us-east-2
```

명령이 성공적인 응답을 반환하므로 이 테이블은 버전 2017.11.29(레거시) 글로벌 테이블 복제본입니다.

```
{
    "GlobalTableDescription": {
        "ReplicationGroup": [
            {
                "RegionName": "us-west-2"
            },
            {
                "RegionName": "us-east-2"
            }
        ],
        "GlobalTableArn": "arn:aws:dynamodb::123456789012:global-table/users",
        "CreationDateTime": "2025-06-10T13:55:53.630000-04:00",
        "GlobalTableStatus": "ACTIVE",
        "GlobalTableName": "users"
    }
}
```

DynamoDB 콘솔을 사용하여 버전 확인

글로벌 테이블 복제본의 버전을 식별하려면 다음을 수행합니다.

1. https://console.aws.amazon.com/dynamodb/home에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 테이블을 선택합니다.
3. 글로벌 테이블 버전을 식별할 테이블을 선택합니다.
4. 전역 테이블 탭을 선택합니다.

   요약 섹션에는 사용 중인 글로벌 테이블의 버전이 표시됩니다.

## 레거시 버전과 현재 버전 간의 동작 차이

다음 목록은 글로벌 테이블의 레거시 버전과 현재 버전 간의 동작 차이를 설명합니다.

- 버전 2019.11.21(현재)은 버전 2017.11.29(레거시)에 비해 몇몇 DynamoDB 작업에 대한 쓰기 용량을 적게 소비합니다. 따라서 대부분의 고객에게 더 비용 효과적입니다. 이러한 DynamoDB 작업의 차이점은 다음과 같습니다.
  - 한 리전에 있는 1KB 항목에 대해 PutItem을 간접 호출하고 다른 리전에 복제할 때 2017.11.29(레거시)의 경우 리전당 2개의 rWRU가 필요하지만 2019.11.21(현재)의 경우 1개만 필요합니다.
  - 1KB 항목에 대해 UpdateItem을 간접 호출할 때 2017.11.29(레거시)의 경우 소스 리전당 2개의 rWRU와 대상 리전당 1개의 rWRU가 필요하지만 2019.11.21(현재)의 경우 소스 및 대상 리전에 모두 1개의 rWRU만 필요합니다.
  - 1KB 항목에 대해 DeleteItem을 간접 호출할 때 2017.11.29(레거시)의 경우 소스 리전당 1개의 rWRU와 대상 리전당 2개의 rWRU가 필요하지만 2019.11.21(현재)의 경우 소스 또는 대상 리전에 1개의 rWRU만 필요합니다.

  다음 표에서는 2개 리전에 있는 1KB 항목에 대한 2017.11.29(레거시) 및 2019.11.21(현재) 테이블의 rWRU 사용량을 보여줍니다.

| Operation | 2017.11.29(레거시) | 2019.11.21(현재) | 절감 |
|---|---|---|---|
| PutItem | rWRU 4개 | rWRU 2개 | 50% |
| UpdateItem | rWRU 3개 | rWRU 2개 | 33% |
| DeleteItem | rWRU 3개 | rWRU 2개 | 33% |

- 버전 2017.11.29(레거시)는 11개 AWS 리전에서만 사용할 수 있습니다. 그러나 버전 2019.11.21(현재)은 모든 AWS 리전에서 사용할 수 있습니다.

- 먼저 빈 리전 테이블 세트를 만든 다음 CreateGlobalTable API 간접 호출로 글로벌 테이블을 구성하여 버전 2017.11.29(레거시) 글로벌 테이블을 생성합니다. UpdateTable API 간접 호출로 기존 리전 테이블에 복제본을 추가하여 버전 2019.11.21(현재) 글로벌 테이블을 생성합니다.
- 버전 2017.11.29(레거시)에서는 생성 중일 때를 포함하여 새 리전에 복제본을 추가하기 전에 테이블의 모든 복제본을 비워야 합니다. 버전 2019.11.21(현재)에서는 이미 데이터가 포함된 테이블의 리전에 복제본을 추가 및 제거할 수 있습니다.
- 버전 2017.11.29(레거시)는 복제본 관리를 위해 다음과 같은 컨트롤 플레인 전용 API 세트를 사용합니다.
  - CreateGlobalTable
  - DescribeGlobalTable
  - DescribeGlobalTableSettings
  - ListGlobalTables
  - UpdateGlobalTable
  - UpdateGlobalTableSettings

  버전 2019.11.21(현재)은 DescribeTable 및 UpdateTable API를 사용하여 복제본을 관리합니다.
- 버전 2017.11.29(레거시)는 각 쓰기에 대해 2개의 DynamoDB Streams 레코드를 게시합니다. 버전 2019.11.21(현재)은 각 쓰기에 대해 1개의 DynamoDB Streams 레코드를 게시합니다.
- 버전 2017.11.29(레거시)는 `aws:rep:deleting`, `aws:rep:updateregion` 및 `aws:rep:updatetime` 속성을 채우고 업데이트합니다. 버전 2019.11.21(현재)은 이러한 속성을 채우거나 업데이트하지 않습니다.
- 버전 2017.11.29(레거시)는 복제본 간에 DynamoDB에서 TTL(Time To Live) 사용 설정을 동기화하지 않습니다. 버전 2019.11.21(현재)은 복제본 간에 TTL 설정을 동기화합니다.
- 버전 2017.11.29(레거시)는 TTL 삭제를 다른 복제본에 복제하지 않습니다. 버전 2019.11.21(현재)은 TTL 삭제를 모든 복제본에 복제합니다.
- 버전 2017.11.29(레거시)는 복제본 간에 Auto Scaling 설정을 동기화하지 않습니다. 버전 2019.11.21(현재)은 복제본 간에 Auto Scaling 설정을 동기화합니다.
- 버전 2017.11.29(레거시)는 복제본 간에 글로벌 보조 인덱스(GSI) 설정을 동기화하지 않습니다. 버전 2019.11.21(현재)은 복제본 간에 GSI 설정을 동기화합니다.
- 버전 2017.11.29(레거시)는 복제본 간에 저장 시 암호화 설정을 동기화하지 않습니다. 버전 2019.11.21(현재)은 복제본 간에 저장 시 암호화 설정을 동기화합니다.
- 버전 2017.11.29(레거시)는 `PendingReplicationCount` 지표를 게시합니다. 버전 2019.11.21(현재)은 이 지표를 게시하지 않습니다.

## 현재 버전으로 업그레이드

### 글로벌 테이블 업그레이드에 필요한 권한

버전 2019.11.21(현재)로 업그레이드하려면 복제본이 있는 모든 리전에 `dynamodb:UpdateGlobalTableversion` 권한이 있어야 합니다. DynamoDB 콘솔에 액세스하고 테이블을 보는 데 필요한 권한에 더해 이러한 권한이 필요합니다.

다음 IAM 정책은 글로벌 테이블을 버전 2019.11.21(현재)로 업그레이드할 수 있는 권한을 부여합니다.

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": "*"
        }
    ]
}
```

다음 IAM 정책은 두 개의 리전에 복제본이 있는 `Music` 글로벌 테이블만 버전 2019.11.21(현재)로 업그레이드할 수 있는 권한을 부여합니다.

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Music",
                "arn:aws:dynamodb:ap-southeast-1:123456789012:table/Music",
                "arn:aws:dynamodb:us-east-2:123456789012:table/Music"
            ]
        }
    ]
}
```

### 업그레이드 중 기대할 수 있는 사항

- 업그레이드하는 동안 모든 글로벌 테이블 복제본은 읽기 및 쓰기 트래픽을 계속 처리합니다.

- 업그레이드 프로세스에는 테이블 크기와 복제본 수에 따라 몇 분에서 몇 시간이 소요됩니다.
- 업그레이드 프로세스 중에 TableStatus의 값은 ACTIVE에서 UPDATING으로 변경됩니다. DescribeTable API를 간접적으로 호출하거나 테이블 보기를 사용하여 테이블 상태를 볼 수 있습니다.
- 테이블이 업그레이드되는 동안에는 Auto Scaling이 글로벌 테이블의 프로비저닝된 용량 설정을 조정하지 않습니다. 업그레이드 중에는 테이블을 온디맨드 용량 모드로 설정하는 것이 좋습니다.
- 업데이트 중에 프로비저닝된 용량 모드를 Auto Scaling과 함께 사용하도록 선택한 경우 업그레이드 기간 동안 예상되는 트래픽 증가를 수용하여 업그레이드 중 제한을 피하기 위해 정책의 최소 읽기 및 쓰기 처리량을 늘려야 합니다.
- 업그레이드 프로세스 중에 ReplicationLatency 지표가 지연 시간 급증을 일시적으로 보고하거나 지표 데이터 보고를 중지할 수 있습니다. 자세한 내용은 the section called “ReplicationLatency”를 참조하세요.
- 업그레이드 프로세스가 완료되면 테이블 상태가 ACTIVE로 변경됩니다.

업그레이드 전, 업그레이드 중, 업그레이드 후의 DynamoDB Streams 동작

| Operation | 복제본 리전 | 업그레이드 전 동작 | 업그레이드 중 동작 | 업그레이드 후 동작 |
|---|---|---|---|---|
| 넣기 또는 업데이트 | 소스 | 타임스탬프 채우기가 UpdateItem을 사용하여 발생합니다. | 타임스탬프 채우기가 PutItem을 사용하여 발생합니다. | 고객이 볼 수 있는 타임스탬프가 생성되지 않습니다. |
| | | 두 개의 Streams 레코드가 생성됩니다. 첫 번째 레코드에 고객이 작성한 속성이 포함됩니다. 두 번째 레코드에 aws:rep:* 속성이 포함됩니다. | 두 개의 Streams 레코드가 생성됩니다. 첫 번째 레코드에 고객이 작성한 속성이 포함됩니다. 두 번째 레코드에 aws:rep:* 속성이 포함됩니다. | 고객이 작성한 속성을 포함하는 단일 Streams 레코드가 생성됩니다. |

| Operation | 복제본 리전 | 업그레이드 전 동작 | 업그레이드 중 동작 | 업그레이드 후 동작 |
|---|---|---|---|---|
| | | 각 고객 쓰기 작업에 rWCU 2개가 사용됩니다. | 각 고객 쓰기 작업에 rWCU 2개가 사용됩니다. | 각 고객 쓰기 작업에 rWCU 1개가 사용됩니다. |
| | | `ReplicationLatency` 및 `PendingReplicationCount` 지표가 CloudWatch에 게시됩니다. | `ReplicationLatency` 및 `PendingReplicationCount` 지표가 CloudWatch에 게시됩니다. | `ReplicationLatency` 지표가 CloudWatch에 게시됩니다. |
| | 대상 | 복제가 PutItem을 사용하여 수행됩니다. | 복제가 PutItem을 사용하여 수행됩니다. | 복제가 PutItem을 사용하여 수행됩니다. |
| | | 고객이 작성한 속성과 `aws:rep:*` 속성을 모두 포함하는 단일 Streams 레코드가 생성됩니다. | 고객이 작성한 속성과 `aws:rep:*` 속성을 모두 포함하는 단일 Streams 레코드가 생성됩니다. | 고객이 작성한 속성만 포함하고 복제 속성은 포함하지 않는 단일 Streams 레코드가 생성됩니다. |
| | | 대상 리전에 항목이 있는 경우 rWCU 1개가 사용됩니다. 대상 리전에 항목이 없는 경우 rWCU 2개가 사용됩니다. | 대상 리전에 항목이 있는 경우 rWCU 1개가 사용됩니다. 대상 리전에 항목이 없는 경우 rWCU 2개가 사용됩니다. | 각 고객 쓰기 작업에 rWCU 1개가 사용됩니다. |

| Operation | 복제본 리전 | 업그레이드 전 동작 | 업그레이드 중 동작 | 업그레이드 후 동작 |
|---|---|---|---|---|
| | | ReplicationLatency 및 PendingReplicationCount 지표가 CloudWatch에 게시됩니다. | ReplicationLatency 및 PendingReplicationCount 지표가 CloudWatch에 게시됩니다. | ReplicationLatency 지표가 CloudWatch에 게시됩니다. |
| 삭제 | 소스 | DeleteItem을 사용하여 타임스탬프가 더 작은 항목을 삭제합니다. | DeleteItem을 사용하여 타임스탬프가 더 작은 항목을 삭제합니다. | DeleteItem을 사용하여 타임스탬프가 더 작은 항목을 삭제합니다. |
| | | 고객이 작성한 속성과 aws:rep:* 속성을 모두 포함하는 단일 Streams 레코드가 생성됩니다. | 고객이 작성한 속성과 aws:rep:* 속성을 모두 포함하는 단일 Streams 레코드가 생성됩니다. | 고객이 작성한 속성을 모두 포함하는 단일 Streams 레코드가 생성됩니다. |
| | | 각 고객 삭제 작업에 rWCU 1개가 사용됩니다. | 각 고객 삭제 작업에 rWCU 1개가 사용됩니다. | 각 고객 삭제 작업에 rWCU 1개가 사용됩니다. |
| | | ReplicationLatency 및 PendingReplicationCount 지표가 CloudWatch에 게시됩니다. | ReplicationLatency 및 PendingReplicationCount 지표가 CloudWatch에 게시됩니다. | ReplicationLatency 지표가 CloudWatch에 게시됩니다. |

| Operation | 복제본 리전 | 업그레이드 전 동작 | 업그레이드 중 동작 | 업그레이드 후 동작 |
|---|---|---|---|---|
| | 대상 | 2단계 삭제가 이루어집니다.<br>• 1단계에서 UpdateItem이 삭제 플래그를 설정합니다.<br>• 2단계에서 DeleteItem이 항목을 삭제합니다. | DeleteItem을 사용하여 항목을 삭제합니다. | DeleteItem을 사용하여 항목을 삭제합니다. |
| | | 두 개의 Streams 레코드가 생성됩니다. 첫 번째 레코드에 `aws:rep:deleting` 필드 변경 사항이 포함됩니다. 두 번째 레코드에 고객이 작성한 속성과 `aws:rep:*` 속성이 포함됩니다. | 고객이 작성한 속성을 모두 포함하는 단일 Stream 레코드가 생성됩니다. | 고객이 작성한 속성을 모두 포함하는 단일 Stream 레코드가 생성됩니다. |
| | | 각 고객 삭제 작업에 rWCU 2개가 사용됩니다. | 각 고객 삭제 작업에 rWCU 1개가 사용됩니다. | 각 고객 삭제 작업에 rWCU 1개가 사용됩니다. |

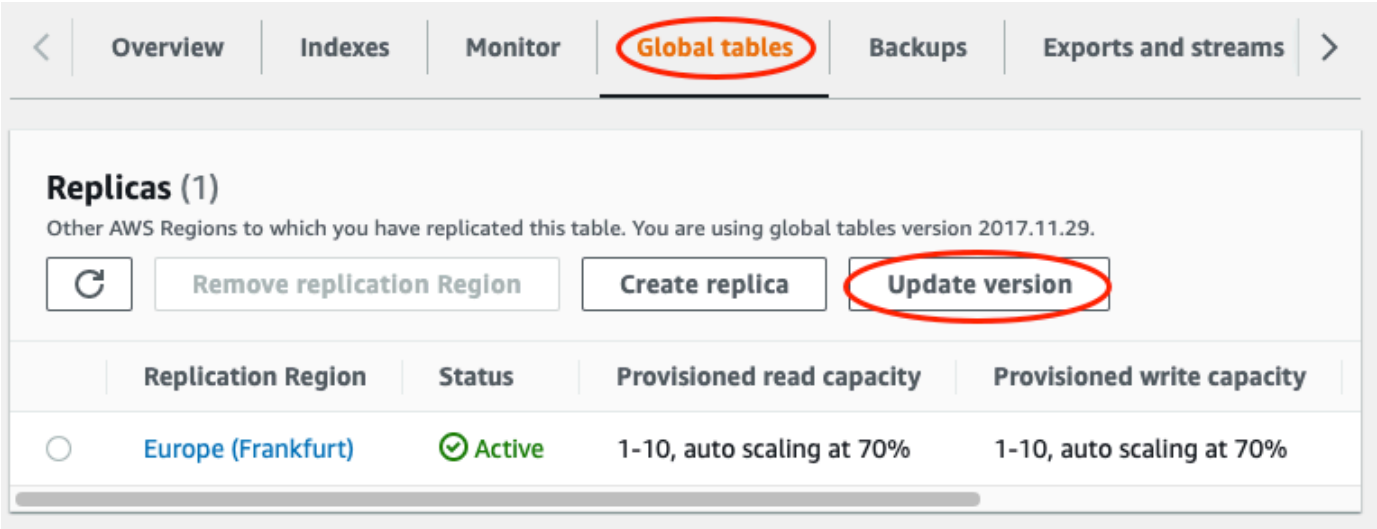
| Operation | 복제본 리전 | 업그레이드 전 동작 | 업그레이드 중 동작 | 업그레이드 후 동작 |
|---|---|---|---|---|
| | | `ReplicationLatency` 및 `PendingReplicationCount` 지표가 CloudWatch에 게시됩니다. | `ReplicationLatency` 지표가 CloudWatch에 게시됩니다. | `ReplicationLatency` 지표가 CloudWatch에 게시됩니다. |

버전 2019.11.21(현재)로 업그레이드

AWS Management Console을 사용하여 DynamoDB 글로벌 테이블의 버전을 업그레이드하려면 다음 단계를 수행하세요.

글로벌 테이블을 버전 2019.11.21(현재)로 업그레이드하는 방법

1. https://console.aws.amazon.com/dynamodb/home에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 테이블을 선택한 다음 버전 2019.11.21(현재)로 업그레이드할 글로벌 테이블을 선택합니다.
3. 전역 테이블 탭을 선택합니다.
4. Update version(버전 업데이트)을 선택합니다.




5. 새 요구 사항을 읽고 동의한 다음 버전 업데이트(Update version)를 선택합니다.

6. 업그레이드 프로세스가 완료되면 콘솔에 나타나는 글로벌 테이블 버전이 2019.11.21로 변경됩니다.

## 글로벌 테이블에 대한 모범 사례

다음 섹션에서는 글로벌 테이블 배포 및 사용에 대한 모범 사례를 설명합니다.

### 버전

DynamoDB 글로벌 테이블에는 버전 2019.11.21(현재)과 버전 2017.11.29(레거시)의 두 가지 버전이 있습니다. 가능하면 버전 2019.11.21(현재)을 사용해야 합니다.

### 삭제 방지

실수로 삭제되지 않도록 보호하려는 글로벌 테이블 복제본에서 삭제 방지를 활성화해야 합니다. 각 복제본에서 삭제 방지를 활성화해야 합니다.

### AWS CloudFormation 사용하기

AWS CloudFormation은 현재 스택 간 글로벌 테이블과 같은 다중 리전 리소스의 조정을 지원하지 않습니다. 별도의 리전 스택에서 글로벌 테이블의 각 복제본을 정의하면 복제본 업데이트를 수행할 때 스택 간에 감지된 드리프트로 인해 오류가 발생합니다. 이 문제를 방지하려면 글로벌 테이블을 배포하기 위한 참조 리전으로 하나의 리전을 선택하고 해당 리전의 스택에 모든 글로벌 테이블의 복제본을 정의해야 합니다.

> **Important**
>
> 템플릿에서 유형을 변경하여 `AWS::DynamoDB::Table` 유형의 리소스를 `AWS::DynamoDB::GlobalTable` 유형의 리소스로 변환할 수 없습니다. CloudFormation 리소스 유형을 변경하여 단일 리전 테이블을 글로벌 테이블로 변환하려고 하면 DynamoDB 테이블이 삭제될 수 있습니다.

`AWS::DynamoDB::GlobalTable` 리소스를 사용하여 단일 리전에 테이블을 만들 수 있습니다. 이 테이블은 다른 단일 리전 테이블처럼 배포됩니다. 나중에 스택을 업데이트하여 리소스에 다른 리전을 추가하면 복제본이 테이블에 추가되고 글로벌 테이블로 안전하게 변환됩니다.

`AWS::DynamoDB::GlobalTable` 리소스로 변환하려는 기존 `AWS::DynamoDB::Table` 리소스가 있는 경우 리소스 유형을 변환하는 권장 단계는 다음과 같습니다.

1. `AWS::DynamoDB::Table` 삭제 정책을 유지하도록 설정합니다.
2. 스택 정의에서 테이블을 제거합니다.
3. AWS 콘솔의 단일 리전 테이블에 복제본을 추가하여 글로벌 테이블로 변환합니다.
4. 새로운 글로벌 테이블을 새 `AWS::DynamoDB::GlobalTable` 리소스로 스택에 가져옵니다.

### 백업 및 시점 복구

글로벌 테이블의 한 복제본에 대해 자동 백업 및 시점 복구(PITR)를 활성화하면 재해 복구 목표를 충족하기에 충분할 수 있습니다. AWS-Backup으로 만들어진 복제본 백업은 복원력을 높이기 위해 리전 간에 자동으로 복제될 수 있습니다. 백업 및 PITR 활성화 전략을 선택할 때 다중 리전 고가용성의 맥락에서 재해 복구 계획 목표를 고려하세요.

### 다중 리전 고가용성을 위한 설계

글로벌 테이블 배포에 대한 권장 지침은 DynamoDB 글로벌 테이블 설계 모범 사례를 참조하세요.

## DynamoDB의 항목 및 속성 작업

Amazon DynamoDB에서 항목은 속성 모음입니다. 각 속성마다 이름과 값이 있습니다. 속성 값은 스칼라, 세트 또는 문서 유형일 수 있습니다. 자세한 내용은 Amazon DynamoDB: 작동 방식 섹션을 참조하세요.

DynamoDB는 기본 CRUD(생성/읽기/업데이트/삭제) 기능을 위한 다음 네 가지 작업을 제공합니다. 이 모든 작업은 원자성입니다.

- `PutItem` - 항목을 생성합니다.
- `GetItem` - 항목을 읽습니다.
- `UpdateItem` - 항목을 업데이트합니다.
- `DeleteItem` - 항목을 삭제합니다.

이러한 각 작업에서는 작업할 항목의 기본 키를 지정해야 합니다. 예를 들어, `GetItem`을 사용하여 항목을 읽으려면 해당 항목에 대한 파티션 키와 정렬 키(적용 가능한 경우)를 지정해야 합니다.

네 개의 기본 CRUD 작업에 추가하여 DynamoDB는 다음과 같은 작업도 제공합니다.

- `BatchGetItem` - 하나 이상의 테이블에서 최대 100개의 항목을 읽습니다.

- `BatchWriteItem` - 하나 이상의 테이블에서 최대 25개의 항목을 생성하거나 삭제합니다.

이러한 배치 작업은 여러 CRUD 작업을 단일 요청으로 결합니다. 또한 배치 작업은 항목 읽기 및 쓰기를 병렬로 실행하여 응답 지연 시간을 최소화합니다.

이 섹션에서는 이 작업을 사용하는 방법을 설명하며 조건부 업데이트 및 원자성 카운터와 같은 관련 주제도 다룹니다. 이 섹션에는 AWS SDK를 사용하는 예제 코드도 포함되어 있습니다.

주제

- DynamoDB 항목 크기 및 형식
- 항목 읽기
- 항목 쓰기
- 반환 값
- 배치 작업
- 원자성 카운터
- 조건부 쓰기
- DynamoDB에서 표현식 사용
- DynamoDB에서 TTL(Time To Live) 사용
- DynamoDB에서 테이블 쿼리
- DynamoDB에서 테이블 스캔
- PartiQL - Amazon DynamoDB용 SQL 호환 쿼리 언어
- 항목 작업: Java
- 항목 작업: .NET

## DynamoDB 항목 크기 및 형식

DynamoDB 테이블에는 기본 키를 제외하면 스키마가 없기 때문에 모든 테이블 항목의 속성, 크기, 데이터 형식이 저마다 다를 수 있습니다.

항목의 총 크기는 해당 속성 이름 및 값의 길이와 아래 설명된 대로 적용 가능한 오버헤드의 합계입니다. 다음 지침에 따라 속성 크기를 예측할 수 있습니다.

- 문자열은 UTF-8 이진수 인코딩을 사용하는 유니코드입니다. 문자열의 크기는 (속성 이름의 UTF-8로 인코딩된 바이트 수) + (UTF-8로 인코딩된 바이트 수)입니다.

- 숫자는 유효 숫자 자릿수 38자까지 길이가 다양합니다. 앞과 끝의 0은 잘립니다. 숫자의 크기는 대략 (속성 이름의 UTF-8로 인코딩된 바이트 수) + (유효 숫자 2자리당 1바이트) + (1바이트)입니다.
- 이진 값을 DynamoDB로 보내려면 base64 형식으로 인코딩해야 하지만, 크기 계산에는 이 값의 원래 바이트 길이를 사용합니다. 바이너리 속성의 크기는 (속성 이름의 UTF-8로 인코딩된 바이트 수) + (원래 바이트 수)입니다.
- null 속성 또는 부울 속성의 크기는 (속성 이름의 UTF-8로 인코딩된 바이트 수) + (1바이트)입니다.
- `List` 또는 `Map` 형식의 속성은 내용에 상관없이 오버헤드로 3바이트가 필요합니다. `List` 또는 `Map`의 크기는 (속성 이름의 UTF-8로 인코딩된 바이트 수) + 합계(중첩된 요소의 크기) + (3바이트)입니다. 빈 `List` 또는 `Map`의 크기는 (속성 이름의 UTF-8로 인코딩된 바이트 수) + (3바이트)입니다.
- 또한 각 `List` 또는`Map` 요소에는 1바이트의 오버헤드가 필요합니다.

> **Note**
>
> 속성 이름은 긴 것보다 짧은 것이 좋습니다. 이렇게 하면 필요한 스토리지 양을 줄일 수 있을 뿐 아니라 사용하는 RCU/WCU의 양도 줄일 수 있습니다.

스토리지 결제의 경우 각 아이템에 활성화한 기능에 따라 아이템당 스토리지 오버헤드가 포함됩니다.

- DynamoDB에 있는 모든 항목에는 인덱싱에 100바이트의 스토리지 오버헤드가 필요합니다.
- 일부 DynamoDB 기능(글로벌 테이블, 트랜잭션, DynamoDB를 사용한 Kinesis Data Streams 스트림에 대한 변경 데이터 캡처)은 이러한 기능을 활성화함으로써 발생하는 시스템 생성 속성을 설명하기 위해 추가 스토리지 오버헤드가 필요합니다. 예를 들어 글로벌 테이블에는 48바이트의 스토리지 오버헤드가 추가로 필요합니다.

## 항목 읽기

DynamoDB 테이블에서 항목을 읽으려면 `GetItem` 작업을 사용합니다. 원하는 항목의 기본 키와 함께 테이블 이름을 제공해야 합니다.

Example

다음은 `ProductCatalog` 테이블에서 항목을 읽는 방법을 보여주는 AWS CLI 예제입니다.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
```

```
    --key '{"Id":{"N":"1"}}'
```

> **Note**
>
> GetItem에는 부분만이 아닌 전체 기본 키를 지정해야 합니다. 예를 들어 테이블에 복합 기본 키(파티션 키 및 정렬 키)가 있는 경우 파티션 키 값과 정렬 키 값을 공급해야 합니다.

기본적으로 GetItem 요청은 최종적으로 일관된 읽기를 수행합니다. 그 대신 ConsistentRead 파라미터를 사용하여 강력한 일관된 읽기를 요청할 수 있습니다. (이 방법은 추가 읽기 용량 단위를 사용하지만 가장 최신 버전의 항목을 반환합니다.)

GetItem은 모든 항목의 속성을 반환합니다. 프로젝션 표현식을 사용하여 속성의 일부만 반환할 수 있습니다. 자세한 내용은 DynamoDB에서 프로젝션 표현식 사용 섹션을 참조하세요.

GetItem에서 사용된 읽기 용량 단위 수를 반환하려면 ReturnConsumedCapacity 파라미터를 TOTAL로 설정합니다.

Example

다음 AWS Command Line Interface(AWS CLI) 예제에는 몇 가지 GetItem 파라미터 옵션이 나와 있습니다.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --consistent-read \
    --projection-expression "Description, Price, RelatedItems" \
    --return-consumed-capacity TOTAL
```

## 항목 쓰기

DynamoDB 테이블에서 항목을 생성, 업데이트 또는 삭제하려면 다음 작업 중 하나를 사용합니다.

- PutItem
- UpdateItem
- DeleteItem

이러한 각각의 작업마다 일부가 아닌 전체 기본 키를 지정해야 합니다. 예를 들어 테이블에 복합 기본 키(파티션 키 및 정렬 키)가 있는 경우 파티션 키 값과 정렬 키 값을 제공해야 합니다.

이러한 작업 중 하나에서 사용된 쓰기 용량 단위 수를 반환하려면 `ReturnConsumedCapacity` 파라미터를 다음 중 하나로 설정합니다.

- `TOTAL` - 사용된 총 쓰기 용량 단위 수를 반환합니다.
- `INDEXES` - 사용된 총 쓰기 용량 단위 수와 작업의 영향을 받은 테이블 및 보조 인덱스의 소계를 반환합니다.
- `NONE` - 쓰기 용량 세부 정보를 반환하지 않습니다. (이 값이 기본값입니다.)

## PutItem

`PutItem` 새 항목을 만듭니다. 동일 키의 항목이 테이블이 이미 존재하는 경우 해당 항목이 새 항목으로 대체됩니다.

### Example

`Thread` 테이블에 새 항목을 씁니다. `Thread`에 대한 기본 키는 `ForumName`(파티션 키)와 `Subject`(정렬 키)로 구성됩니다.

```
aws dynamodb put-item \
    --table-name Thread \
    --item file://item.json
```

`--item`의 인수는 `item.json` 파일에 저장됩니다.

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"},
    "Message": {"S": "First post in this thread"},
    "LastPostedBy": {"S": "fred@example.com"},
    "LastPostDateTime": {"S": "201603190422"}
}
```

## UpdateItem

지정된 키가 있는 항목이 존재하지 않으면 `UpdateItem`은 새 항목을 생성합니다. 그렇지 않으면 기존 항목의 속성을 수정합니다.

업데이트 표현식을 사용하여 수정하려는 속성과 새 값을 지정합니다. 자세한 내용은 DynamoDB에서 업데이트 표현식 사용 섹션을 참조하세요.

업데이트 표현식 내에서는 표현식 속성 값을 실제 값에 대한 자리 표시자로 사용합니다. 자세한 내용은 DynamoDB에서 표현식 속성 값 사용 섹션을 참조하세요.

Example

Thread 항목에서 다양한 속성을 수정합니다. 선택적 ReturnValues 파라미터는 업데이트 후 나타나는 항목을 표시합니다. 자세한 내용은 반환 값 섹션을 참조하세요.

```
aws dynamodb update-item \
    --table-name Thread \
    --key file://key.json \
    --update-expression "SET Answered = :zero, Replies = :zero, LastPostedBy
 = :lastpostedby" \
    --expression-attribute-values file://expression-attribute-values.json \
    --return-values ALL_NEW
```

--key의 인수는 key.json 파일에 저장됩니다.

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"}
}
```

--expression-attribute-values의 인수는 expression-attribute-values.json 파일에 저장됩니다.

```
{
    ":zero": {"N":"0"},
    ":lastpostedby": {"S":"barney@example.com"}
}
```

## DeleteItem

DeleteItem는 지정된 키가 있는 항목을 삭제합니다.

Example

다음 AWS CLI 예제는 Thread 항목을 삭제하는 방법을 보여줍니다.

```
aws dynamodb delete-item \
    --table-name Thread \
    --key file://key.json
```

## 반환 값

경우에 따라서는 DynamoDB에서 수정 전 또는 후에 나타난 특정 속성 값을 반환하도록 할 수 있습니다. PutItem, UpdateItem 및 DeleteItem 작업에는 수정 전 또는 후의 속성 값을 반환하기 위해 사용할 수 있는 ReturnValues 파라미터가 있습니다.

ReturnValues의 기본값은 DynamoDB에서 수정된 속성에 대한 정보를 반환하지 않는 NONE입니다.

다음은 DynamoDB API 작업에서 구성되는 ReturnValues에 대한 기타 유효한 설정입니다.

### PutItem

- ReturnValues: ALL_OLD
  - 기존 항목을 덮어쓰면 ALL_OLD는 덮어쓰기 전에 나타난 전체 항목을 반환합니다.
  - 존재하지 않는 항목을 쓰면 ALL_OLD는 효과를 나타내지 않습니다.

### UpdateItem

UpdateItem의 가장 일반적인 용도는 기존 항목을 업데이트하는 것입니다. 하지만 실제로 UpdateItem은 항목이 아직 없는 경우 항목을 자동으로 생성하는 upsert를 수행합니다.

- ReturnValues: ALL_OLD
  - 기존 항목을 업데이트하면 ALL_OLD는 업데이트 전에 나타난 전체 항목을 반환합니다.
  - 존재하지 않는 항목을 업데이트하면(upsert) ALL_OLD는 효과를 나타내지 않습니다.
- ReturnValues: ALL_NEW
  - 기존 항목을 업데이트하면 ALL_NEW는 업데이트 후에 나타난 전체 항목을 반환합니다.
  - 존재하지 않는 항목을 업데이트하면(upsert) ALL_NEW는 전체 항목을 반환합니다.
- ReturnValues: UPDATED_OLD
  - 기존 항목을 업데이트하면 UPDATED_OLD는 업데이트 전에 나타난 업데이트된 속성만 반환합니다.
  - 존재하지 않는 항목을 업데이트하면(upsert) UPDATED_OLD는 효과를 나타내지 않습니다.
- ReturnValues: UPDATED_NEW

- 기존 항목을 업데이트하면 UPDATED_NEW는 업데이트 후에 나타난 영향을 받은 속성만 반환합니다.
- 존재하지 않는 항목을 업데이트하면(upsert) UPDATED_NEW는 업데이트 후에 나타나는 업데이트된 속성만 반환합니다.

### DeleteItem

- ReturnValues: ALL_OLD
  - 기존 항목을 삭제하면 ALL_OLD는 삭제하기 전에 나타난 전체 항목을 반환합니다.
  - 존재하지 않는 항목을 삭제하면 ALL_OLD는 데이터를 반환하지 않습니다.

## 배치 작업

여러 항목을 읽거나 써야 하는 애플리케이션의 경우 DynamoDB는 BatchGetItem 및 BatchWriteItem 작업을 제공합니다. 이러한 작업을 사용하면 애플리케이션과 DynamoDB 간의 네트워크 왕복 수가 감소될 수 있습니다. 또한 DynamoDB는 개별 읽기 또는 쓰기 작업을 병렬로 수행합니다. 애플리케이션은 동시성 또는 스레딩을 관리할 필요 없이 이 병렬 실행에서 이익을 얻을 수 있습니다.

배치 작업은 본질적으로 다중 읽기 또는 쓰기 요청에 대한 래퍼입니다. 예를 들어, BatchGetItem 요청에 5개 항목이 포함되어 있으면 DynamoDB는 사용자를 대신하여 5개의 GetItem 작업을 수행합니다. 마찬가지로, BatchWriteItem 요청에 추가 요청 두 개와 삭제 요청 네 개가 포함되어 있으면 DynamoDB는 PutItem 요청 두 개와 DeleteItem 요청 네 개를 수행합니다.

일반적으로 배치에 있는 모든 요청이 실패하지 않는 한 배치 작업은 실패하지 않습니다. 예를 들어 BatchGetItem 작업을 수행하지만 배치에 있는 개별 GetItem 요청 중 하나가 실패한다고 가정합니다. 이 경우 BatchGetItem은 실패한 GetItem 요청에서 키와 데이터를 반환합니다. 배치에 있는 기타 GetItem 요청은 영향을 받지 않습니다.

### BatchGetItem

단일 BatchGetItem 작업은 최대 100개의 개별 GetItem 요청을 포함할 수 있으며, 최대 16MB의 데이터를 검색할 수 있습니다. 또한 BatchGetItem 작업은 여러 테이블에서 항목을 검색할 수 있습니다.

#### Example

일부 속성만 반환하는 프로젝션 표현식을 사용하여 Thread 테이블에서 두 개의 항목을 검색합니다.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json
```

`--request-items`의 인수는 `request-items.json` 파일에 저장됩니다.

```
{
    "Thread": {
        "Keys": [
            {
                "ForumName":{"S": "Amazon DynamoDB"},
                "Subject":{"S": "DynamoDB Thread 1"}
            },
            {
                "ForumName":{"S": "Amazon S3"},
                "Subject":{"S": "S3 Thread 1"}
            }
        ],
        "ProjectionExpression":"ForumName, Subject, LastPostedDateTime, Replies"
    }
}
```

## BatchWriteItem

`BatchWriteItem` 작업은 최대 25개의 개별 `PutItem` 및 `DeleteItem` 요청을 포함할 수 있으며, 최대 16MB의 데이터를 쓸 수 있습니다. (개별 항목의 최대 크기는 400KB입니다.) 또한 `BatchWriteItem` 작업은 여러 테이블에서 항목을 추가하거나 삭제할 수 있습니다.

> **Note**
>
> `BatchWriteItem`은 `UpdateItem` 요청을 지원하지 않습니다.

### Example

`ProductCatalog` 테이블에 두 개의 항목을 추가합니다.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json
```

`--request-items`의 인수는 `request-items.json` 파일에 저장됩니다.

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "601" },
                    "Description": { "S": "Snowboard" },
                    "QuantityOnHand": { "N": "5" },
                    "Price": { "N": "100" }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "602" },
                    "Description": { "S": "Snow shovel" }
                }
            }
        }
    ]
}
```

## 원자성 카운터

UpdateItem 작업을 사용하여 원자성 카운터를 구현할 수 있습니다. 이는 다른 쓰기 요청과 충돌하지 않고 조건 없이 증감되는 숫자 속성입니다. 모든 쓰기 요청은 수신된 순서대로 적용됩니다. 원자성 카운터가 있으면 업데이트는 idempotent 방식이 아닙니다. 다시 말해서, UpdateItem을 호출할 때마다 숫자 값이 증가하거나 감소합니다. 원자성 카운터를 업데이트하는 데 사용되는 증분 값이 양수이면 수가 많게 계산될 수 있습니다. 증분 값이 음수이면 수가 적게 계산될 수 있습니다.

원자성 카운터를 사용하여 웹 사이트의 방문객 수를 계속 추적할 수 있습니다. 이 경우 애플리케이션은 현재 값과 상관없이 숫자 값을 계속 증가시킵니다. UpdateItem 작업이 실패할 경우 애플리케이션은 작업을 단순히 재시도할 수 있습니다. 이렇게 하면 카운터를 두 번 업데이트할 위험이 있지만, 웹 사이트 방문객 수를 약간 많게 또는 적게 계산하는 것을 허용할 수 있습니다.

수를 많게 또는 적게 계산하는 것을 허용할 수 없는 경우에는(예: 은행 애플리케이션) 원자성 카운터가 적합하지 않습니다. 이러한 경우 원자성 카운터 대신 조건부 업데이트를 사용하는 것이 더 안전합니다.

자세한 내용은 수치 속성 증가 및 감소 섹션을 참조하세요.

Example

다음은 제품의 Price가 5씩 증가하는 AWS CLI 예입니다. 이 예에서는 카운터가 업데이트되기 전에 항목이 존재하는 것으로 알려져 있습니다. UpdateItem은 멱등성이 없으므로 이 코드를 실행할 때마다 Price가 증가합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": { "N": "601" }}' \
    --update-expression "SET Price = Price + :incr" \
    --expression-attribute-values '{":incr":{"N":"5"}}' \
    --return-values UPDATED_NEW
```

## 조건부 쓰기

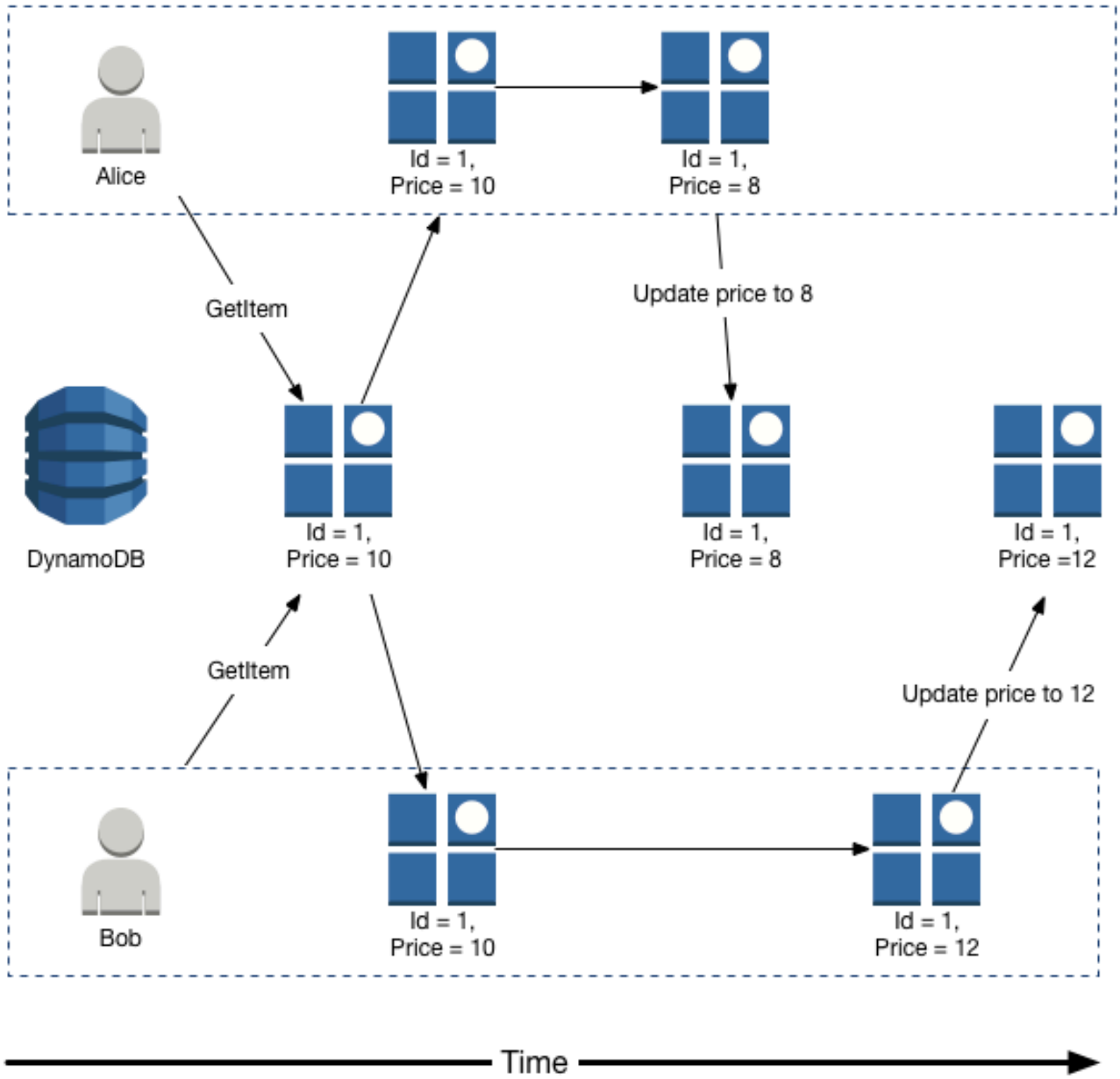
기본적으로 DynamoDB 쓰기 작업(PutItem, UpdateItem, DeleteItem)은 무조건적입니다. 즉, 이러한 각 작업은 지정된 기본 키가 있는 기존 항목을 덮어씁니다.

DynamoDB는 선택적으로 이러한 작업의 조건부 쓰기를 지원합니다. 조건부 쓰기는 항목 속성이 하나 이상의 예상 조건과 일치하는 경우에만 성공합니다. 그렇지 않으면 오류를 반환합니다.

조건부 쓰기는 가장 최근에 업데이트된 항목 버전과 비교하여 조건을 확인합니다. 항목이 이전에 존재하지 않았거나 해당 항목에 대해 가장 최근에 성공한 작업이 삭제인 경우 조건부 쓰기는 이전 항목을 찾지 못한다는 점에 유의하시기 바랍니다.

조건부 쓰기는 많은 상황에서 유용합니다. 예를 들면 동일한 기본 키가 있는 항목이 아직 없는 경우에만 PutItem 작업이 성공하도록 하려고 합니다. 또는 속성 중 하나에 특정 값이 있으면 UpdateItem 작업을 통해 항목을 수정할 수 없도록 할 수 있습니다.

조건부 쓰기는 여러 사용자가 동일한 항목을 수정하려고 시도하는 경우에 유용합니다. 두 명의 사용자(Alice와 Bob)가 DynamoDB 테이블에서 동일한 항목을 작업하고 있는 다음 다이어그램을 생각해 봅니다.




Alice가 AWS CLI를 사용하여 `Price` 속성을 8로 업데이트한다고 가정합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
```

```
    --expression-attribute-values file://expression-attribute-values.json
```

`--expression-attribute-values`의 인수는 `expression-attribute-values.json` 파일에 저장됩니다.

```
{
    ":newval":{"N":"8"}
}
```

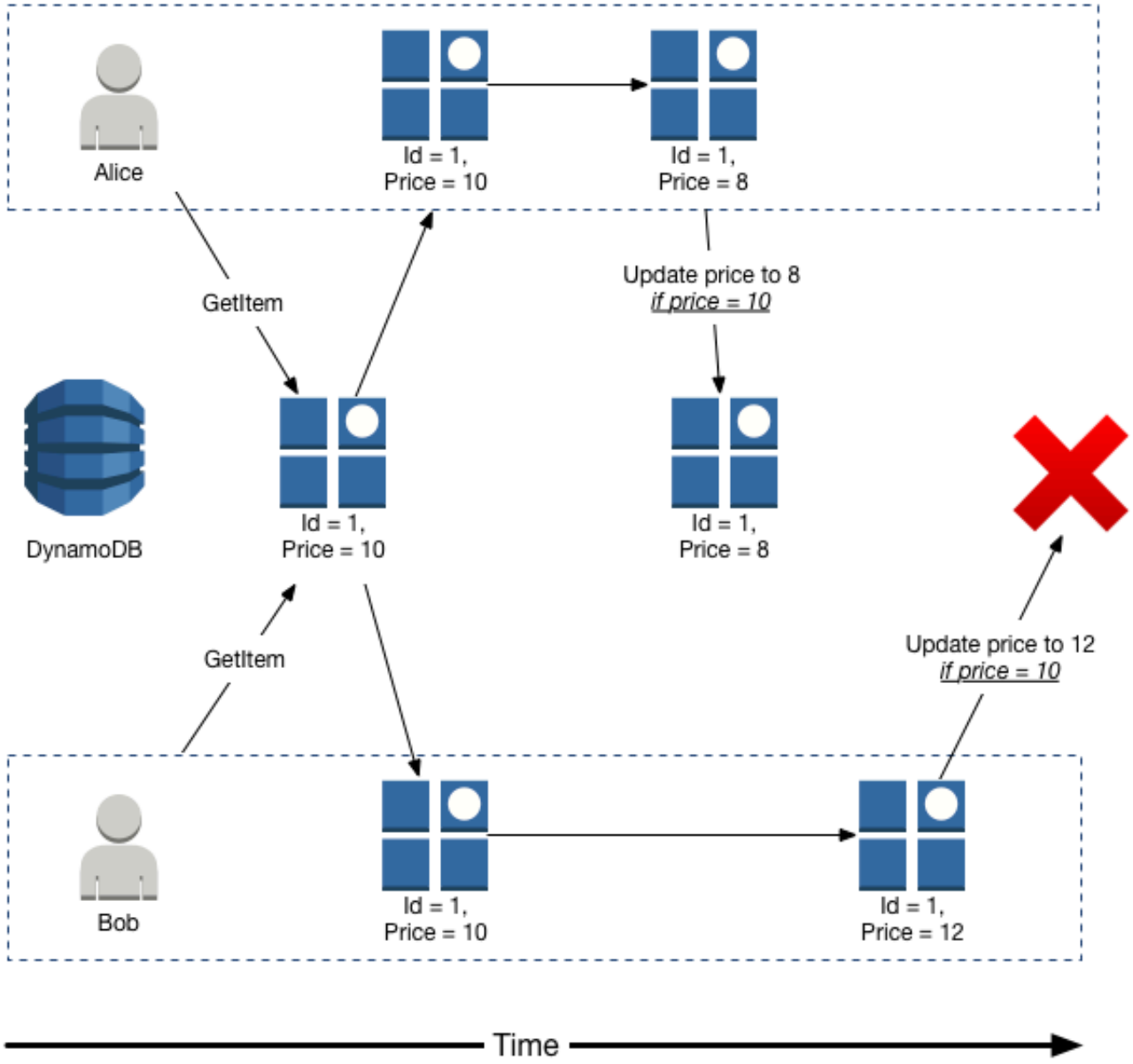
이제 Bob이 비슷한 `UpdateItem` 요청을 나중에 실행하지만, `Price`를 12로 변경한다고 가정합니다. Bob의 경우 `--expression-attribute-values` 파라미터는 다음과 같습니다.

```
{
    ":newval":{"N":"12"}
}
```

Bob의 요청은 성공하지만 Alice의 이전 업데이트는 손실됩니다.

조건부 `PutItem`, `DeleteItem` 또는 `UpdateItem`을 요청하려면 조건 표현식을 지정합니다. 조건 표현식은 속성 이름, 조건부 연산자 및 기본 제공 함수를 포함하는 문자열입니다. 전체 표현식이 true로 평가되어야 합니다. 그렇지 않으면 작업이 실패합니다.

이제 조건부 쓰기를 사용하여 Alice의 업데이트를 덮어쓰지 못하도록 하는 방법을 보여 주는 다음 다이어그램을 생각해 봅니다.




처음에 Alice는 현재 `Price`가 10인 경우에만 `Price`를 8로 업데이트하려고 시도합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
    --condition-expression "Price = :currval" \
```

```
    --expression-attribute-values file://expression-attribute-values.json
```

`--expression-attribute-values`의 인수는 `expression-attribute-values.json` 파일에 저장됩니다.

```
{
    ":newval":{"N":"8"},
    ":currval":{"N":"10"}
}
```

조건이 true로 평가되기 때문에 Alice의 업데이트가 성공합니다.

다음에 Bob은 현재 `Price`가 10인 경우에만 `Price`를 12로 업데이트하려고 시도합니다. Bob의 경우 `--expression-attribute-values` 파라미터는 다음과 같습니다.

```
{
    ":newval":{"N":"12"},
    ":currval":{"N":"10"}
}
```

Alice가 이전에 `Price`를 8로 변경했기 때문에 조건 표현식은 false로 평가되고 Bob의 업데이트는 실패합니다.

자세한 내용은 DynamoDB 조건 표현식 CLI 예제 섹션을 참조하세요.

## 조건부 쓰기 멱등성

업데이트 할 동일한 속성에 대한 조건부 검사의 경우 조건부 쓰기는 idempotent가 될 수 있습니다. 즉, 항목의 특정 속성 값이 요청 시 원하는 값과 일치하는 경우에만 DynamoDB가 해당 쓰기 요청을 수행합니다.

예를 들어, `Price`가 현재 20인 경우에만 항목의 `Price`를 3으로 증가시키는 `UpdateItem` 요청을 실행한다고 가정합니다. 요청을 보낸 후 결과를 다시 받기 전에 네트워크 오류가 나타나고 요청이 성공적이었는지 여부를 알 수 없습니다. 이 조건부 쓰기는 멱등성 방식이기 때문에 동일한 `UpdateItem` 요청을 재시도할 수 있으며 `Price`가 현재 20인 경우에만 DynamoDB가 항목을 업데이트합니다.

## 조건부 쓰기에서 사용된 용량 단위

조건부 쓰기 중에 `ConditionExpression`이 false로 평가되는 경우에도 DynamoDB는 테이블에서 쓰기 용량을 사용합니다. 사용되는 양은 기존 항목의 크기(또는 최소 1개)에 따라 달라집니다. 예를 들

어 기존 항목이 300kb이고 새로 만들거나 업데이트하려는 항목이 310kb인 경우, 사용되는 쓰기 용량 단위는 조건이 충족되지 않으면 300, 조건이 충족되면 310이 됩니다. 새 항목(기존 항목 없음)인 경우, 사용되는 쓰기 용량 단위는 조건이 충족되지 않으면 1, 조건이 충족되면 310이 됩니다.

> **Note**
>
> 쓰기 작업은 쓰기 용량 단위만 사용합니다. 이 작업은 읽기 용량 단위를 사용하지 않습니다.

조건부 쓰기가 실패하면 `ConditionalCheckFailedException`이 반환됩니다. 이 상황이 발생하면 사용된 쓰기 용량에 대한 정보를 응답에서 받을 수 없습니다.

조건부 쓰기 중에 사용된 쓰기 용량 단위 수를 반환하려면 `ReturnConsumedCapacity` 파라미터를 사용합니다.

- `TOTAL` - 사용된 총 쓰기 용량 단위 수를 반환합니다.
- `INDEXES` - 사용된 총 쓰기 용량 단위 수와 작업의 영향을 받은 테이블 및 보조 인덱스의 소계를 반환합니다.
- `NONE` - 쓰기 용량 세부 정보를 반환하지 않습니다. (이 값이 기본값입니다.)

> **Note**
>
> 글로벌 보조 인덱스와 달리 로컬 보조 인덱스는 프로비저닝된 처리 용량을 해당 테이블과 공유합니다. 로컬 보조 인덱스에서의 읽기 또는 쓰기 작업에는 테이블의 프로비저닝된 처리 용량이 사용됩니다.

## DynamoDB에서 표현식 사용

Amazon DynamoDB에서는 표현식을 사용하여 항목에서 읽을 속성을 지정하고, 조건이 충족될 때 데이터를 쓰고, 항목을 업데이트하는 방법을 지정하고, 쿼리를 정의하고, 쿼리 결과를 필터링할 수 있습니다.

이 표에서는 기본 표현식 문법과 사용 가능한 표현식 종류를 설명합니다.

| 표현식 유형 | 설명 |
| --- | --- |
| 프로젝션 표현식 | 프로젝션 표현식은 GetItem, Query 또는 Scan과 같은 작업을 사용할 때 항목에서 검색하려는 속성을 식별합니다. |
| 조건 표현식 | 조건 표현식은 PutItem, UpdateItem 및 DeleteItem 작업을 사용할 때 수정해야 하는 항목을 결정합니다. |
| 업데이트 표현식 | 업데이트 표현식은 UpdateItem이 항목의 속성을 수정하는 방법을 지정합니다(예: 스칼라 값 설정 또는 목록이나 맵에서 요소 제거). |
| 키 조건 표현식 | 키 조건 표현식은 쿼리가 테이블 또는 인덱스에서 읽을 항목을 결정합니다. |
| 필터 표현식 | 필터 표현식은 Query 결과 내에서 어떤 항목을 반환할지를 결정합니다. 다른 모든 결과는 폐기됩니다. |

표현식 구문과 각 표현식 유형에 대한 자세한 내용은 다음 섹션을 참조하세요.

주제

- DynamoDB에서 표현식 사용 시 항목 속성 참조
- DynamoDB의 표현식 속성 이름(별칭)
- DynamoDB에서 표현식 속성 값 사용
- DynamoDB에서 프로젝션 표현식 사용
- DynamoDB에서 업데이트 표현식 사용
- DynamoDB의 조건 및 필터 표현식, 연산자, 함수
- DynamoDB 조건 표현식 CLI 예제

> **Note**
>
> DynamoDB에서는 이전 버전과의 호환성을 위해 표현식을 사용하지 않는 조건부 파라미터도 지원합니다. 자세한 내용은 레거시 DynamoDB 조건부 파라미터 섹션을 참조하세요.
> 새 애플리케이션에서는 레거시 파라미터보다는 식을 사용해야 합니다.

## DynamoDB에서 표현식 사용 시 항목 속성 참조

이 섹션에서는 Amazon DynamoDB의 표현식에서 항목 속성을 참조하는 방법을 설명합니다. 여러 목록과 맵 내부에 깊이 중첩된 경우에도 속성에 대한 작업을 수행할 수 있습니다.

주제

- 최상위 속성
- 중첩 속성
- 문서 경로

샘플 항목: ProductCatalog

이 페이지의 예에서는 `ProductCatalog` 테이블의 다음 샘플 항목을 사용합니다. 이 테이블에 대해서는 DynamoDB에서 사용할 테이블 및 데이터 예제에서 설명합니다.

```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "Description": "123 description",
    "BicycleType": "Hybrid",
    "Brand": "Brand-Company C",
    "Price": 500,
    "Color": ["Red", "Black"],
    "ProductCategory": "Bicycle",
    "InStock": true,
    "QuantityOnHand": null,
    "RelatedItems": [
        341,
        472,
        649
    ],
    "Pictures": {
```

```
        "FrontView": "http://example.com/products/123_front.jpg",
        "RearView": "http://example.com/products/123_rear.jpg",
        "SideView": "http://example.com/products/123_left_side.jpg"
    },
    "ProductReviews": {
     "FiveStar": [
       "Excellent! Can't recommend it highly enough! Buy it!",
       "Do yourself a favor and buy this."
     ],
     "OneStar": [
       "Terrible product! Do not buy this."
     ]
    },
    "Comment": "This product sells out quickly during the summer",
    "Safety.Warning": "Always wear a helmet"
}
```

다음 사항에 유의하세요.

- 파티션 키 값(Id)은 123입니다. 정렬 키는 없습니다.
- 대부분의 속성에는 String, Number, Boolean 및 Null 같은 스칼라 데이터 형식이 있습니다.
- 속성 하나(Color)는String Set입니다.
- 다음 속성은 문서 데이터 형식입니다.
  - RelatedItems 목록. 각 요소가 관련 제품의 Id입니다.
  - Pictures 맵. 각 요소가 이미지에 대한 간단한 설명이며, 해당 이미지 파일의 URL이 함께 제공됩니다.
  - ProductReviews 맵. 각 요소가 평점과 해당 평점에 상응하는 리뷰 목록을 나타냅니다. 처음에 이 맵은 별 5개 및 1개 리뷰로 채워집니다.

### 최상위 속성

속성이 다른 속성 내부에 포함되어 있지 않은 경우 이러한 속성을 최상위라고 합니다.
ProductCatalog 항목의 경우 최상위 속성은 다음과 같습니다.

- Id
- Title
- Description
- BicycleType

- Brand
- Price
- Color
- ProductCategory
- InStock
- QuantityOnHand
- RelatedItems
- Pictures
- ProductReviews
- Comment
- Safety.Warning

이러한 최상위 속성은 Color(목록), RelatedItems(목록), Pictures(맵) 및 ProductReviews(맵)를 제외하고 모두 스칼라입니다.

중첩 속성

속성이 다른 속성 내부에 포함되어 있는 경우 이러한 속성을 중첩이라고 합니다. 중첩 속성에 액세스하려면 역참조 연산자를 사용합니다.

- [n] - 목록 요소의 경우
- .(점) - 맵 요소의 경우

목록 요소에 액세스

목록 요소의 역참조 연산자는 [N]이며, 여기에서 n은 요소 번호입니다. 목록 요소는 0부터 시작되므로 [0]은 목록의 첫 번째 요소, [1]은 두 번째 요소를 나타냅니다. 여기 몇 가지 예가 있습니다:

- MyList[0]
- AnotherList[12]
- ThisList[5][11]

ThisList[5] 요소 자체는 하나의 중첩된 목록입니다. 그러므로 ThisList[5][11]은 해당 목록의 열두 번째 요소를 나타냅니다.

대괄호 내부의 숫자는 음수가 아닌 정수여야 합니다. 그러므로 다음 표현식은 유효하지 않습니다.

- `MyList[-1]`
- `MyList[0.4]`

맵 요소에 액세스

맵 요소의 역참조 연산자는 .(점)입니다. 점을 맵 내에서 요소 간 구분 기호로 사용합니다.

- `MyMap.nestedField`
- `MyMap.nestedField.deeplyNestedField`

문서 경로

표현식에서 문서 경로를 사용하여 DynamoDB에 속성을 찾을 위치를 알려 줍니다. 최상의 속성의 경우, 문서 경로는 단순히 속성 이름입니다. 중첩된 속성의 경우 역참조 연산자를 사용하여 문서 경로를 생성합니다.

문서 경로의 몇 가지 예는 다음과 같습니다. DynamoDB에서 표현식 사용 시 항목 속성 참조에 표시된 항목을 참조하세요.

- 최상위 스칼라 속성.

  `Description`
- 최상위 목록 속성. (이 속성은 일부 요소만이 아닌 전체 목록을 반환합니다.)

  `RelatedItems`
- `RelatedItems` 목록의 세 번째 요소입니다. 목록 요소는 0부터 시작합니다.

  `RelatedItems[2]`
- 제품의 전면도 그림입니다.

  `Pictures.FrontView`
- 모든 별 5개 리뷰입니다.

  `ProductReviews.FiveStar`
- 별 5개 리뷰 중 첫 번째 리뷰입니다.

  `ProductReviews.FiveStar[0]`

> **Note**
>
> 문서 경로의 최대 깊이는 32입니다. 따라서 경로의 역참조 연산자 수는 이 제한을 초과할 수 없습니다.

다음 요구 사항을 충족하는 한 문서 경로에 모든 속성 이름을 사용할 수 있습니다.

- 첫 번째 문자는 `a-z`, `A-Z` 또는 `0-9`여야 합니다.
- 두 번째 문자(있는 경우)는 `a-z` 또는 `A-Z`여야 합니다.

> **Note**
>
> 속성 이름이 이 요구 사항을 충족하지 않으면 표현식 속성 이름을 자리 표시자로 정의해야 합니다.

자세한 내용은 DynamoDB의 표현식 속성 이름(별칭) 섹션을 참조하세요.

## DynamoDB의 표현식 속성 이름(별칭)

표현식 속성 이름은 Amazon DynamoDB 표현식에서 실제 속성 이름의 대체 이름으로 사용하는 별칭(자리 표시자)입니다. 표현식 속성 이름은 파운드 기호(#)로 시작해야 하며, 그 뒤에 하나 이상의 영숫자가 이어져야 합니다. 밑줄(_) 문자도 사용할 수 있습니다.

이 섹션에서는 표현식 속성 이름을 사용해야 하는 여러 가지 상황에 대해 설명합니다.

> **Note**
>
> 이 섹션의 예제에서는 AWS Command Line Interface(AWS CLI)를 사용합니다.

주제

- 예약어
- 특수 문자가 포함된 속성 이름
- 중첩 속성
- 속성 이름 반복 참조

### 예약어

경우에 따라 DynamoDB 예약어와 충돌하는 속성 이름을 포함하는 표현식을 작성해야 할 때가 있습니다. 전체 예약어 목록은 DynamoDB의 예약어를 참조하세요.

예를 들어, 다음 AWS CLI 예제는 `COMMENT`가 예약어이기 때문에 실패합니다.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Comment"
```

이 문제를 해결하려면 `Comment`를 `#c`와 같은 표현식 속성 이름으로 대체할 수 있습니다. #(파운드 기호)가 필요하며 이 기호는 속성 이름의 자리 표시자임을 나타냅니다. 이제 AWS CLI 예제는 다음과 같습니다.

```
aws dynamodb get-item \
     --table-name ProductCatalog \
     --key '{"Id":{"N":"123"}}' \
     --projection-expression "#c" \
     --expression-attribute-names '{"#c":"Comment"}'
```

> **Note**
>
> 속성 이름이 숫자로 시작하거나 공백을 포함하거나 예약어를 포함하는 경우, 반드시 표현식 속성 이름을 사용하여 표현식에서 해당 속성의 이름을 대체해야 합니다.

### 특수 문자가 포함된 속성 이름

식에서 점(".")은 문서 경로의 특수 문자로 해석됩니다. 하지만 DynamoDB에서는 점 문자 및 하이픈('-')과 같은 기타 특수 문자를 속성 이름의 일부로 사용할 수도 있습니다. 이러한 사용은 경우에 따라 의미가 명확하지 않을 수도 있습니다. 설명을 위해 `ProductCatalog` 항목에서 `Safety.Warning` 속성을 가져오려 한다고 가정합니다(DynamoDB에서 표현식 사용 시 항목 속성 참조 참조).

프로젝션 표현식을 사용하여 `Safety.Warning`에 액세스하려 한다고 가정하겠습니다.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
```

```
    --projection-expression "Safety.Warning"
```

DynamoDB는 예상 문자열("Always wear a helmet")이 아닌 빈 결과를 반환합니다. 이는 DynamoDB가 표현식의 점을 문서 경로 구분 기호로 해석하기 때문입니다. 이 경우 표현식 속성 이름(#sw)을 Safety.Warning의 대체 이름으로 정의해야 합니다. 그러고 나면 다음 프로젝션 표현식을 사용할 수 있습니다.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#sw" \
    --expression-attribute-names '{"#sw":"Safety.Warning"}'
```

그러면 DynamoDB에서 정확한 결과가 반환됩니다.

> **Note**
>
> 속성 이름에 점('.') 또는 하이픈('-') 이 포함된 경우 반드시 표현식 속성 이름을 사용하여 표현식에서 해당 속성의 이름을 대체해야 합니다.

중첩 속성

중첩된 속성 ProductReviews.OneStar에 액세스하려고 한다고 가정해 보겠습니다. 표현식 속성 이름에서 DynamoDB는 점(".")을 속성 이름 내의 문자로 취급합니다. 중첩된 속성을 참조하려면 문서 경로에 있는 각 요소의 표현식 속성 이름을 정의합니다.

- #pr – ProductReviews
- #1star – OneStar

이제 프로젝션 표현식에 #pr.#1star를 사용할 수 있습니다.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.#1star"  \
    --expression-attribute-names '{"#pr":"ProductReviews", "#1star":"OneStar"}'
```

그러면 DynamoDB에서 정확한 결과가 반환됩니다.

### 속성 이름 반복 참조

표현식 속성 이름은 동일한 속성 이름을 반복적으로 참조해야 할 때 도움이 됩니다. 예를 들어 ProductCatalog 항목에서 리뷰 몇 개를 검색하는 다음 식을 살펴보겠습니다.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "ProductReviews.FiveStar, ProductReviews.ThreeStar,
 ProductReviews.OneStar"
```

식을 간소화하기 위해 ProductReviews를 #pr과 같은 식 속성 이름으로 대체할 수 있습니다. 수정된 표현식은 다음과 같습니다.

- #pr.FiveStar, #pr.ThreeStar, #pr.OneStar

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.FiveStar, #pr.ThreeStar, #pr.OneStar" \
    --expression-attribute-names '{"#pr":"ProductReviews"}'
```

식 속성 이름을 정의하는 경우 전체 식에서 해당 식 속성 이름을 일관되게 사용해야 합니다. 또한 # 기호를 생략하면 안 됩니다.

## DynamoDB에서 표현식 속성 값 사용

Amazon DynamoDB의 표현식 속성 값은 변수로 작동합니다. 이 값은 런타임까지 모를 수 있는 비교하려는 실제 값을 대체합니다. 표현식 속성 값은 콜론(:)으로 시작해야 하고, 그 뒤에 하나 이상의 영숫자가 와야 합니다.

예를 들어 Black에서 사용 가능하고 비용이 500 이하인 모든 ProductCatalog 항목을 반환하고 싶었다고 가정합니다. 다음 이 AWS Command Line Interface(AWS CLI) 예제와 같이 필터 표현식과 함께 Scan 작업을 사용할 수 있습니다.

```
aws dynamodb scan \
    --table-name ProductCatalog \
    --filter-expression "contains(Color, :c) and Price <= :p" \
    --expression-attribute-values file://values.json
```

`--expression-attribute-values`의 인수는 `values.json` 파일에 저장됩니다.

```
{
    ":c": { "S": "Black" },
    ":p": { "N": "500" }
}
```

식 속성 값을 정의하는 경우 전체 식에서 해당 식 속성 이름을 일관되게 사용해야 합니다. 또한 : 기호를 생략하면 안 됩니다.

표현식 속성 값은 키 조건 표현식, 조건 표현식, 업데이트 표현식 및 필터 표현식에서 사용됩니다.

## DynamoDB에서 프로젝션 표현식 사용

테이블에서 데이터를 읽으려면 `GetItem`, `Query`, `Scan` 등과 같은 작업을 사용합니다. Amazon DynamoDB는 기본적으로 모든 항목 속성을 반환합니다. 모든 속성 대신 일부 속성만 가져오려면 프로젝션 표현식을 사용합니다.

프로젝션 표현식은 원하는 속성을 식별하는 문자열입니다. 단일 속성을 가져오려면 속성의 이름을 지정합니다. 여러 속성의 경우 이름을 쉼표로 구분해야 합니다.

DynamoDB에서 표현식 사용 시 항목 속성 참조의 `ProductCatalog` 항목을 기반으로 한 프로젝션 표현식의 몇 가지 예는 다음과 같습니다.

- 단일 최상위 속성입니다.

  ```
  Title
  ```

- 세 가지 최상위 속성입니다. DynamoDB는 전체 `Color` 집합을 검색합니다.

  ```
  Title, Price, Color
  ```

- 네 가지 최상위 속성입니다. DynamoDB는 `RelatedItems` 및 `ProductReviews`의 전체 내용을 반환합니다.

  ```
  Title, Description, RelatedItems, ProductReviews
  ```

> **Note**
>
> 프로젝션 표현식은 프로비저닝된 처리량 소비에 영향을 주지 않습니다. DynamoDB는 애플리케이션에 반환되는 데이터 크기 대신 항목 크기를 기준으로 소비된 용량 단위를 결정합니다.

### 예약어 및 특수 문자

DynamoDB에는 예약어와 특수 문자가 있습니다. DynamoDB에서는 이름에 이러한 예약어와 특수 문자를 사용하도록 허용하지만, 표현식에서 이러한 이름을 사용할 때마다 별칭을 사용해야 하므로 사용하지 않는 것이 좋습니다. 전체 목록은 DynamoDB의 예약어 섹션을 참조하세요.

다음과 같은 경우에는 실제 이름 대신 표현식 속성 이름을 사용해야 합니다.

- 속성 이름은 DynamoDB의 예약어 목록에 있습니다.
- 속성 이름이 첫 번째 문자가 `a-z` 또는 `A-Z`이고 두 번째 문자(있는 경우)가 `a-Z`, `A-Z` 또는 `0-9`여야 하는 요구 사항을 충족하지 않습니다.
- 속성 이름에는 #(해시) 또는 :(콜론)이 포함됩니다.

다음은 `GetItem` 작업과 함께 프로젝션 표현식을 사용하는 방법을 보여주는 AWS CLI 예입니다. 이 프로젝션 표현식은 최상위 스칼라 속성(`Description`), 목록의 첫 번째 요소(`RelatedItems[0]`) 및 맵 내에 중첩된 목록(`ProductReviews.FiveStar`)을 가져옵니다.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '"Id": { "N": "123" } \
    --projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"
```

이 예제에서는 다음 JSON이 반환됩니다.

```
{
    "Item": {
        "Description": {
            "S": "123 description"
        },
        "ProductReviews": {
            "M": {
                "FiveStar": {
                    "L": [
                        {
                            "S": "Excellent! Can't recommend it highly enough! Buy it!"
                        },
                        {
                            "S": "Do yourself a favor and buy this."
                        }
                    ]
```

```
                }
            }
        },
        "RelatedItems": {
            "L": [
                {
                    "N": "341"
                }
            ]
        }
    }
}
```

## DynamoDB에서 업데이트 표현식 사용

`UpdateItem` 작업은 기존 항목을 업데이트하거나 기존 항목이 없는 경우 새 항목을 테이블에 추가합니다. 업데이트할 항목의 키를 제공해야 합니다. 또한 수정할 속성과 그러한 속성에 할당할 값을 지정하는 업데이트 표현식도 제공해야 합니다.

업데이트 표현식은 `UpdateItem`이 항목의 속성을 수정하는 방법을 지정합니다(예: 스칼라 값 설정 또는 목록이나 맵에서 요소 제거).

다음은 업데이트 표현식에 대한 구문을 요약한 것입니다.

```
update-expression ::=
    [ SET action [, action] ... ]
    [ REMOVE action [, action] ...]
    [ ADD action [, action] ... ]
    [ DELETE action [, action] ...]
```

업데이트 표현식은 하나 이상의 절로 구성됩니다. 각 절은 `SET`, `REMOVE`, `ADD` 또는 `DELETE` 키워드로 시작됩니다. 이러한 절은 업데이트 표현식에 어떤 순서로든 포함할 수 있습니다. 하지만 각 작업 키워드는 한 번만 표시할 수 있습니다.

각 절 내에는 하나 이상의 작업이 쉼표로 구분되어 있습니다. 각 작업은 데이터 수정을 나타냅니다.

이 섹션의 예제는 DynamoDB에서 프로젝션 표현식 사용의 `ProductCatalog` 항목을 기반으로 합니다.

아래 주제에서는 `SET` 작업의 다양한 사용 사례를 다룹니다.

주제

- SET - 항목 속성 수정 또는 추가
- REMOVE - 항목에서 속성 삭제
- ADD - 숫자 및 세트 업데이트
- DELETE - 세트에서 요소 제거
- 여러 업데이트 표현식 사용

### SET - 항목 속성 수정 또는 추가

업데이트 표현식에서 SET 작업을 사용하여 하나 이상의 속성을 항목에 추가합니다. 이러한 속성 중 하나가 이미 존재하는 경우에는 새 값이 해당 값을 덮어씁니다. 기존 속성의 덮어쓰기를 방지하려는 경우 `if_not_exists` 함수와 함께 SET를 사용할 수 있습니다. `if_not_exists` 함수는 SET 작업에 고유하며 업데이트 표현식에서만 사용할 수 있습니다.

SET을 사용하여 목록 요소를 업데이트하는 경우 해당 요소의 내용이 사용자가 지정한 새 데이터로 바뀝니다. 요소가 아직 없으면 SET은 목록의 끝에 새 요소를 추가합니다.

단일 SET 작업에서 여러 요소를 추가하는 경우 요소가 요소 번호에 따라 순서대로 정렬됩니다.

SET을 사용하여 Number 형식의 속성에서 더하거나 뺄 수도 있습니다. 여러 SET 작업을 수행하려면 각 작업을 쉼표로 구분합니다.

다음 구문 요약에서:

- *path* 요소는 항목의 문서 경로입니다.
- *####* 요소는 항목의 문서 경로 또는 함수일 수 있습니다.

```
set-action ::=
    path = value

value ::=
    operand
    | operand '+' operand
    | operand '-' operand

operand ::=
    path | function

function ::=
    if_not_exists (path, value)
```

항목의 지정된 경로에 속성이 포함되지 않으면 `if_not_exists`가 `value`로 평가됩니다. 그렇지 않으면 `path`로 평가됩니다.

다음 `PutItem` 작업은 예제에서 참조할 샘플 항목을 생성합니다.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

`--item`의 인수는 `item.json` 파일에 저장됩니다. 간소화를 위해 항목 속성 몇 개만 사용됩니다.

```
{
    "Id": {"N": "789"},
    "ProductCategory": {"S": "Home Improvement"},
    "Price": {"N": "52"},
    "InStock": {"BOOL": true},
    "Brand": {"S": "Acme"}
}
```

주제

- 속성 수정
- 목록 및 맵 추가
- 목록에 요소 추가
- 중첩 맵 속성 추가
- 수치 속성 증가 및 감소
- 목록에 요소 추가
- 기존 속성의 덮어쓰기 방지

### 속성 수정

#### Example

`ProductCategory` 및 `Price` 속성을 업데이트합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET ProductCategory = :c, Price = :p" \
    --expression-attribute-values file://values.json \
```

```
    --return-values ALL_NEW
```

`--expression-attribute-values`의 인수는 `values.json` 파일에 저장됩니다.

```
{
    ":c": { "S": "Hardware" },
    ":p": { "N": "60" }
}
```

> **Note**
>
> `UpdateItem` 작업에서 `--return-values ALL_NEW`를 실행하면 DynamoDB에서 업데이트 후 나타나는 항목을 반환합니다.

목록 및 맵 추가

Example

새 목록과 새 맵을 추가합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems = :ri, ProductReviews = :pr" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

`--expression-attribute-values`의 인수는 `values.json` 파일에 저장됩니다.

```
{
    ":ri": {
        "L": [
            { "S": "Hammer" }
        ]
    },
    ":pr": {
        "M": {
            "FiveStar": {
                "L": [
                    { "S": "Best product ever!" }
```

```
                ]
            }
        }
    }
}
```

목록에 요소 추가

Example

새로운 속성을 RelatedItems 목록에 추가합니다. 목록 요소는 0부터 시작되므로 [0]은 목록의 첫 번째 요소를 나타내고 [1]은 두 번째 요소를 나타냅니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems[1] = :ri" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

--expression-attribute-values의 인수는 values.json 파일에 저장됩니다.

```
{
    ":ri": { "S": "Nails" }
}
```

> **Note**
>
> SET을 사용하여 목록 요소를 업데이트하는 경우 해당 요소의 내용이 사용자가 지정한 새 데이터로 바뀝니다. 요소가 아직 없으면 SET은 목록의 끝에 새 요소를 추가합니다.
> 단일 SET 작업에서 여러 요소를 추가하는 경우 요소가 요소 번호에 따라 순서대로 정렬됩니다.

중첩 맵 속성 추가

Example

일부 중첩 맵 속성을 추가합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
```

```
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #pr.#5star[1] = :r5, #pr.#3star = :r3" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

`--expression-attribute-names`의 인수는 `names.json` 파일에 저장됩니다.

```
{
    "#pr": "ProductReviews",
    "#5star": "FiveStar",
    "#3star": "ThreeStar"
}
```

`--expression-attribute-values`의 인수는 `values.json` 파일에 저장됩니다.

```
{
    ":r5": { "S": "Very happy with my purchase" },
    ":r3": {
        "L": [
            { "S": "Just OK - not that great" }
        ]
    }
}
```

수치 속성 증가 및 감소

기존 수치 속성에 더하거나 뺄 수 있습니다. 이 작업을 수행하려면 +(더하기) 및 -(빼기) 연산자를 사용합니다.

Example

항목의 `Price`을 줄입니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

`Price`을 증가시키려면 업데이트 표현식에서 + 연산자를 사용합니다.

목록에 요소 추가

목록 끝에 요소를 추가할 수 있습니다. 이렇게 하려면 list_append 함수와 함께 SET를 사용합니다. (파일 이름은 대/소문자를 구분합니다.) list_append 함수는 SET 작업에 고유하며 업데이트 표현식에서만 사용할 수 있습니다. 구문은 다음과 같습니다.

- `list_append (list1, list2)`

이 함수는 입력으로 두 개의 목록을 가져오고 *list2*의 모든 요소를 *list1*에 추가합니다.

Example

목록에 요소 추가에서는 RelatedItems 목록을 생성하고 목록을 Hammer 및 Nails라는 두 개의 요소로 채웠습니다. 이제 RelatedItems의 끝에 두 개의 요소를 더 추가합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #ri = list_append(#ri, :vals)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values file://values.json  \
    --return-values ALL_NEW
```

--expression-attribute-values의 인수는 values.json 파일에 저장됩니다.

```
{
    ":vals": {
        "L": [
            { "S": "Screwdriver" },
            {"S": "Hacksaw" }
        ]
    }
}
```

마지막으로 RelatedItems의 시작에 요소 하나를 더 추가합니다. 이렇게 하려면 list_append 요소의 순서를 스왑합니다. (list_append는 두 개의 목록을 입력으로 가져오고 두 번째 목록을 첫 번째 목록에 추가한다는 점을 기억하세요.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
```

```
    --update-expression "SET #ri = list_append(:vals, #ri)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values '{":vals": {"L": [ { "S": "Chisel" }]}}' \
    --return-values ALL_NEW
```

이제 결과적으로 생성되는 `RelatedItems` 속성에는 5개의 요소가 `Chisel`, `Hammer`, `Nails`, `Screwdriver`, `Hacksaw` 순서로 포함됩니다.

기존 속성의 덮어쓰기 방지

Example

항목에 `Price` 속성이 아직 없는 경우에만 항목의 `Price`를 설정합니다. (`Price`가 이미 있는 경우 아무 동작도 발생하지 않습니다.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = if_not_exists(Price, :p)" \
    --expression-attribute-values '{":p": {"N": "100"}}' \
    --return-values ALL_NEW
```

REMOVE - 항목에서 속성 삭제

업데이트 표현식에서 `REMOVE` 작업을 사용하여 Amazon DynamoDB에서 하나 이상의 속성을 항목에서 제거합니다. 여러 `REMOVE` 작업을 수행하려면 각 작업을 쉼표로 구분합니다.

다음은 업데이트 식의 `REMOVE`에 대한 구문을 요약한 것입니다. 제거하려는 속성의 문서 경로가 유일한 피연산자입니다.

```
remove-action ::=
    path
```

Example

항목에서 일부 속성을 제거합니다. (이 속성이 없는 경우 아무 동작도 발생하지 않습니다.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE Brand, InStock, QuantityOnHand" \
    --return-values ALL_NEW
```

### 목록에서 요소 제거

REMOVE를 사용하여 목록에서 개별 요소를 삭제합니다.

#### Example

목록에 요소 추가에서 다음 5개의 요소가 포함되도록 목록 속성(RelatedItems)을 수정합니다.

- [0]—Chisel
- [1]—Hammer
- [2]—Nails
- [3]—Screwdriver
- [4]—Hacksaw

다음 AWS Command Line Interface(AWS CLI) 예제에서는 목록에서 Hammer 및 Nails를 삭제합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE RelatedItems[1], RelatedItems[2]" \
    --return-values ALL_NEW
```

Hammer와 Nails를 제거한 후에는 나머지 요소가 이동됩니다. 이제 목록에는 다음이 포함됩니다.

- [0]—Chisel
- [1]—Screwdriver
- [2]—Hacksaw

### ADD - 숫자 및 세트 업데이트

> **Note**
>
> 일반적으로 ADD 보다는 SET을 사용할 것을 권장합니다.

업데이트 표현식에서 ADD 작업을 사용하여 새로운 속성과 값을 항목에 추가합니다.

속성이 이미 있으면 ADD의 동작은 속성의 데이터 형식에 따라 달라집니다.

- 속성이 숫자이고 추가하려는 값도 숫자이면 값이 산술적으로 기존 속성에 더해집니다. (값이 음수이면 기존 속성에서 차감됩니다.)
- 속성이 집합이고 추가하려는 값도 집합이면 값이 기존 집합에 추가됩니다.

> **Note**
>
> ADD 작업은 숫자 및 집합 데이터 형식만 지원합니다.

여러 ADD 작업을 수행하려면 각 작업을 쉼표로 구분합니다.

다음 구문 요약에서:

- *path* 요소는 속성에 대한 문서 경로입니다. 속성은 Number 또는 집합 데이터 형식이어야 합니다.
- *#* 요소는 속성에 더할 숫자(Number 데이터 형식의 경우) 또는 속성에 추가할 세트(집합 형식의 경우)입니다.

```
add-action ::=
    path value
```

아래 주제에서는 ADD 작업의 다양한 사용 사례를 다룹니다.

주제

- 숫자 추가
- 세트에 요소 추가

숫자 추가

QuantityOnHand 속성이 존재하지 않는다고 가정합니다. 다음 AWS CLI 예제에서는 QuantityOnHand을 5로 설정합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD QuantityOnHand :q" \
```

```
    --expression-attribute-values '{":q": {"N": "5"}}' \
    --return-values ALL_NEW
```

이제 QuantityOnHand가 존재하므로 예제를 다시 실행하여 QuantityOnHand를 매번 5씩 증가시킬 수 있습니다.

세트에 요소 추가

Color 속성이 존재하지 않는다고 가정합니다. 다음 AWS CLI 예제는 두 요소가 있는 문자열 집합으로 Color을 설정합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Orange", "Purple"]}}' \
    --return-values ALL_NEW
```

이제 Color가 존재하므로 여기에 요소를 더 추가할 수 있습니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Yellow", "Green", "Blue"]}}' \
    --return-values ALL_NEW
```

DELETE - 세트에서 요소 제거

> **Important**
>
> DELETE 작업은 Set 데이터 형식만 지원합니다.

업데이트 표현식에서 DELETE 작업을 사용하여 집합에서 하나 이상의 요소를 제거합니다. 여러 DELETE 작업을 수행하려면 각 작업을 쉼표로 구분합니다.

다음 구문 요약에서:

- *path* 요소는 속성에 대한 문서 경로입니다. 속성은 설정된 Set 데이터 형식이어야 합니다.

- *## ##*은 *##*에서 삭제하려는 하나 이상의 요소입니다. *## ##*을 집합 형식으로 지정해야 합니다.

```
delete-action ::=
    path subset
```

Example

세트에 요소 추가에서 Color 문자열 집합을 생성합니다. 이 예제에서는 해당 집합에서 일부 요소를 제거합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "DELETE Color :p" \
    --expression-attribute-values '{":p": {"SS": ["Yellow", "Purple"]}}' \
    --return-values ALL_NEW
```

여러 업데이트 표현식 사용

단일 명령에 여러 업데이트 표현식을 사용할 수 있습니다.

Example

속성 값을 수정하고 다른 속성을 완전히 제거하려면 단일 명령문에서 SET 및 REMOVE 작업을 사용할 수 있습니다. 이 작업을 수행하면 Price 값이 15로 감소되고 항목에서 InStock 속성이 제거됩니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p REMOVE InStock" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

Example

목록에 추가하고 다른 속성의 값도 변경하려면 단일 명령문에 두 개의 SET 작업을 사용할 수 있습니다. 이 작업을 수행하면 RelatedItems 목록 속성에 'Nails'가 추가되고 Price 값도 21로 설정됩니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
```

```
    --update-expression "SET RelatedItems[1] = :newValue, Price = :newPrice" \
    --expression-attribute-values '{":newValue": {"S":"Nails"}, ":newPrice":
{"N":"21"}}'  \
    --return-values ALL_NEW
```

## DynamoDB의 조건 및 필터 표현식, 연산자, 함수

DynamoDB 테이블에서 데이터를 조작하려면 `PutItem`, `UpdateItem` 및 `DeleteItem` 작업을 사용합니다. 이러한 데이터 조작 작업의 경우 조건 표현식을 지정하여 어떤 항목을 수정할지를 결정할 수 있습니다. 조건 표현식이 true로 평가되는 경우 작업이 성공합니다. 그렇지 않으면 작업이 실패합니다.

이 섹션에서는 Amazon DynamoDB에서 필터 표현식 및 조건 표현식을 작성하기 위한 기본 제공 함수와 키워드에 대해 살펴봅니다. DynamoDB를 사용한 함수 및 프로그래밍에 대한 자세한 내용은 DynamoDB 및 AWS SDK를 사용한 프로그래밍 및 DynamoDB API 참조를 참조하세요.

주제

- 필터 및 조건 표현식 구문
- 비교 실행
- 함수
- 논리 평가
- 괄호
- 조건식의 우선 순위

### 필터 및 조건 표현식 구문

아래의 구문 요약에서 *### ##*는 다음과 같이 사용됩니다.

- `Id`, `Title`, `Description`, `ProductCategory` 등의 최상위 속성
- 내포 속성을 참조하는 문서 경로

```
condition-expression ::=
      operand comparator operand
    | operand BETWEEN operand AND operand
    | operand IN ( operand (',' operand (, ...) ))
    | function
    | condition AND condition
    | condition OR condition
    | NOT condition
```

```
    | ( condition )

comparator ::=
    =
    | <>
    | <
    | <=
    | >
    | >=

function ::=
    attribute_exists (path)
    | attribute_not_exists (path)
    | attribute_type (path, type)
    | begins_with (path, substr)
    | contains (path, operand)
    | size (path)
```

비교 실행

다음 비교기를 사용하여 피연산자를 단일 값과 비교합니다.

- *a* = *b* - *a*가 *b*와 같은 경우 true입니다.
- *a* <> *b* - *a*가 *b*와 같지 않은 경우 true입니다.
- *a* < *b* - *a*가 *b*보다 작은 경우 true입니다.
- *a* <= *b* - *a*가 *b*보다 작거나 같은 경우 true입니다.
- *a* > *b* - *a*가 *b*보다 큰 경우 true입니다.
- *a* >= *b* - *a*가 *b*보다 크거나 같은 경우 true입니다.

BETWEEN 및 IN 키워드를 사용하여 피연산 함수와 값의 범위 또는 열거된 값 목록을 비교합니다.

- *a* BETWEEN *b* AND *c* - *a*가 *b*보다 크거나 같고 *c*보다 작거나 같은 경우 true입니다.
- *a* IN (*b*, *c*, *d*) - *a*가 목록의 임의 값(*b*, *c*, *d* 값 등)과 같은 경우 true입니다. 목록에는 쉼표로 구분된 최대 100개의 값이 포함될 수 있습니다.

함수

다음의 함수를 사용하여 항목에 속성이 존재하는지 판단하거나 속성 값을 평가합니다. 함수 이름은 대/소문자를 구분합니다. 중첩 속성의 경우 전체 문서 경로를 제공해야 합니다.

| 함수 | 설명 |
| --- | --- |
| `attribute_exists ( path)` | path에 의해 지정된 속성이 항목에 포함되어 있는 경우 true입니다.<br><br>예: `Product` 테이블에 있는 항목에 측면 사진이 있는지를 확인합니다.<br><br>• `attribute_exists (#Pictures.#SideView)` |
| `attribute_not_exists ( path)` | path에 의해 지정된 속성이 항목에 포함되어 있지 않은 경우 true입니다.<br><br>예: 항목에 `Manufacturer` 속성이 포함되어 있는지 확인합니다.<br><br>• `attribute_not_exists (Manufacturer)` |
| `attribute_type ( path, type)` | 지정된 경로의 속성이 특정 데이터 유형인 경우 true. `type` 파라미터는 반드시 다음 값 중 하나이어야 합니다.<br><br>• `S` - 문자열<br>• `SS` - 문자열 집합<br>• `N` - 숫자<br>• `NS` - 숫자 집합<br>• `B` - 이진수<br>• `BS` - 이진수 집합<br>• `BOOL` - 부울 |

| 함수 | 설명 |
| --- | --- |
| | • NULL - Null<br>• L - 목록<br>• M - 맵<br><br>type 파라미터에는 반드시 표현식 속성 값을 사용해야 합니다.<br><br>예: QuantityOnHand 속성이 List 유형인지 확인합니다. 위의 예에서 :v_sub가 L 문자열의 자리 표시자입니다.<br><br>• attribute_type (ProductReviews.FiveStar, :v_sub)<br><br>type 파라미터에는 반드시 표현식 속성 값을 사용해야 합니다. |
| begins_with ( *path*, *substr*) | path에 의해 지정된 속성이 특정 하위 문자열로 시작하는 경우 true입니다.<br><br>예: 정면 사진 URL이 http://로 시작되는지 확인합니다.<br><br>• begins_with (Pictures.FrontView, :v_sub)<br><br>수식 속성 값인 :v_sub가 http://의 자리 표시자입니다. |

| 함수 | 설명 |
| --- | --- |
| `contains (path, operand)` | path에 의해 지정된 속성이 다음에 해당하는 경우 true입니다.<br>• 특정 하위 문자열을 포함하는 String<br>• 집합 내에 특정 요소를 포함하는 Set입니다.<br>• 목록 내에 특정 요소를 포함하는 List입니다.<br><br>path에서 지정한 속성이 String인 경우 operand는 String이어야 합니다. path에서 지정한 속성이 Set인 경우 operand는 집합의 요소 형식이어야 합니다.<br><br>경로와 피연산자는 서로 달라야 합니다. 즉, `contains (a, a)` 는 오류를 반환합니다.<br><br>예: Brand 속성에 하위 문자열 Company가 포함되어 있는지 확인합니다.<br>• `contains (Brand, :v_sub)`<br><br>수식 속성 값인 :v_sub가 Company의 자리 표시자입니다.<br><br>예: 빨간색 제품 색상을 선택할 수 있는지 확인합니다.<br>• `contains (Color, :v_sub)`<br><br>수식 속성 값인 :v_sub가 Red의 자리 표시자입니다. |

| 함수 | 설명 |
| --- | --- |
| `size (path)` | 속성의 크기를 나타내는 숫자가 반환됩니다. `size`를 사용하는 데 유효한 데이터 유형은 다음과 같습니다.<br><br>속성이 `String` 유형인 경우, `size`에서는 문자열의 길이가 반환됩니다.<br><br>예: `Brand` 문자열이 20자 이하인지 확인합니다. 수식 속성 값인 `:v_sub`가 `20`의 자리 표시자입니다.<br><br>• `size (Brand) <= :v_sub`<br><br>속성이 `Binary` 유형인 경우, `size`에서는 속성 값의 바이트 수가 반환됩니다.<br><br>예: `ProductCatalog` 항목에 사용 중인 제품의 동영상 클립이 포함된 `VideoClip` 이라는 바이너리 속성이 있는 경우를 가정할 수 있습니다. 다음 수식은 `VideoClip` 이 64,000 바이트를 초과하는지 확인합니다. 수식 속성 값인 `:v_sub`가 `64000`의 자리 표시자입니다.<br><br>• `size(VideoClip) > :v_sub`<br><br>속성이 `Set` 데이터 유형인 경우, `size`에서는 집합에 포함된 요소의 개수가 반환됩니다.<br><br>예: 한 가지 이상의 제품 색상을 선택할 수 있는지 확인합니다. 수식 속성 값인 `:v_sub`가 `1`의 자리 표시자입니다.<br><br>• |

| 함수 | 설명 |
| --- | --- |
| | `size (Color) < :v_sub`<br><br>속성이 `List` 또는 `Map` 유형인 경우, `size`에서는 집합에 포함된 요소의 개수가 반환됩니다.<br><br>예: `OneStar` 리뷰의 개수가 특정 임계 값을 초과하는지 확인합니다. 수식 속성 값인 `:v_sub`가 3의 자리 표시자입니다.<br><br>• `size(ProductReviews.OneStar) > :v_sub` |

논리 평가

AND, OR 및 NOT 키워드를 사용하여 논리 평가를 수행합니다. 아래의 목록에서 *a*와 *b*는 평가될 조건을 나타냅니다.

- *a* AND *b* - *a*와 *b*가 모두 true인 경우 true입니다.
- *a* OR *b* - *a* 또는 *b*(또는 둘 모두)가 true인 경우 true입니다.
- NOT *a* - *a*가 false인 경우 true입니다. *a*가 true인 경우 false입니다.

다음은 연산에서 AND를 사용한 코드 예제입니다.

```
dynamodb-local (*)> select * from exprtest where a > 3 and a < 5;
```

괄호

논리 평가의 우선 순위를 변경하려면 괄호를 사용합니다. 예를 들어 *a* 및 *b* 조건이 true이고 *c* 조건이 false인 경우를 가정할 수 있습니다. 다음 수식들은 true로 평가됩니다.

- *a* OR *b* AND *c*

그러나 조건을 괄호로 묶으면 해당 조건을 먼저 평가하게 됩니다. 예를 들어 아래의 경우에는 false로 평가됩니다.

- (*a* OR *b*) AND *c*

> **Note**
>
> 수식에 괄호를 포함시킬 수 있습니다. 가장 안쪽에 위치한 괄호부터 먼저 평가됩니다.

다음은 논리 평가에 괄호가 포함된 코드 예제입니다.

```
dynamodb-local (*)> select * from exprtest where attribute_type(b, string)
or ( a = 5 and c = “coffee”);
```

조건식의 우선 순위

DynamoDB는 다음의 우선 순위 규칙을 사용하여 왼쪽에서 오른쪽 방향으로 조건식을 평가합니다.

- = <> < <= > >=
- IN
- BETWEEN
- attribute_exists attribute_not_exists begins_with contains
- 괄호
- NOT
- AND
- OR

## DynamoDB 조건 표현식 CLI 예제

다음은 조건 표현식을 사용하는 몇 가지 AWS Command Line Interface(AWS CLI) 예제입니다. 이 예제는 DynamoDB에서 표현식 사용 시 항목 속성 참조에서 소개된 ProductCatalog 테이블을 기반으로 합니다. 이 테이블의 파티션 키는 Id이며 정렬 키는 없습니다. 다음 PutItem 작업은 예제에서 참조할 샘플 ProductCatalog 항목을 생성합니다.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

--item의 인수는 item.json 파일에 저장됩니다. 간소화를 위해 항목 속성 몇 개만 사용됩니다.

```
{
    "Id": {"N": "456" },
    "ProductCategory": {"S": "Sporting Goods" },
    "Price": {"N": "650" }
}
```

주제

- 조건부 추가
- 조건부 삭제
- 조건부 업데이트
- 조건부 표현식 예시

조건부 추가

PutItem 작업은 동일한 프라이머리 키가 있는 항목을 덮어씁니다(있는 경우). 이 동작을 방지하려면 조건 표현식을 사용합니다. 이렇게 하면 해당 항목에 동일한 프라이머리 키가 아직 없는 경우에만 쓰기를 진행할 수 있습니다.

다음 예제에서는 attribute_not_exists()를 사용하여, 쓰기 작업을 시도하기 전에 테이블에 프라이머리 키가 있는지 확인합니다.

> **Note**
>
> 프라이머리 키가 파티션 키(pk)와 정렬 키(sk)로 구성된 경우 이 파라미터는 쓰기 작업을 시도하기 전에 attribute_not_exists(pk) AND attribute_not_exists(sk) 전체 문이 true로 평가되는지 아니면 false로 평가되는지 확인합니다.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json \
    --condition-expression "attribute_not_exists(Id)"
```

조건 표현식이 false로 평가되는 경우 DynamoDB는 조건부 요청 실패라는 오류 메시지를 반환합니다.

> **Note**
>
> `attribute_not_exists` 및 기타 함수에 대한 자세한 내용은 DynamoDB의 조건 및 필터 표현식, 연산자, 함수을 참조하세요.

조건부 삭제

조건부 삭제를 수행하려면 조건 표현식과 함께 `DeleteItem` 작업을 사용합니다. 작업이 성공하려면 조건식이 true로 평가되어야 하며, 그렇지 않으면 작업이 실패합니다.

위에서 정의한 항목을 고려해 보겠습니다.

다음 조건에서만 항목을 삭제하려 한다고 가정합니다.

- `ProductCategory`는 "스포츠 상품" 또는 "원예용 소모품"입니다.
- `Price`는 500에서 600 사이입니다.

다음 예제에서는 항목을 삭제하려고 시도합니다.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (Price between :lo
 and :hi)" \
    --expression-attribute-values file://values.json
```

`--expression-attribute-values`의 인수는 `values.json` 파일에 저장됩니다.

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

> **Note**
>
> 조건 표현식에서 `:`(콜론 문자)은 실제 값의 자리 표시자인 표현식 속성 값을 나타냅니다. 자세한 내용은 DynamoDB에서 표현식 속성 값 사용 섹션을 참조하세요.

IN, AND 및 기타 키워드에 대한 자세한 내용은 DynamoDB의 조건 및 필터 표현식, 연산자, 함수 섹션을 참조하세요.

이 예에서 ProductCategory 비교는 true로 평가되지만 Price 비교는 false로 평가됩니다. 이로 인해 조건 표현식이 false로 평가되고 DeleteItem 작업이 실패합니다.

조건부 업데이트

조건부 업데이트를 수행하려면 조건 표현식과 함께 UpdateItem 작업을 사용합니다. 작업이 성공하려면 조건식이 true로 평가되어야 하며, 그렇지 않으면 작업이 실패합니다.

> **Note**
>
> UpdateItem은 업데이트 표현식도 지원하는데, 이 표현식에서 사용자는 항목에 대해 변경하고자 하는 사항을 지정합니다. 자세한 내용은 DynamoDB에서 업데이트 표현식 사용 섹션을 참조하세요.

위에서 정의한 항목으로 시작했다고 가정합니다.

다음 예제에서는 UpdateItem 작업을 수행합니다. 이 작업은 제품의 Price를 75만큼 감소시키려고 시도하지만, 조건 표현식은 현재 Price가 500보다 작거나 같은 경우 업데이트를 방지합니다.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --update-expression "SET Price = Price - :discount" \
    --condition-expression "Price > :limit" \
    --expression-attribute-values file://values.json
```

--expression-attribute-values의 인수는 values.json 파일에 저장됩니다.

```
{
    ":discount": { "N": "75"},
    ":limit": {"N": "500"}
}
```

시작 Price가 650이면 UpdateItem 작업은 Price를 575로 감소시킵니다. UpdateItem 작업을 다시 실행하면 Price는 500으로 감소됩니다. 세 번째로 작업을 실행하면 조건 표현식이 false로 평가되고 업데이트가 실패합니다.

> **Note**
>
> 조건 표현식에서 :(콜론 문자)은 실제 값의 자리 표시자인 표현식 속성 값을 나타냅니다. 자세한 내용은 DynamoDB에서 표현식 속성 값 사용 섹션을 참조하세요.
> ">" 및 기타 연산자에 대한 자세한 내용은 DynamoDB의 조건 및 필터 표현식, 연산자, 함수를 참조하세요.

조건부 표현식 예시

다음 예에 사용된 함수에 대한 자세한 내용은 DynamoDB의 조건 및 필터 표현식, 연산자, 함수 섹션을 참조하세요. 표현식에서 다양한 속성 유형을 지정하는 방법에 대한 자세한 내용은 DynamoDB에서 표현식 사용 시 항목 속성 참조 섹션을 참조하세요.

항목의 속성 확인

속성이 존재하는지(또는 존재하지 않는지)를 확인할 수 있습니다. 조건 표현식이 true로 평가되는 경우 작업이 성공하고, 그렇지 않으면 작업이 실패합니다.

다음 예제에서는 attribute_not_exists를 사용하여 Price 속성이 없는 경우에만 제품을 삭제합니다.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_not_exists(Price)"
```

DynamoDB는 attribute_exists 함수도 제공합니다. 다음 예제에서는 좋지 않은 평가를 받은 경우에만 제품을 삭제합니다.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

속성 유형 확인

attribute_type 함수를 사용하여 속성 값의 데이터 형식을 확인할 수 있습니다. 조건 표현식이 true로 평가되는 경우 작업이 성공하고, 그렇지 않으면 작업이 실패합니다.

다음 예제에서는 문자열 집합 형식의 Color 속성이 있는 경우에만 attribute_type을 사용하여 제품을 삭제합니다.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_type(Color, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

--expression-attribute-values의 인수는 expression-attribute-values.json 파일에 저장됩니다.

```
{
    ":v_sub":{"S":"SS"}
}
```

문자열 시작 값 확인

begins_with 함수를 사용하여 문자열 속성 값이 특정 하위 문자열로 시작하는지 확인할 수 있습니다. 조건 표현식이 true로 평가되는 경우 작업이 성공하고, 그렇지 않으면 작업이 실패합니다.

다음 예제에서는 Pictures 맵의 FrontView 요소가 특정 값으로 시작하는 경우에만 begins_with를 사용하여 제품을 삭제합니다.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "begins_with(Pictures.FrontView, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

--expression-attribute-values의 인수는 expression-attribute-values.json 파일에 저장됩니다.

```
{
    ":v_sub":{"S":"http://"}
}
```

세트의 요소 확인

contains 함수를 사용하여 집합의 요소를 확인하거나 문자열 내의 하위 문자열을 찾을 수 있습니다. 조건 표현식이 true로 평가되는 경우 작업이 성공하고, 그렇지 않으면 작업이 실패합니다.

다음 예제에서는 Color 문자열 집합에 특정 값을 가진 요소가 있는 경우에만 contains를 사용하여 제품을 삭제합니다.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "contains(Color, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

--expression-attribute-values의 인수는 expression-attribute-values.json 파일에 저장됩니다.

```
{
    ":v_sub":{"S":"Red"}
}
```

속성 값의 크기 확인

size 함수를 사용하여 속성 값의 크기를 확인할 수 있습니다. 조건 표현식이 true로 평가되는 경우 작업이 성공하고, 그렇지 않으면 작업이 실패합니다.

다음 예제에서는 VideoClip 이진 속성의 크기가 64000바이트보다 큰 경우만 size를 사용하여 제품을 삭제합니다.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "size(VideoClip) > :v_sub" \
    --expression-attribute-values file://expression-attribute-values.json
```

--expression-attribute-values의 인수는 expression-attribute-values.json 파일에 저장됩니다.

```
{
    ":v_sub":{"N":"64000"}
}
```

# DynamoDB에서 TTL(Time To Live) 사용

DynamoDB용 Time To Live(TTL)는 더 이상 관련이 없는 항목을 삭제하기 위한 비용 효율적인 방법입니다. TTL을 사용하면 항목이 더 이상 필요하지 않은 시점을 나타내는 항목별 만료 타임스탬프를 정의할 수 있습니다. DynamoDB는 쓰기 처리량을 소비하지 않고 만료 후 며칠 내에 만료된 항목을 자동으로 삭제합니다.

TTL을 사용하려면 먼저 테이블에서 TTL을 활성화한 다음 TTL 만료 타임스탬프를 저장할 특정 속성을 정의해야 합니다. 타임스탬프는 초 단위로 Unix epoch 시간 형식으로 저장해야 합니다. 항목이 생성되거나 업데이트될 때마다 만료 시간을 계산하여 TTL 속성에 저장할 수 있습니다.

유효하고 만료된 TTL 속성을 가진 항목은 시스템에서 언제든지, 일반적으로 만료 후 며칠 이내에 삭제할 수 있습니다. 삭제 보류 중인 만료된 항목은 여전히 TTL 속성 변경 또는 제거 등의 작업으로 업데이트할 수 있습니다. 만료된 항목을 업데이트할 때는 조건식을 사용하여 해당 항목이 이후에 삭제되지 않도록 하는 것이 좋습니다. 필터 표현식을 사용하여 스캔 및 쿼리 결과에서 만료된 항목을 제거합니다.

삭제된 항목은 일반적인 삭제 작업을 통해 삭제된 항목과 비슷하게 작동합니다. 삭제된 항목은 사용자 삭제 대신 서비스 삭제로 DynamoDB Streams에 저장되며, 다른 삭제 작업과 마찬가지로 로컬 보조 인덱스 및 글로벌 보조 인덱스에서 제거됩니다.

글로벌 테이블의 글로벌 테이블 버전 2019.11.21(현재)을 사용하고 TTL 기능도 사용한다면 DynamoDB는 TTL 삭제를 모든 복제본 테이블에 복제합니다. 최초 TTL 삭제는 TTL 만료가 발생하는 리전의 쓰기 용량 단위(WCU)를 사용하지 않습니다. 그러나 복제 테이블에 대한 복제 TTL 삭제는 각 복제본 리전에서 프로비저닝된 용량을 사용할 때는 복제된 쓰기 용량 단위를 사용하고, 온디맨드 용량 모드를 사용할 때는 복제된 쓰기 단위를 사용하며 요금이 부과됩니다.

TTL에 대한 자세한 내용은 다음 주제를 참조하십시오.

주제


- DynamoDB에서 TTL(Time To Live) 사용
- DynamoDB에서 TTL(Time To Live) 연산
- 만료된 항목 및 TTL(Time To Live) 작업

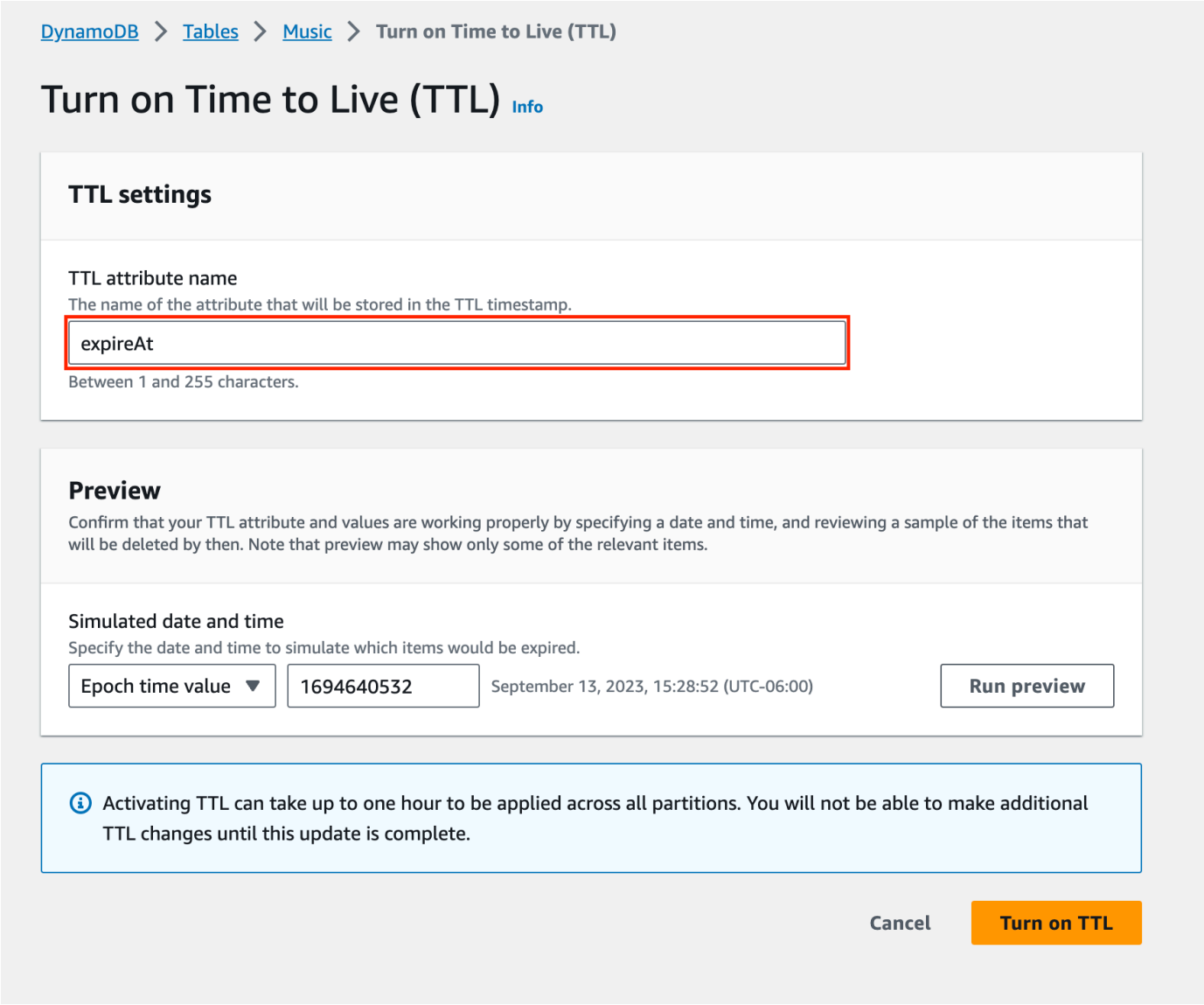
# DynamoDB에서 TTL(Time To Live) 사용

> **Note**
>
> TTL 기능이 제대로 작동하는지 디버깅 및 검증하는 데 도움이 되도록 항목 TTL에 제공된 값이 DynamoDB 진단 로그에 일반 텍스트로 로깅됩니다.

Amazon DynamoDB 콘솔, AWS Command Line Interface(AWS CLI)에서 TTL을 활성화하거나 지원되는 AWS SDK와 함께 Amazon DynamoDB API 참조를 사용하여 TTL을 활성화할 수 있습니다. 모든 파티션에서 TTL을 활성화하는 데 약 1시간이 걸립니다.

AWS 콘솔을 사용하여 DynamoDB TTL 활성화

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 테이블을 선택한 후 수정하려는 테이블을 선택합니다.
3. 추가 설정 탭의 Time To Live(TTL) 섹션에서 켜기를 선택하여 TTL을 활성화합니다.
4. 테이블에서 TTL을 활성화할 경우, DynamoDB에서는 항목이 만료될 수 있는지 여부를 결정할 때 서비스가 검색할 특정 속성 이름을 식별해야 합니다. 아래 표시된 TTL 속성 이름은 대소문자를 구분하며 읽기 및 쓰기 작업에 정의된 속성과 일치해야 합니다. 일치하지 않을 경우 만료된 항목이 삭제되지 않습니다. TTL 속성의 이름을 바꾸려면 TTL을 비활성화했다가 새 속성으로 다시 활성화해야 합니다. TTL은 비활성화된 후에도 약 30분 동안 삭제를 계속 처리합니다. 복원된 테이블에서 TTL을 재구성해야 합니다.




5. (선택 사항) 만료 날짜 및 시간을 시뮬레이션하고 몇 가지 항목을 일치시켜 테스트를 수행할 수 있습니다. 그러면 항목의 샘플 목록이 제공되며 만료 시간과 함께 제공된 TTL 속성 이름을 포함하는 항목이 있는지 확인할 수 있습니다.

TTL이 활성화된 후에 DynamoDB 콘솔에서 항목을 볼 때 TTL 속성이 TTL로 표시되어 있습니다. 속성 위에 마우스 포인터를 놓으면 항목이 만료되는 날짜와 시간을 볼 수 있습니다.

API를 사용하여 DynamoDB TTL 활성화

Python

UpdateTimeToLive 작업을 사용하여 코드로 TTL을 활성화할 수 있습니다.

```
import boto3


def enable_ttl(table_name, ttl_attribute_name):
    """
    Enables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table
    :param ttl_attribute_name: The name of the TTL attribute being provided to the
 table.
    """
    try:
        dynamodb = boto3.client('dynamodb')

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={
                'Enabled': True,
                'AttributeName': ttl_attribute_name
            }
        )

        # In the returned response, check for a successful status code.
        if response['ResponseMetadata']['HTTPStatusCode'] == 200:
            print("TTL has been enabled successfully.")
        else:
            print(f"Failed to enable TTL, status code {response['ResponseMetadata']
['HTTPStatusCode']}")
    except Exception as ex:
        print("Couldn't enable TTL in table %s. Here's why: %s" % (table_name, ex))
        raise


# your values
enable_ttl('your-table-name', 'expirationDate')
```

테이블의 TTL 상태를 설명하는 DescribeTimeToLive 작업을 사용하여 TTL이 활성화되었는지 확인할 수 있습니다. `TimeToLive` 상태는 `ENABLED` 또는 `DISABLED`입니다.

```
# create a DynamoDB client
```

```
dynamodb = boto3.client('dynamodb')

# set the table name
table_name = 'YourTable'

# describe TTL
response = dynamodb.describe_time_to_live(TableName=table_name)
```

### JavaScript

UpdateTimeToLiveCommand 작업을 사용하여 코드로 TTL을 활성화할 수 있습니다.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const enableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
```

```
enableTTL('ExampleTable', 'exampleTtlAttribute');
```

AWS CLI를 사용하여 Time To Live 활성화

1. TTLExample 테이블에서 TTL을 활성화합니다.

```
aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
specification "Enabled=true, AttributeName=ttl"
```

2. TTLExample 테이블에서 TTL을 비활성화합니다.

```
aws dynamodb describe-time-to-live --table-name TTLExample
{
    "TimeToLiveDescription": {
        "AttributeName": "ttl",
        "TimeToLiveStatus": "ENABLED"
    }
}
```

3. BASH 셸 및 AWS CLI를 사용하여 TTLExample 테이블에 유지 시간(TTL) 속성 설정을 갖는 항목을 추가합니다.

```
EXP=`date -d '+5 days' +%s`
aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl":
 {"N": "'$EXP'"}}'
```

이 예제에서는 현재 날짜부터 시작하여 여기에 5일을 추가하여 만료 시간을 생성합니다. 그런 다음 만료 시간을 epoch 시간 형식으로 변환하여 TTLExample 테이블에 항목을 추가합니다.

> **Note**
>
> 유지 시간(TTL)에 만료 값을 설정하는 한 가지 방법은 초 수를 계산하여 만료 시간을 추가하는 것입니다. 예를 들어, 5일은 432,000초입니다. 하지만 날짜로 시작하여 환산하는 방법이 더 많이 사용됩니다.

다음 예제에서와 같이 현재 시간을 epoch 시간 형식으로 변환하는 방법은 간단합니다.

- Linux 터미널: `date +%s`

- Python: `import time; int(time.time())`
- Java: `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

AWS CloudFormation을 사용하여 DynamoDB TTL 활성화

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  TTLExampleTable:
    Type: AWS::DynamoDB::Table
    Description: "A DynamoDB table with TTL Specification enabled"
    Properties:
      AttributeDefinitions:
        - AttributeName: "Album"
          AttributeType: "S"
        - AttributeName: "Artist"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "Album"
          KeyType: "HASH"
        - AttributeName: "Artist"
          KeyType: "RANGE"
      ProvisionedThroughput:
        ReadCapacityUnits: "5"
        WriteCapacityUnits: "5"
      TimeToLiveSpecification:
        AttributeName: "TTLExampleAttribute"
        Enabled: true
```

AWS CloudFormation 템플릿 내에서 TTL을 사용하는 방법에 대한 추가 세부 정보는 여기에서 확인할 수 있습니다.

## DynamoDB에서 TTL(Time To Live) 연산

TTL을 구현하는 일반적인 방법은 항목이 생성되거나 마지막으로 업데이트된 시기를 기준으로 항목의 만료 시간을 설정하는 것입니다. 이를 위해서는 `createdAt` 및 `updatedAt` 타임스탬프에 시간을 추가하면 됩니다. 예를 들어 새로 생성한 항목의 TTL을 `createdAt` +90일로 설정할 수 있습니다. 항목이 업데이트되면 TTL을 `updatedAt` +90일로 다시 계산할 수 있습니다.

계산된 만료 시간은 epoch 형식(초)이어야 합니다. 만료 및 삭제 대상으로 간주되려면 TTL이 과거 5년을 초과해서는 안 됩니다. 다른 형식을 사용하는 경우 TTL 프로세스는 해당 항목을 무시합니다. 만료

날짜를 항목을 만료하려는 미래 날짜로 설정하면 해당 시점이 지나면 항목이 만료됩니다. 예를 들어 만료 날짜를 1724241326(2024년 8월 21일 월요일 11:55:26(GMT))으로 설정했다고 가정해 보겠습니다. 지정된 시간 후 항목이 만료됩니다.

주제

- 항목 생성 및 Time to Live 설정
- 항목 업데이트 및 Time to Live 새로 고침

항목 생성 및 Time to Live 설정

다음 예시는 TTL 속성 이름으로 `expireAt`을 사용하여 새 항목을 만들 때 만료 시간을 계산하는 방법을 보여줍니다. 대입문은 현재 시간을 변수로 가져옵니다. 이 예시에서는 만료 시간을 현재 시간으로부터 90일로 계산합니다. 그러면 시간이 epoch 형식으로 변환되고 TTL 속성에 정수 데이터 유형으로 저장됩니다.

다음 코드 예제는 TTL로 항목을 만드는 방법을 보여줍니다.

Java

SDK for Java 2.x

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

/**
 * Creates an item in a DynamoDB table with TTL attributes.
 * This class demonstrates how to add TTL expiration timestamps to DynamoDB
 items.
```

```
 */
public class CreateTTL {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String CREATION_DATE_ATTR = "creationDate";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String SUCCESS_MESSAGE = "%s PutItem operation with TTL
 successful.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
 DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs a CreateTTL instance with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public CreateTTL(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Constructs a CreateTTL with a default DynamoDB client.
     */
    public CreateTTL() {
        this.dynamoDbClient = null;
    }
```

```
    /**
     * Main method to demonstrate creating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new CreateTTL().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and create an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final CreateTTL createTTL = new CreateTTL(ddb);
            createTTL.createItemWithTTL(tableName, primaryKey, sortKey);
            return 0;
```

```
        } catch (Exception e) {
            throw e;
        }
    }

    /**
     * Creates an item in the specified table with TTL attributes.
     *
     * @param tableName The name of the table
     * @param primaryKeyValue The value for the primary key
     * @param sortKeyValue The value for the sort key
     * @return The response from the PutItem operation
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     */
    public PutItemResponse createItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
sortKeyValue) {
        // Get current time in epoch second format
        final long createDate = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = createDate + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        final Map<String, AttributeValue> itemMap = new HashMap<>();
        itemMap.put(
            PRIMARY_KEY_ATTR,
AttributeValue.builder().s(primaryKeyValue).build());
        itemMap.put(SORT_KEY_ATTR,
AttributeValue.builder().s(sortKeyValue).build());
        itemMap.put(
            CREATION_DATE_ATTR,
            AttributeValue.builder().n(String.valueOf(createDate)).build());
        itemMap.put(
            EXPIRE_AT_ATTR,
            AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final PutItemRequest request =
            PutItemRequest.builder().tableName(tableName).item(itemMap).build();

        try {
            final PutItemResponse response = dynamoDbClient.putItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
```

```
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 PutItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 *
 1000) / 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
```

```
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, err);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 PutItem을 참조하세요.

## Python

### SDK for Python(Boto3)

```
from datetime import datetime, timedelta

import boto3


def create_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Creates a DynamoDB item with an attached expiry attribute.

    :param table_name: Table name for the boto3 resource to target when creating
 an item
    :param region: string representing the AWS region. Example: `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
```

```
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expiration time (90 days from now) in epoch second format
        expiration_time = int((datetime.now() + timedelta(days=90)).timestamp())

        item = {
            "primaryKey": primary_key,
            "sortKey": sort_key,
            "creationDate": current_time,
            "expireAt": expiration_time,
        }
        response = table.put_item(Item=item)

        print("Item created successfully.")
        return response
    except Exception as e:
        print(f"Error creating item: {e}")
        raise e


# Use your own values
create_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 PutItem를 참조하세요.

### 항목 업데이트 및 Time to Live 새로 고침

이 예시는 이전 섹션의 예시에서 이어집니다. 항목이 업데이트되면 만료 시간을 다시 계산할 수 있습니다. 다음 예시에서는 `expireAt` 타임스탬프를 현재 시간으로부터 90일이 되도록 다시 계산합니다.

다음 코드 예제에서는 항목의 TTL을 업데이트하는 방법을 보여줍니다.

Java

SDK for Java 2.x

테이블의 기존 DynamoDB 항목에서 TTL을 업데이트합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

    public UpdateItemResponse updateItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
 sortKeyValue) {
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = new HashMap<>();
        keyMap.put(PRIMARY_KEY_ATTR,
 AttributeValue.builder().s(primaryKeyValue).build());
        keyMap.put(SORT_KEY_ATTR,
 AttributeValue.builder().s(sortKeyValue).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            ":c",
 AttributeValue.builder().n(String.valueOf(currentTime)).build());
        expressionAttributeValues.put(
            ":e",
 AttributeValue.builder().n(String.valueOf(expireDate)).build());
```

```
        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final UpdateItemResponse response =
 dynamoDbClient.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
```

```
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-
value');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

Python

SDK for Python(Boto3)

```
from datetime import datetime, timedelta

import boto3


def update_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Update an existing DynamoDB item with a TTL.
    :param table_name: Name of the DynamoDB table
```

```
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        # Create the DynamoDB resource.
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expireAt time (90 days from now) in epoch second format
        expire_at = int((datetime.now() + timedelta(days=90)).timestamp())

        table.update_item(
            Key={"partitionKey": primary_key, "sortKey": sort_key},
            UpdateExpression="set updatedAt=:c, expireAt=:e",
            ExpressionAttributeValues={":c": current_time, ":e": expire_at},
        )

        print("Item updated successfully.")
    except Exception as e:
        print(f"Error updating item: {e}")


# Replace with your own values
update_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

이 소개에서 설명하는 TTL 예시는 최근에 업데이트된 항목만 테이블에 보관하도록 하는 방법을 보여줍니다. 업데이트된 항목은 수명이 연장되는 반면, 업데이트되지 않은 항목은 생성 후 만료되고 비용 없이 삭제되므로 스토리지에서 차지하는 공간이 줄어들고 테이블이 정리됩니다.

## 만료된 항목 및 TTL(Time To Live) 작업

삭제 보류 중인 만료된 항목은 읽기 및 쓰기 작업에서 필터링할 수 있습니다. 이는 만료된 데이터가 더 이상 유효하지 않아 사용해서는 안 되는 시나리오에서 유용합니다. 필터링되지 않은 경우 백그라운드 프로세스에서 삭제될 때까지 읽기 및 쓰기 작업에 계속 표시됩니다.

> **Note**
>
> 이러한 항목은 삭제되기 전까지는 여전히 스토리지 및 읽기 비용에 포함됩니다.

TTL 삭제는 DynamoDB Streams에서 식별할 수 있지만 삭제가 발생한 리전에서만 식별할 수 있습니다. 글로벌 테이블 리전에 복제된 TTL 삭제는 삭제가 복제되는 리전의 DynamoDB 스트림에서 식별할 수 없습니다.

읽기 작업에서 만료된 항목 필터링

스캔 및 쿼리와 같은 읽기 작업의 경우 필터 표현식을 사용하여 삭제 보류 중인 만료된 항목을 필터링할 수 있습니다. 다음 코드 스니펫에서 볼 수 있듯이 필터 표현식은 TTL 시간이 현재 시간과 같거나 그보다 전인 항목을 필터링할 수 있습니다. 예를 들어, Python SDK 코드에는 현재 시간을 변수(now)로 가져와서 epoch 시간 형식의 경우 `int`로 변환하는 할당 문이 포함되어 있습니다.

다음 코드 예제는 TTL 항목을 쿼리하는 방법을 보여줍니다.

Java

SDK for Java 2.x

필터링된 표현식을 쿼리하여 AWS SDK for Java 2.x를 사용해 DynamoDB 테이블에서 TTL 항목을 수집합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.Map;
import java.util.Optional;
```

```
        final QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final QueryResponse response = ddb.query(request);
            System.out.println("Query successful. Found " + response.count() + "
items that have not expired yet.");

            // Print each item
            response.items().forEach(item -> {
                System.out.println("Item: " + item);
            });

            return 0;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

필터링된 표현식을 쿼리하여 AWS SDK for JavaScript를 사용해 DynamoDB 테이블에서 TTL 항목을 수집합니다.

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";
```

```
export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1')
 => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

필터링된 표현식을 쿼리하여 AWS SDK for Python (Boto3)를 사용해 DynamoDB 테이블에서 TTL 항목을 수집합니다.

```
from datetime import datetime

import boto3


def query_dynamodb_items(table_name, partition_key):
    """

    :param table_name: Name of the DynamoDB table
    :param partition_key:
    :return:
    """
    try:
        # Initialize a DynamoDB resource
        dynamodb = boto3.resource("dynamodb", region_name="us-east-1")

        # Specify your table
        table = dynamodb.Table(table_name)

        # Get the current time in epoch format
        current_time = int(datetime.now().timestamp())

        # Perform the query operation with a filter expression to exclude expired
 items
        # response = table.query(
        #
 KeyConditionExpression=boto3.dynamodb.conditions.Key('partitionKey').eq(partition_key),
        #
 FilterExpression=boto3.dynamodb.conditions.Attr('expireAt').gt(current_time)
        # )
        response = table.query(

 KeyConditionExpression=dynamodb.conditions.Key("partitionKey").eq(partition_key),

 FilterExpression=dynamodb.conditions.Attr("expireAt").gt(current_time),
        )
```

```
        # Print the items that are not expired
        for item in response["Items"]:
            print(item)

    except Exception as e:
        print(f"Error querying items: {e}")


# Call the function with your values
query_dynamodb_items("Music", "your-partition-key-value")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

### 만료된 항목에 조건부 쓰기

조건식을 사용하면 만료된 항목에 대한 쓰기를 방지할 수 있습니다. 아래 코드 스니펫은 만료 시간이 현재 시간보다 큰지를 확인하는 조건부 업데이트입니다. true인 경우 쓰기 작업이 계속됩니다.

다음 코드 예제는 항목의 TTL을 조건부로 업데이트하는 방법을 보여줍니다.

Java

SDK for Java 2.x

조건을 사용하여 테이블의 기존 DynamoDB 항목에서 TTL을 업데이트합니다.

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Map;
import java.util.Optional;
```

```
/**
 * Updates an item in a DynamoDB table with TTL attributes using a conditional
 expression.
 * This class demonstrates how to conditionally update TTL expiration timestamps.
 */
public class UpdateTTLConditional {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String UPDATED_AT_ATTR = "updatedAt";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String UPDATE_EXPRESSION = "SET " + UPDATED_AT_ATTR +
"=:c, " + EXPIRE_AT_ATTR + "=:e";
    private static final String CONDITION_EXPRESSION = "attribute_exists(" +
 PRIMARY_KEY_ATTR + ")";
    private static final String SUCCESS_MESSAGE = "%s UpdateItem operation with
 TTL successful.";
    private static final String CONDITION_FAILED_MESSAGE = "Condition check
 failed. Item does not exist.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
 DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs an UpdateTTLConditional with a default DynamoDB client.
     */
    public UpdateTTLConditional() {
```

```
        this.dynamoDbClient = null;
    }

    /**
     * Constructs an UpdateTTLConditional with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public UpdateTTLConditional(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Main method to demonstrate conditionally updating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new UpdateTTLConditional().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and conditionally update an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
```

```
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = Map.of(
            PRIMARY_KEY_ATTR, AttributeValue.builder().s(primaryKey).build(),
            SORT_KEY_ATTR, AttributeValue.builder().s(sortKey).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = Map.of(
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build(),
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .conditionExpression(CONDITION_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final UpdateItemResponse response = ddb.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return 0;
        } catch (ConditionalCheckFailedException e) {
            System.err.println(CONDITION_FAILED_MESSAGE);
            throw e;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
```

```
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

조건을 사용하여 테이블의 기존 DynamoDB 항목에서 TTL을 업데이트합니다.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItemConditional = async (tableName, partitionKey, sortKey,
 region = 'us-east-1', newAttribute = 'default-value') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
```

```
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-
sort-key-value');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

Python

SDK for Python(Boto3)

조건을 사용하여 테이블의 기존 DynamoDB 항목에서 TTL을 업데이트합니다.

```
from datetime import datetime, timedelta

import boto3
from botocore.exceptions import ClientError


def update_dynamodb_item_ttl(table_name, region, primary_key, sort_key,
 ttl_attribute):
    """
    Updates an existing record in a DynamoDB table with a new or updated TTL
 attribute.

    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
```

```
    :param ttl_attribute: name of the TTL attribute in the target DynamoDB table
    :return:
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Generate updated TTL in epoch second format
        updated_expiration_time = int((datetime.now() +
 timedelta(days=90)).timestamp())

        # Define the update expression for adding/updating a new attribute
        update_expression = "SET newAttribute = :val1"

        # Define the condition expression for checking if 'expireAt' is not
 expired
        condition_expression = "expireAt > :val2"

        # Define the expression attribute values
        expression_attribute_values = {":val1": ttl_attribute, ":val2":
 updated_expiration_time}

        response = table.update_item(
            Key={"primaryKey": primary_key, "sortKey": sort_key},
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeValues=expression_attribute_values,
        )

        print("Item updated successfully.")
        return response["ResponseMetadata"]["HTTPStatusCode"]  # Ideally a 200 OK
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            print("Condition check failed: Item's 'expireAt' is expired.")
        else:
            print(f"Error updating item: {e}")
    except Exception as e:
        print(f"Error updating item: {e}")


# replace with your values
update_dynamodb_item_ttl(
    "your-table-name",
    "us-east-1",
```

```
    "your-partition-key-value",
    "your-sort-key-value",
    "your-ttl-attribute-value",
)
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

DynamoDB Streams에서 삭제된 항목 식별

스트림 레코드에는 사용자 ID 필드 `Records[<index>].userIdentity`가 포함되어 있습니다. TTL 프로세스에 의해 삭제된 항목은 다음 필드를 갖습니다.

```
Records[<index>].userIdentity.type
"Service"

Records[<index>].userIdentity.principalId
"dynamodb.amazonaws.com"
```

다음 JSON은 단일 스트림 레코드의 해당 부분을 보여줍니다.

```
"Records": [
  {
 ...
  "userIdentity": {
  "type": "Service",
       "principalId": "dynamodb.amazonaws.com"
    }
   ...
 }
]
```

## DynamoDB에서 테이블 쿼리

Amazon DynamoDB에서 `Query` API 작업을 사용하여 프라이머리 키 값을 기반으로 항목을 찾을 수 있습니다.

파티션 키 속성의 이름과 해당 속성의 단일 값을 제공해야 합니다. `Query`는 해당 파티션 키 값을 갖는 모든 항목을 반환합니다. 선택에 따라 정렬 키 속성을 제공하고 비교 연산자를 사용하여 검색 결과의 범위를 좁힐 수 있습니다.

요청 구문, 응답 파라미터 및 추가 예제와 같은 Query 사용 방법에 대한 자세한 내용은 Amazon DynamoDB API 참조의 쿼리를 참조하세요.

주제

- DynamoDB에서의 쿼리 작업에 대한 키 조건 표현식
- DynamoDB에서의 쿼리 작업에 대한 필터 표현식
- DynamoDB의 테이블 쿼리 결과 페이지 매김
- DynamoDB에서의 쿼리 작업에 대한 기타 측면

## DynamoDB에서의 쿼리 작업에 대한 키 조건 표현식

키 조건 표현식에서는 첫 번째 문자가 a-z 또는 A-Z이고 나머지 문자(두 번째 문자부터 시작, 있는 경우)가 a-z, A-Z 또는 0-9인 경우에 한해 속성 이름은 아무거나 사용할 수 있습니다. 또한 속성 이름이 DynamoDB 예약어가 되어서는 안 됩니다. (예약어 전체 목록은 DynamoDB의 예약어 단원을 참조하세요.) 속성 이름이 이러한 요건을 만족하지 않으면 표현식 속성 이름을 자리 표시자로 정의해야 합니다. 자세한 내용은 DynamoDB의 표현식 속성 이름(별칭) 섹션을 참조하세요.

일정한 파티션 키 값을 갖는 항목들에 대해서는 DynamoDB가 정렬 키 값을 기준으로 순서를 정렬하여 모두 함께 저장합니다. Query 작업을 할 때는 DynamoDB는 정렬된 순서대로 항목을 가져온 다음 FilterExpression과 모든 KeyConditionExpression(있는 경우)을 사용해 항목을 처리합니다. 그러면 클라이언트에게는 Query 결과만 다시 보내집니다.

Query 작업은 항상 결과 집합을 반환합니다. 일치하는 항목이 없다면 결과 집합은 비어 있습니다.

Query 결과는 항상 정렬 키 값을 기준으로 정렬됩니다. 정렬 키의 데이터 형식이 Number이면 결과가 숫자 순서대로 반환됩니다. 그렇지 않으면 결과가 UTF-8 바이트 순서로 반환됩니다. 기본적으로 정렬 순서는 오름차순입니다. 오름차순을 역순으로 바꾸려면 ScanIndexForward 파라미터를 false로 설정하면 됩니다.

단일 Query 작업은 최대 1MB의 데이터를 가져올 수 있습니다. 이러한 크기 제한은 FilterExpression 또는 ProjectionExpression이 결과에 반영되기 전에 적용됩니다. 응답에 LastEvaluatedKey가 존재하고 null이 아니라면 결과 집합을 페이지 매김해야 합니다(DynamoDB의 테이블 쿼리 결과 페이지 매김 참조).

키 조건 표현식 예제

검색 기준을 지정하려면 테이블 또는 인덱스에서 읽을 항목을 결정하는 문자열인 키 조건 표현식을 사용합니다.

파티션 키 이름 및 값은 등식 조건으로 지정해야 합니다. 키 조건 표현식에는 키가 아닌 속성을 사용할 수 없습니다.

정렬 키의 두 번째 조건(있는 경우)은 옵션으로 입력할 수 있습니다. 단, 정렬 키 조건은 다음 중 한 가지 비교 연산자를 사용해야 합니다.

- *a* = *b* - 속성 *a*가 값 *b*와 같은 경우 true
- *a* < *b* - *a*가 *b*보다 작은 경우 true
- *a* <= *b* - *a*가 *b*보다 작거나 같은 경우 true
- *a* > *b* - *a*가 *b*보다 큰 경우 true
- *a* >= *b* - *a*가 *b*보다 크거나 같은 경우 true
- *a* BETWEEN *b* AND *c* - *a*가 *b*보다 크거나 같고 *c*보다 작거나 같은 경우 true

다음 함수도 지원됩니다.

- begins_with (*a*, *substr*) - *a* 속성 값이 특정 하위 문자열로 시작하는 경우 true

다음 AWS Command Line Interface(AWS CLI) 예제는 키 조건 표현식의 사용을 보여 줍니다. 이 표현식들은 실제 값이 아닌 자리 표시자(:name 및 :sub)를 사용합니다. 자세한 내용은 DynamoDB의 표현식 속성 이름(별칭) 및 DynamoDB에서 표현식 속성 값 사용 단원을 참조하세요.

Example

Thread 테이블에 대한 쿼리를 실행하여 특정 ForumName(파티션 키)을 찾습니다. 쿼리 결과에 따라 ForumName 값을 갖는 항목을 모두 읽어옵니다. 정렬 키(Subject)가 KeyConditionExpression에 추가되지 않았기 때문입니다.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name" \
    --expression-attribute-values  '{":name":{"S":"Amazon DynamoDB"}}'
```

Example

Thread 테이블에 대한 쿼리를 실행하여 특정 ForumName(파티션 키)을 찾습니다. 하지만 이번에는 Subject(정렬 키)를 갖는 항목만 반환됩니다.

```
aws dynamodb query \
```

```
    --table-name Thread \
    --key-condition-expression "ForumName = :name and Subject = :sub" \
    --expression-attribute-values  file://values.json
```

`--expression-attribute-values`의 인수는 `values.json` 파일에 저장됩니다.

```
{
    ":name":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"}
}
```

Example

`Reply` 테이블에 대해 쿼리를 실행하여 특정 `Id`(파티션 키)를 찾습니다. 하지만 `ReplyDateTime`(정렬 키)이 몇 가지 문자로 시작되는 항목만 반환됩니다.

```
aws dynamodb query \
    --table-name Reply \
    --key-condition-expression "Id = :id and begins_with(ReplyDateTime, :dt)" \
    --expression-attribute-values  file://values.json
```

`--expression-attribute-values`의 인수는 `values.json` 파일에 저장됩니다.

```
{
    ":id":{"S":"Amazon DynamoDB#DynamoDB Thread 1"},
    ":dt":{"S":"2015-09"}
}
```

## DynamoDB에서의 쿼리 작업에 대한 필터 표현식

`Query` 결과를 한층 더 좁혀야 하는 경우 선택적으로 필터 표현식을 제공할 수 있습니다. 필터 표현식은 `Query` 결과 내에서 어떤 항목을 반환할지를 결정합니다. 다른 모든 결과는 폐기됩니다.

필터 표현식은 `Query`이 완료된 후 결과가 반환되기 전에 적용됩니다. 따라서 필터 표현식이 있는지 여부와 상관없이 `Query`은 동일한 양의 읽기 용량을 사용합니다.

`Query` 작업은 최대 1MB의 데이터를 가져올 수 있습니다. 이 제한은 필터 표현식이 평가되기 전에 적용됩니다.

필터 표현식에는 파티션 키 또는 정렬 키 속성이 포함될 수 없습니다. 필터 표현식이 아닌 키 조건 표현식에 있는 속성을 지정해야 합니다.

필터 표현식의 구문은 키 조건 표현식의 구문과 유사합니다. 필터 표현식에는 키 조건 표현식과 동일한 비교기, 함수 및 논리적 연산자를 사용할 수 있습니다. 또한 필터 표현식에는 같지 않음 연산자(<>), OR 연산자, CONTAINS 연산자, IN 연산자, BEGINS_WITH 연산자, BETWEEN 연산자, EXISTS 연산자 및 SIZE 연산자를 사용할 수 있습니다. 자세한 내용은 DynamoDB에서의 쿼리 작업에 대한 키 조건 표현식 및 필터 및 조건 표현식 구문 단원을 참조하세요.

Example

다음 AWS CLI 예제에서는 Thread 테이블에 대해 쿼리를 실행하여 특정 ForumName(파티션 키)과 Subject(정렬 키)를 찾아봅니다. 찾은 항목 중에서 가장 인기 있는 토론 스레드가 반환됩니다. 즉, 일정 수 이상의 Views가 있는 스레드만 반환됩니다.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :fn and Subject begins_with :sub" \
    --filter-expression "#v >= :num" \
    --expression-attribute-names '{"#v": "Views"}' \
    --expression-attribute-values file://values.json
```

--expression-attribute-values의 인수는 values.json 파일에 저장됩니다.

```
{
    ":fn":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"},
    ":num":{"N":"3"}
}
```

Views는 DynamoDB에서 예약어이므로(DynamoDB의 예약어 참조) 이 예제에서는 #v를 자리 표시자로 사용합니다. 자세한 내용은 DynamoDB의 표현식 속성 이름(별칭) 섹션을 참조하세요.

> **Note**
>
> 필터 표현식은 Query 결과 집합에서 항목을 제거합니다. 많은 수의 항목을 반환할 것으로 예상되지만 해당 항목 중 대부분을 폐기해야 하는 경우 가능하다면 Query를 사용하지 마세요.

## DynamoDB의 테이블 쿼리 결과 페이지 매김

DynamoDB는 Query 작업 결과의 페이지를 매깁니다. 페이지를 매기면 Query 결과는 크기가 1MB 이하인 데이터 '페이지'로 분리됩니다. 애플리케이션은 결과의 첫 번째 페이지를 처리한 다음 두 번째 페이지를 처리하고 이런 식으로 계속할 수 있습니다.

단일 Query는 1MB 크기 한도 내에 맞는 결과 집합만 반환합니다. 추가 결과가 있는지 확인하고 이러한 결과를 한번에 한 페이지에 가져오려면 애플리케이션에서 다음을 수행해야 합니다.

1. 하위 수준 Query 결과를 확인합니다.
   - 결과가 LastEvaluatedKey 요소를 포함하고 null이 아닌 경우 2단계로 계속합니다.
   - 결과에 LastEvaluatedKey가 없는 경우 더 이상 가져올 항목이 없습니다.
2. 동일한 KeyConditionExpression을 사용하여 Query를 구성합니다. 그러나 이번에는 1단계에서 LastEvaluatedKey 값을 가져와서 새로운 Query 요청의 ExclusiveStartKey 파라미터로 사용합니다.
3. 새로운 Query 요청을 실행합니다.
4. 1단계로 이동합니다.

다시 말해서, Query 응답의 LastEvaluatedKey를 다음 Query 요청에 대한 ExclusiveStartKey로 사용해야 합니다. Query 응답에 LastEvaluatedKey 요소가 없는 경우 결과의 최종 페이지를 검색한 것입니다. LastEvaluatedKey가 비어 있지 않는 경우 결과 집합에 데이터가 더 있음을 의미하는 것은 아닙니다. 결과 집합의 마지막 페이지를 알 수 있는 유일한 방법은 LastEvaluatedKey가 비어 있을 때입니다.

AWS CLI를 사용하여 이 동작을 볼 수 있습니다. AWS CLI는 LastEvaluatedKey가 결과에 더 이상 없을 때까지 하위 수준 Query 요청을 DynamoDB에 반복적으로 보냅니다. 특정 연도의 영화 제목을 검색하는 다음 AWS CLI 예제를 살펴보세요.

```
aws dynamodb query --table-name Movies \
    --projection-expression "title" \
    --key-condition-expression "#y = :yyyy" \
    --expression-attribute-names '{"#y":"year"}' \
    --expression-attribute-values '{":yyyy":{"N":"1993"}}' \
    --page-size 5 \
    --debug
```

일반적으로 AWS CLI에서는 페이지 매김이 자동으로 처리됩니다. 그러나 이 예제에서는 AWS CLI `--page-size` 파라미터가 페이지당 항목 수를 제한합니다. `--debug` 파라미터는 요청 및 응답에 대한 하위 수준 정보를 출력합니다.

예제를 실행하면 DynamoDB의 첫 응답이 다음과 유사합니다.

```
2017-07-07 11:13:15,603 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":5,"Items":[{"title":{"S":"A Bronx Tale"}},
{"title":{"S":"A Perfect World"}},{"title":{"S":"Addams Family Values"}},
{"title":{"S":"Alive"}},{"title":{"S":"Benny & Joon"}}],
"LastEvaluatedKey":{"year":{"N":"1993"},"title":{"S":"Benny & Joon"}},
"ScannedCount":5}'
```

응답의 `LastEvaluatedKey`는 가져온 항목이 전부가 아님을 나타냅니다. 그러면 AWS CLI는 DynamoDB에 다른 `Query` 요청을 보냅니다. 이 요청과 응답 패턴은 마지막 응답이 반환될 때까지 계속됩니다.

```
2017-07-07 11:13:16,291 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"What\'s Eating Gilbert
 Grape"}}],"ScannedCount":1}'
```

`LastEvaluatedKey`가 없으면 가져올 항목이 더 이상 없음을 나타냅니다.

> **Note**
>
> AWS SDK는 하위 수준 DynamoDB 응답(`LastEvaluatedKey`의 유무 포함)을 처리하여 `Query` 결과 페이지 매김에 대해 다양한 추상을 제공합니다. 예를 들어, SDK for Java 문서 인터페이스는 `java.util.Iterator` 지원을 제공하므로 한 번에 하나씩 결과를 볼 수 있습니다.
> 다양한 프로그래밍 언어의 코드 예제를 보려면 Amazon DynamoDB 시작 안내서 및 해당 언어의 AWS SDK 설명서를 참조하세요.

`Query` 작업의 `Limit` 파라미터를 사용하여 결과 세트의 항목 수를 제한하여 페이지 크기를 줄일 수도 있습니다.

DynamoDB를 사용한 쿼리 방법에 대한 자세한 내용은 DynamoDB에서 테이블 쿼리 단원을 참조하세요.

## DynamoDB에서의 쿼리 작업에 대한 기타 측면

이 섹션에서는 결과 크기 제한, 스캔된 항목과 반환된 항목 수 계산 비교, 읽기 용량 소비 모니터링, 읽기 일관성 제어 등 DynamoDB 쿼리 작업의 추가 측면을 다룹니다.

### 결과 세트의 항목 수 제한

`Query` 작업을 사용하여 읽는 항목 수를 제한할 수 있습니다. 이렇게 하려면 `Limit` 파라미터를 원하는 최대 항목 수로 설정합니다.

예를 들어 `Limit` 값이 6이고 필터 표현식이 없는 상태에서 테이블을 `Query`한다고 가정합니다. `Query` 결과에는 테이블에서 요청의 키 조건 표현식과 일치하는 처음 6개의 항목이 포함됩니다.

이제 필터 표현식을 `Query`에 추가한다고 가정합니다. 이 경우 DynamoDB는 최대 6개의 항목을 읽은 다음 필터 표현식과 일치하는 항목만 반환합니다. DynamoDB에서 더 많은 항목을 계속 읽은 경우 필터 표현식과 일치하는 항목이 더 많더라도 최종 `Query` 결과에는 여섯 개 이하의 항목이 포함됩니다.

### 결과 내 항목 수 계산

`Query` 응답에는 기준과 일치하는 항목 외에도 다음 요소가 포함됩니다.

- `ScannedCount` - 필터 표현식(있는 경우)을 적용하기 전에 키 조건 표현식과 일치한 항목 수입니다.
- `Count` - 필터 표현식(있는 경우)을 적용한 후에 남아 있는 항목 수입니다.

> **Note**
>
> 필터 표현식을 사용하지 않으면 `ScannedCount`와 `Count`는 동일한 값을 갖습니다.

`Query` 결과 집합의 크기가 1MB보다 크면 `ScannedCount`와 `Count`는 총 항목 수의 일부만 표시합니다. 모든 결과를 검색하려면 여러 `Query` 작업을 수행해야 합니다(DynamoDB의 테이블 쿼리 결과 페이지 매김 참조).

각 `Query` 응답에는 해당하는 특정 `Query` 요청에 따라 처리된 항목의 `ScannedCount`와 `Count`가 포함됩니다. 모든 `Query` 요청의 총계를 얻으려면 `ScannedCount`와 `Count`의 누계를 계산할 수 있습니다.

### 쿼리에서 사용된 용량 단위

파티션 키 속성의 이름과 해당 속성의 단일 값을 제공하기만 하면 모든 테이블 또는 보조 인덱스를 `Query`할 수 있습니다. `Query`는 해당 파티션 키 값을 갖는 모든 항목을 반환합니다. 필요에 따라 정렬

키 속성을 제공하고 비교 연산자를 사용하여 검색 결과의 범위를 좁힐 수 있습니다. Query API 작업은 다음과 같은 읽기 용량 단위를 사용합니다.

| 다음을 **Query**하는 경우... | DynamoDB는 다음에서 읽기 용량 단위를 사용합니다... |
|---|---|
| 표 | 테이블의 할당된 읽기 용량. |
| 글로벌 보조 인덱스 | 인덱스의 할당된 읽기 용량. |
| 로컬 보조 인덱스 | 기본 테이블의 프로비저닝된 읽기 용량. |

기본적으로, Query 작업은 얼마나 많은 읽기 용량을 사용하는지에 대한 데이터를 반환하지 않습니다. 하지만 Query 요청에서 ReturnConsumedCapacity 파라미터를 지정하여 이 정보를 얻을 수 있습니다. 다음은 ReturnConsumedCapacity에 대한 유효한 설정입니다.

- NONE - 사용된 용량 데이터가 반환되지 않습니다. (이 값이 기본값입니다.)
- TOTAL - 사용된 읽기 용량 단위의 집계 수가 응답에 포함됩니다.
- INDEXES - 액세스한 각 테이블 및 인덱스에 사용된 용량과 함께 사용된 읽기 용량 단위의 집계 수가 응답에 표시됩니다.

DynamoDB는 애플리케이션에 반환되는 데이터 양이 아닌 항목 개수와 항목 크기를 기준으로 사용된 읽기 용량 단위의 수를 계산합니다. 이러한 이유로, 모든 속성을 요청하든(기본 동작) 또는 일부 속성만 요청하든(프로젝션 표현식 사용) 상관없이 사용된 용량 단위의 수는 동일합니다. 필터 표현식 사용 여부와 관계없이 숫자는 동일합니다. Query는 최대 4KB의 항목에 대해 초당 강력히 일관된 읽기 1회 또는 초당 최종적으로 일관된 읽기 2회를 수행하기 위해 최소한의 읽기 용량 단위를 사용합니다. 보다 큰 항목을 읽어야 하는 경우, DynamoDB에 추가 읽기 요청 단위가 필요합니다. 파티션 키의 양이 적은 빈 테이블과 매우 큰 테이블에서는 쿼리된 데이터 양을 초과하여 일부 추가 RCU에 요금이 부과될 수 있습니다. 여기에는 데이터가 없는 경우에도 Query 요청을 처리하는 데 드는 비용이 포함됩니다.

쿼리에 대한 읽기 정합성

기본적으로 Query 작업은 최종적 일관된 읽기를 수행합니다. 따라서 최근 완료된 PutItem 또는 UpdateItem 작업으로 인해 Query 결과에 변경 사항이 반영되지 않을 수 있습니다. 자세한 내용은 DynamoDB 읽기 일관성 섹션을 참조하세요.

강력한 일관된 읽기가 필요한 경우 Query 요청에서 ConsistentRead 매개 변수를 true로 설정합니다.

# DynamoDB에서 테이블 스캔

Amazon DynamoDB의 Scan 작업은 테이블 또는 보조 인덱스의 모든 항목을 읽어옵니다. 기본적으로 Scan 작업은 테이블이나 인덱스에 속한 항목의 데이터 속성을 모두 반환합니다. 하지만 Scan 작업에서 ProjectionExpression 파라미터를 사용하면 모두가 아닌 일부 속성만 가져올 수 있습니다.

Scan은 항상 결과 집합을 반환합니다. 일치하는 항목이 없다면 결과 집합은 비어 있습니다.

단일 Scan 요청으로 최대 1MB의 데이터를 가져올 수 있습니다. 선택에 따라 DynamoDB가 이 데이터에 필터 표현식을 적용하여 사용자에게 반환되기 전에 결과의 범위를 좁힐 수 있습니다.

주제

- 스캔에 대한 필터 표현식
- 결과 세트의 항목 수 제한
- 결과 페이지 매김
- 결과 내 항목 수 계산
- 스캔에서 사용된 용량 단위
- 스캔에 대한 읽기 정합성
- 병렬 스캔

## 스캔에 대한 필터 표현식

Scan 결과를 한층 더 좁혀야 하는 경우 선택적으로 필터 표현식을 제공할 수 있습니다. 필터 표현식은 Scan 결과 내에서 어떤 항목을 반환할지를 결정합니다. 다른 모든 결과는 폐기됩니다.

필터 표현식은 Scan이 완료된 후 결과가 반환되기 전에 적용됩니다. 따라서 필터 표현식이 있는지 여부와 상관없이 Scan은 동일한 양의 읽기 용량을 사용합니다.

Scan 작업은 최대 1MB의 데이터를 가져올 수 있습니다. 이 제한은 필터 표현식이 평가되기 전에 적용됩니다.

Scan을 사용하면 필터 표현식에 파티션 키와 정렬 키 속성을 포함하여 모든 속성을 지정할 수 있습니다.

필터 표현식의 구분은 조건 표현식의 구문과 같습니다. 필터 표현식은 동일한 비교기, 함수 및 논리적 연산자를 조건 표현식으로 사용할 수 있습니다. 논리 연산자에 대한 자세한 내용은 DynamoDB의 조건 및 필터 표현식, 연산자, 함수 섹션을 참조하세요.

Example

다음 AWS Command Line Interface(AWS CLI) 예제에서는 Thread 테이블을 스캔하여 특정 사용자가 마지막으로 게시한 항목만 반환합니다.

```
aws dynamodb scan \
     --table-name Thread \
     --filter-expression "LastPostedBy = :name" \
     --expression-attribute-values '{":name":{"S":"User A"}}'
```

## 결과 세트의 항목 수 제한

Scan 작업을 사용하여 결과에 반환되는 항목 수를 제한할 수 있습니다. 이렇게 하려면 필터 표현식 평가 전에 Limit 파라미터를 Scan 작업에서 반환할 최대 항목 수로 설정합니다.

예를 들어 Limit 값이 6이고 필터 표현식이 없는 상태에서 테이블을 Scan한다고 가정합니다. Scan 결과에 테이블의 처음 6개 항목이 포함됩니다.

이제 필터 표현식을 Scan에 추가한다고 가정합니다. 이 경우 DynamoDB는 반환된 6개 항목에 필터 표현식을 적용하고 일치하지 않는 항목을 무시합니다. 최종 Scan 결과에는 필터링된 항목 수에 따라 6개 이하의 항목이 포함됩니다.

## 결과 페이지 매김

DynamoDB는 Scan 작업 결과의 페이지를 매깁니다. 페이지를 매기면 Scan 결과는 크기가 1MB 이하인 데이터 '페이지'로 분리됩니다. 애플리케이션은 결과의 첫 번째 페이지를 처리한 다음 두 번째 페이지를 처리하고 이런 식으로 계속할 수 있습니다.

단일 Scan은 1MB 크기 한도 내에 맞는 결과 집합만 반환합니다.

추가 결과가 있는지 확인하고 이러한 결과를 한번에 한 페이지에 가져오려면 애플리케이션에서 다음을 수행해야 합니다.

1. 하위 수준 Scan 결과를 확인합니다.
   - 결과에 LastEvaluatedKey 요소가 포함되는 경우 2단계로 계속합니다.
   - 결과에 LastEvaluatedKey가 없는 경우 더 이상 가져올 항목이 없습니다.

2. 이전과 동일한 파라미터를 이용해 새로운 Scan 요청을 구성합니다. 그러나 이번에는 1단계에서 LastEvaluatedKey 값을 가져와서 새로운 Scan 요청의 ExclusiveStartKey 파라미터로 사용합니다.
3. 새로운 Scan 요청을 실행합니다.
4. 1단계로 이동합니다.

다시 말해서, Scan 응답의 LastEvaluatedKey를 다음 Scan 요청에 대한 ExclusiveStartKey로 사용해야 합니다. Scan 응답에 LastEvaluatedKey 요소가 없는 경우 결과의 최종 페이지를 검색한 것입니다. 결과 집합의 마지막에 도달했음을 알 수 있는 유일한 방법은 LastEvaluatedKey가 없는지 확인하는 것입니다.

AWS CLI를 사용하여 이 동작을 볼 수 있습니다. AWS CLI는 LastEvaluatedKey가 결과에 더 이상 없을 때까지 하위 수준 Scan 요청을 DynamoDB에 반복적으로 보냅니다. 전체 Movies 테이블을 스캔하여 특정 장르의 영화만 반환하는 다음 AWS CLI 예제를 살펴보겠습니다.

```
aws dynamodb scan \
    --table-name Movies \
    --projection-expression "title" \
    --filter-expression 'contains(info.genres,:gen)' \
    --expression-attribute-values '{":gen":{"S":"Sci-Fi"}}' \
    --page-size 100  \
    --debug
```

일반적으로 AWS CLI에서는 페이지 매김이 자동으로 처리됩니다. 그러나 이 예제에서는 AWS CLI --page-size 파라미터가 페이지당 항목 수를 제한합니다. --debug 파라미터는 요청 및 응답에 대한 하위 수준 정보를 출력합니다.

> **Note**
>
> 페이지 매김 결과는 전달한 입력 매개 변수에 따라 달라집니다.
>
> - aws dynamodb scan --table-name Prices --max-items 1 사용 시 NextToken을(를) 반환합니다.
> - aws dynamodb scan --table-name Prices --limit 1 사용 시 LastEvaluatedKey을(를) 반환합니다.
>
> 특히 --starting-token 사용 시 NextToken 값이 필요하다는 점에 유의하세요.

예제를 실행하면 DynamoDB의 첫 응답이 다음과 유사합니다.

```
2017-07-07 12:19:14,389 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":7,"Items":[{"title":{"S":"Monster on the Campus"}},{"title":{"S":"+1"}},
{"title":{"S":"100 Degrees Below Zero"}},{"title":{"S":"About Time"}},{"title":
{"S":"After Earth"}},
{"title":{"S":"Age of Dinosaurs"}},{"title":{"S":"Cloudy with a Chance of Meatballs
 2"}}],
"LastEvaluatedKey":{"year":{"N":"2013"},"title":{"S":"Curse of
 Chucky"}},"ScannedCount":100}'
```

응답의 LastEvaluatedKey는 가져온 항목이 전부가 아님을 나타냅니다. 그러면 AWS CLI는 DynamoDB에 다른 Scan 요청을 보냅니다. 이 요청과 응답 패턴은 마지막 응답이 반환될 때까지 계속됩니다.

```
2017-07-07 12:19:17,830 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"WarGames"}}],"ScannedCount":6}'
```

LastEvaluatedKey가 없으면 가져올 항목이 더 이상 없음을 나타냅니다.

> **Note**
>
> AWS SDK는 하위 수준 DynamoDB 응답(LastEvaluatedKey의 유무 포함)을 처리하여 Scan 결과 페이지 매김에 대해 다양한 추상을 제공합니다. 예를 들어, SDK for Java 문서 인터페이스는 java.util.Iterator 지원을 제공하므로 한 번에 하나씩 결과를 볼 수 있습니다. 다양한 프로그래밍 언어의 코드 예제를 보려면 Amazon DynamoDB 시작 안내서 및 해당 언어의 AWS SDK 설명서를 참조하세요.

## 결과 내 항목 수 계산

Scan 응답에는 기준과 일치하는 항목 외에도 다음 요소가 포함됩니다.

- ScannedCount - ScanFilter를 적용하기 전에 평가되는 항목 수입니다. ScannedCount 값이 높지만 Count 결과가 거의 없거나 전혀 없는 경우 Scan 작업이 비효율적이라는 것을 나타냅니다. 요청에 필터를 사용하지 않은 경우 ScannedCount는 Count와 동일합니다.
- Count - 필터 표현식(있는 경우)을 적용한 후에 남아 있는 항목 수입니다.

> **Note**
>
> 필터 표현식을 사용하지 않으면 ScannedCount와 Count는 동일한 값을 갖습니다.

Scan 결과 집합의 크기가 1MB보다 크면 ScannedCount와 Count는 총 항목 수의 일부만 표시합니다. 모든 결과를 검색하려면 여러 Scan 작업을 수행해야 합니다(결과 페이지 매김 참조).

각 Scan 응답에는 해당하는 특정 Scan 요청에 따라 처리된 항목의 ScannedCount와 Count가 포함됩니다. 모든 Scan 요청의 총계를 가져오려면 ScannedCount와 Count의 누계를 계산할 수 있습니다.

### 스캔에서 사용된 용량 단위

테이블이나 보조 인덱스를 Scan할 수 있습니다. Scan 작업은 다음과 같이 읽기 용량 단위를 사용합니다.

| 다음을 **Scan**하는 경우... | DynamoDB는 다음에서 읽기 용량 단위를 사용합니다... |
|---|---|
| 표 | 테이블의 할당된 읽기 용량. |
| 글로벌 보조 인덱스 | 인덱스의 할당된 읽기 용량. |
| 로컬 보조 인덱스 | 기본 테이블의 프로비저닝된 읽기 용량. |

> **Note**
>
> 보조 인덱스 스캔 작업에 대한 교차 계정 액세스는 현재 리소스 기반 정책에서 지원되지 않습니다.

기본적으로, Scan 작업은 얼마나 많은 읽기 용량을 사용하는지에 대한 데이터를 반환하지 않습니다. 하지만 Scan 요청에서 ReturnConsumedCapacity 파라미터를 지정하여 이 정보를 얻을 수 있습니다. 다음은 ReturnConsumedCapacity에 대한 유효한 설정입니다.

- NONE - 사용된 용량 데이터가 반환되지 않습니다. (이 값이 기본값입니다.)
- TOTAL - 사용된 읽기 용량 단위의 집계 수가 응답에 포함됩니다.

- INDEXES - 액세스한 각 테이블 및 인덱스에 사용된 용량과 함께 사용된 읽기 용량 단위의 집계 수가 응답에 표시됩니다.

DynamoDB는 애플리케이션에 반환되는 데이터 양이 아닌 항목 개수와 항목 크기를 기준으로 사용된 읽기 용량 단위의 수를 계산합니다. 이러한 이유로, 모든 속성을 요청하든(기본 동작) 또는 일부 속성만 요청하든(프로젝션 표현식 사용) 상관없이 사용된 용량 단위의 수는 동일합니다. 필터 표현식 사용 여부와 관계없이 숫자는 동일합니다. Scan는 최대 4KB의 항목에 대해 초당 강력히 일관된 읽기 1회 또는 초당 최종적으로 일관된 읽기 2회를 수행하기 위해 최소한의 읽기 용량 단위를 사용합니다. 보다 큰 항목을 읽어야 하는 경우, DynamoDB에 추가 읽기 요청 단위가 필요합니다. 파티션 키의 양이 적은 빈 테이블과 매우 큰 테이블에서는 스캔된 데이터 양을 초과하여 일부 추가 RCU에 요금이 부과될 수 있습니다. 여기에는 데이터가 없는 경우에도 Scan 요청을 처리하는 데 드는 비용이 포함됩니다.

## 스캔에 대한 읽기 정합성

기본적으로 Scan 작업은 최종적 일관된 읽기를 수행합니다. 따라서 최근 완료된 PutItem 또는 UpdateItem 작업으로 인해 Scan 결과에 변경 사항이 반영되지 않을 수 있습니다. 자세한 내용은 DynamoDB 읽기 일관성 섹션을 참조하세요.

Scan이 시작되는 시점을 기준으로 강력한 일관된 읽기가 필요한 경우 Scan 요청에서 ConsistentRead 파라미터를 true로 설정합니다. 그러면 Scan 작업 시작 전에 마친 쓰기 작업까지 모두 Scan 응답에 포함됩니다.

ConsistentRead를 true로 설정하면 DynamoDB Streams를 함께 사용하는 테이블 백업 또는 복제 시나리오에서 유용할 수 있습니다. 처음에는 ConsistentRead을 true로 설정한 상태에서 Scan를 사용하여 테이블 데이터에 대한 일관된 복사본을 얻습니다. 그러면 DynamoDB Streams가 Scan 작업 중에 테이블에서 추가로 발행하는 쓰기 활동을 기록합니다. Scan 작업을 마치면 스트림의 쓰기 연산을 테이블에 적용할 수 있습니다.

> **Note**
>
> ConsistentRead를 true로 설정한 상태에서 Scan 작업을 수행하면 ConsistentRead를 기본값(false)으로 그대로 두는 경우와 비교할 때 2배 더 많은 읽기 용량 단위를 사용합니다.

## 병렬 스캔

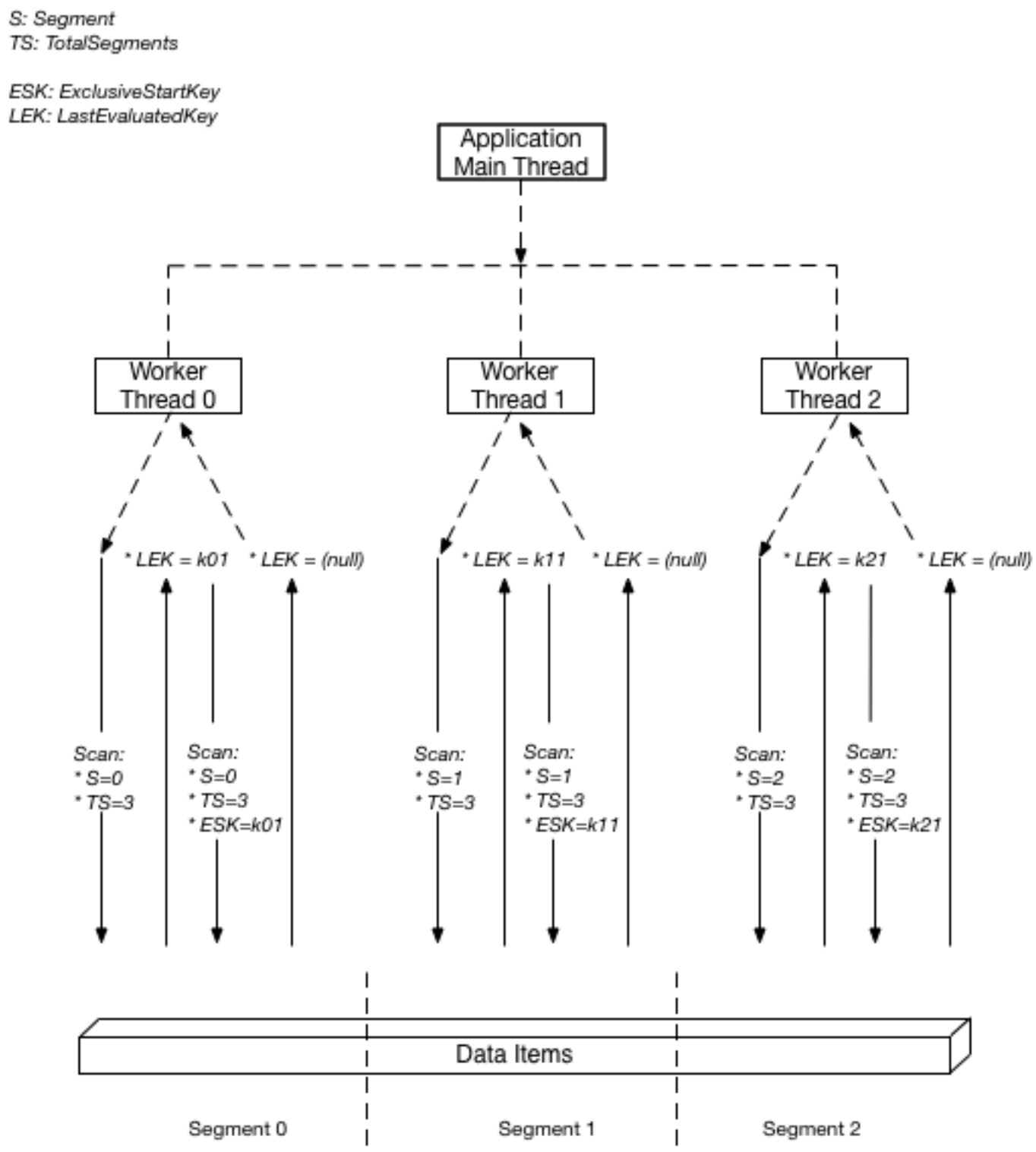
기본적으로 Scan 작업은 데이터를 순차적으로 처리합니다. Amazon DynamoDB는 애플리케이션에 1MB 단위로 데이터를 반환하고 애플리케이션은 추가 Scan 작업을 수행하여 다음 1MB의 데이터를 가져옵니다.

스캔할 테이블 또는 인덱스가 클수록 Scan을 완료하는 데 걸리는 시간이 늘어납니다. 또한 순차적 Scan은 프로비저닝된 읽기 처리 용량을 항상 최대한 사용할 수 있는 것은 아닙니다. DynamoDB가 여러 물리적 파티션 간에 큰 테이블 데이터를 분산해도 Scan 작업은 한 번에 한 파티션만 읽을 수 있습니다. 이러한 이유로 Scan의 처리량은 단일 파티션의 최대 처리량에 따라 제약을 받습니다.

이 문제를 해결하기 위해 Scan 작업에서는 테이블 또는 보조 인덱스를 여러 세그먼트로 논리적으로 나눠 여러 애플리케이션 작업자가 세그먼트를 병렬로 처리하도록 할 수 있습니다. 각 작업자는 스레드(멀티스레딩을 지원하는 프로그래밍 언어에서) 또는 운영 체제 프로세스일 수 있습니다. 병렬 스캔을 수행하려면 각 작업자가 다음 파라미터를 사용하여 Scan 요청을 발행합니다.

- Segment - 특정 작업자가 스캔할 세그먼트입니다. 각 작업자는 Segment에 서로 다른 값을 사용해야 합니다.
- TotalSegments - 병렬 스캔에 사용되는 총 세그먼트 수입니다. 이 값은 애플리케이션에서 사용할 작업자 수와 같아야 합니다.

다음 다이어그램은 멀티스레드 애플리케이션이 3 병렬 처리 수준을 사용하여 병렬 Scan을 수행하는 방법에 대해 설명합니다.




이 다이어그램에서 애플리케이션은 세 개의 스레드를 생성하여 각 스레드에 번호를 지정합니다. (세그먼트는 0부터 시작하므로 첫 번째 번호는 항상 0입니다.) 각 스레드는 `Scan` 요청을 실행하며, `Segment`를 해당 지정 번호로 설정하고 `TotalSegments`를 3으로 설정합니다. 각 스레드는 해당 지정

세그먼트를 스캔하며 한 번에 1MB의 데이터를 가져와서 애플리케이션의 기본 스레드에 데이터를 반환합니다.

`Segment` 및 `TotalSegments` 값을 개별 `Scan` 요청에 적용하여 언제든지 다른 값을 사용할 수 있습니다. 애플리케이션이 최적의 성능을 발휘하려면 이러한 값과 사용하는 작업자 수에 대한 여러 번의 시도가 필요할 수 있습니다.

> **Note**
>
> 작업자 수가 많은 병렬 스캔 작업은 스캔할 테이블 또는 인덱스에 대한 모든 할당된 처리량을 족히 소비할 수 있습니다. 테이블 또는 인덱스가 다른 애플리케이션에서 과도한 읽기 또는 쓰기 작업을 발생시키는 경우 이러한 스캔을 수행하지 않아야 합니다.
> 요청당 반환되는 데이터 양을 제어하려면 `Limit` 파라미터를 사용하세요. 이를 사용하면 다른 모든 작업자가 이용할 비용으로 한 작업자가 할당된 모든 처리량을 소비하게 되는 상황을 방지할 수 있습니다.

## PartiQL - Amazon DynamoDB용 SQL 호환 쿼리 언어

Amazon DynamoDB에서는 SQL 호환 쿼리 언어인 PartiQL을 사용하여 Amazon DynamoDB에서 데이터를 선택, 삽입, 업데이트, 삭제할 수 있습니다. PartiQL을 사용하면 DynamoDB 테이블과 쉽게 상호 작용하고 AWS Management Console, NoSQL Workbench, AWS Command Line Interface 및 PartiQL용 DynamoDB API를 사용하여 임시 쿼리를 실행할 수 있습니다.

PartiQL 작업은 다른 DynamoDB 데이터 영역 작업과 동일한 가용성, 대기 시간 및 성능을 제공합니다.

다음 단원에서는 PartiQL의 DynamoDB 구현을 설명합니다.

주제

- PartiQL이란?
- Amazon DynamoDB의 PartiQL
- DynamoDB용 PartiQL 시작하기
- DynamoDB에 대한 PartiQL 데이터 형식
- DynamoDB의 PartiQL 문
- DynamoDB에서 PartiQL 함수 사용
- DynamoDB용 PartiQL 산술, 비교 및 논리 연산자

- DynamoDB용 PartiQL에서 트랜잭션 수행
- DynamoDB용 PartiQL에서 일괄 작업 실행
- DynamoDB용 PartiQL을 사용하는 IAM 보안 정책

## PartiQL이란?

PartiQL은 구조화 데이터, 반구조화 데이터 및 중첩 데이터를 포함하는 여러 데이터 스토어에서 SQL 호환 쿼리 액세스를 제공합니다. Amazon 내에서 널리 사용되며 현재 DynamoDB를 비롯한 여러 AWS 서비스의 일부로 제공됩니다.

PartiQL 사양과 핵심 쿼리 언어에 대한 자습서는 PartiQL 설명서를 참조하세요.

> **Note**
>
> - Amazon DynamoDB는 PartiQL 쿼리 언어의 하위 집합을 지원합니다.
> - Amazon DynamoDB에서는 Amazon ion 데이터 형식 또는 Amazon Ion 리터럴을 지원하지 않습니다.

## Amazon DynamoDB의 PartiQL

DynamoDB에서 PartiQL 쿼리를 실행하려면 다음을 사용할 수 있습니다.

- DynamoDB 콘솔
- NoSQL Workbench
- AWS Command Line Interface(AWS CLI)
- DynamoDB API

DynamoDB에 액세스하는 이러한 방법에 대한 자세한 내용은 DynamoDB 액세스를 참조하세요.

## DynamoDB용 PartiQL 시작하기

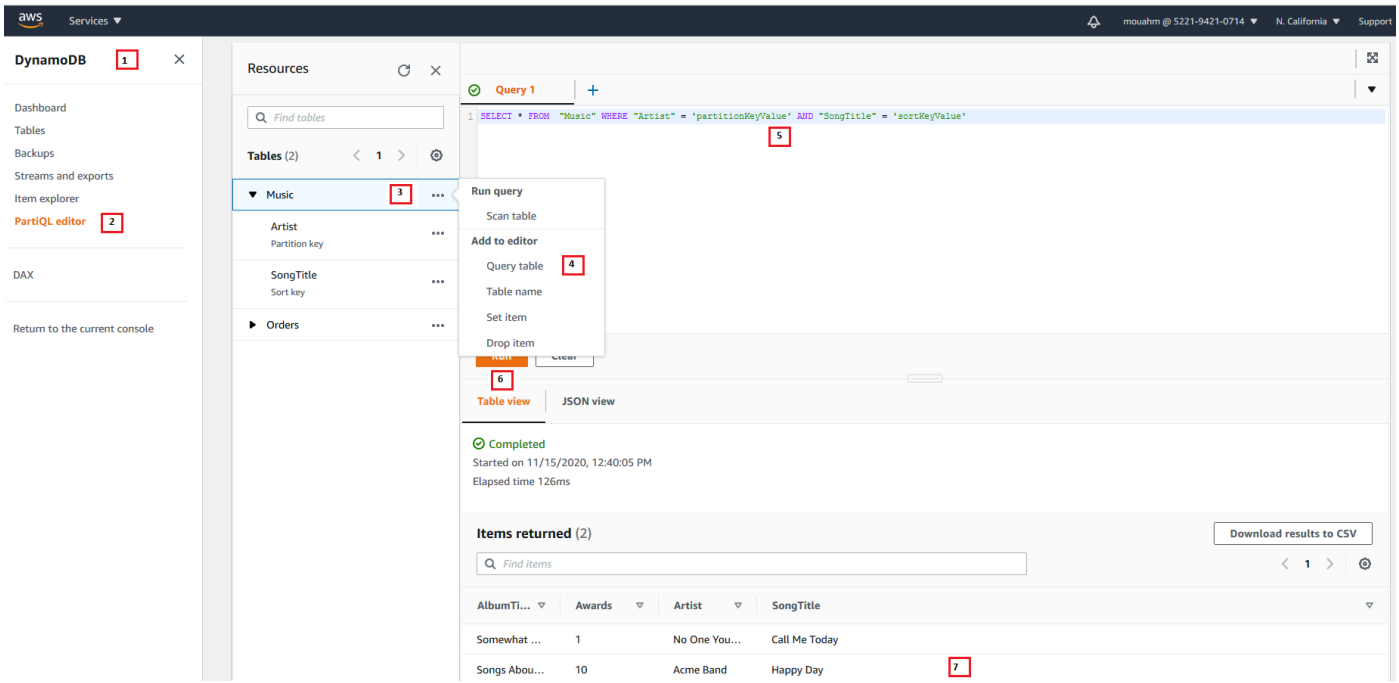
이 단원에서는 Amazon DynamoDB 콘솔, AWS Command Line Interface(AWS CLI) 및 DynamoDB API에서 DynamoDB용 PartiQL을 사용하는 방법을 설명합니다.

다음 예에서 DynamoDB 시작하기 자습서에 정의된 DynamoDB 테이블은 사전 조건입니다.

DynamoDB 콘솔, AWS Command Line Interface 또는 DynamoDB API를 사용하여 DynamoDB에 액세스하는 방법에 대한 자세한 내용은 DynamoDB DB 액세스를 참조하세요.

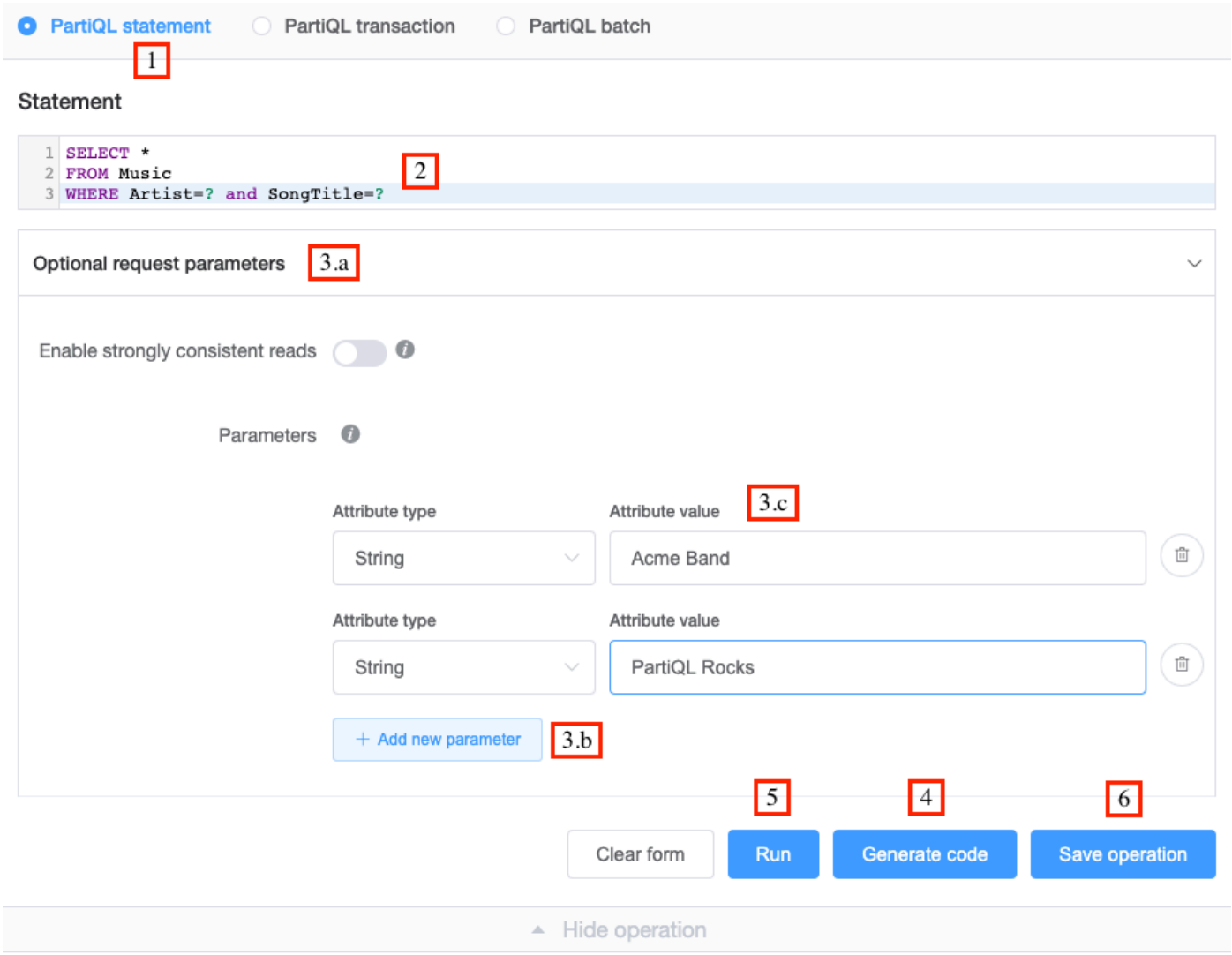
NoSQL 워크벤치를 다운로드하고 사용하여 DynamoDB용 PartiQL 문을 빌드하려면 DynamoDB용 NoSQL Workbench 작업 빌더(Operation Builder) 오른쪽 맨 위에서 PartiQL 작업(PartiQL operations)를 선택합니다.

Console




1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 PartiQL editor(PartiQL 편집기)를 선택합니다.
3. Music 테이블을 선택합니다.
4. Query table(테이블 쿼리)을 선택합니다. 이 작업을 수행하면 전체 테이블이 스캔되지 않는 쿼리가 생성됩니다.
5. `partitionKeyValue`를 문자열 값 `Acme Band`로 바꿉니다. `sortKeyValue`를 문자열 값 `Happy Day`로 바꿉니다.
6. 실행 버튼을 선택합니다.
7. Table view(테이블 보기) 또는 JSON view(JSON 보기) 버튼을 선택하여 쿼리의 결과를 볼 수 있습니다.

## NoSQL workbench




1. PartiQL statement(PartiQL 문)를 선택합니다.
2. 다음 PartiQL SELECT 문을 입력합니다.

```
SELECT *
FROM Music
WHERE Artist=? and SongTitle=?
```

3. `Artist` 및 `SongTitle` 파라미터의 값을 지정하려면 다음을 수행합니다.

   a. Optional request parameters(선택적 요청 파라미터)를 선택합니다.

   b. Add new parameter(새 파라미터 추가)를 선택합니다.

   c. 속성 형식 문자열과 값 `Acme Band`를 선택합니다.

d. b단계와 c단계를 반복하고 형식 문자열과 값 `PartiQL Rocks`를 선택합니다.

4. 코드를 생성하려면 Generate code(코드 생성)를 선택합니다.

   표시된 탭에서 원하는 언어를 선택합니다. 이제 이 코드를 복사하여 애플리케이션에서 사용할 수 있습니다.

5. 작업을 즉시 실행하려면 Run(실행)을 선택합니다.

AWS CLI

1. INSERT PartiQL 문을 사용하여 `Music` 테이블에 항목을 생성합니다.

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
         VALUE  \
         {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

2. SELECT PartiQL 문을 사용하여 Music 테이블에서 항목을 검색합니다.

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

3. UPDATE PartiQL 문을 사용하여 `Music` 테이블의 항목을 업데이트합니다.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardsWon=1  \
                                            SET AwardDetail={'Grammys':[2020,
 2018]}  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

   `Music` 테이블에서 항목의 목록 값을 추가합니다.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.Grammys
 =list_append(AwardDetail.Grammys,[2016])  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

   `Music` 테이블에서 항목의 목록 값을 제거합니다.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            REMOVE AwardDetail.Grammys[2]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Music 테이블에서 항목의 새 맵 멤버를 추가합니다.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.BillBoard=[2020]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Music 테이블에서 항목의 새 문자열 집합 속성을 추가합니다.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers =<<'member1',
 'member2'>>  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Music 테이블에서 항목의 문자열 집합 속성을 업데이트합니다.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers
 =set_add(BandMembers, <<'newmember'>>)  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

4. DELETE PartiQL 문을 사용하여 Music 테이블에서 항목을 삭제합니다.

```
aws dynamodb execute-statement --statement "DELETE  FROM Music  \
    WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
```

Java

```
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.AmazonServiceException;
import software.amazon.dynamodb.AmazonDynamoDB;
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.model.AttributeValue;
import software.amazon.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.dynamodb.model.ExecuteStatementRequest;
import software.amazon.dynamodb.model.ExecuteStatementResult;
import software.amazon.dynamodb.model.InternalServerErrorException;
import software.amazon.dynamodb.model.ItemCollectionSizeLimitExceededException;
import software.amazon.dynamodb.model.ProvisionedThroughputExceededException;
import software.amazon.dynamodb.model.RequestLimitExceededException;
import software.amazon.dynamodb.model.ResourceNotFoundException;
import software.amazon.dynamodb.model.TransactionConflictException;

public class DynamoDBPartiQGettingStarted {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-1");

        try {
            // Create ExecuteStatementRequest
            ExecuteStatementRequest executeStatementRequest = new
 ExecuteStatementRequest();
            List<AttributeValue> parameters= getPartiQLParameters();

            //Create an item in the Music table using the INSERT PartiQL statement
            processResults(executeStatementRequest(dynamoDB, "INSERT INTO Music
 value {'Artist':?,'SongTitle':?}", parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
 statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
 where Artist=? and SongTitle=?", parameters));

            //Update an item in the Music table using the UPDATE PartiQL statement.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
 SET AwardsWon=1 SET AwardDetail={'Grammys':[2020, 2018]}  where Artist=? and
 SongTitle=?", parameters));

            //Add a list value for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
 AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])  where Artist=? and
 SongTitle=?", parameters));
```

```
            //Remove a list value for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music REMOVE
AwardDetail.Grammys[2]   where Artist=? and SongTitle=?", parameters));

            //Add a new map member for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist=? and SongTitle=?", parameters));

            //Add a new string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
SET BandMembers =<<'member1', 'member2'>> where Artist=? and SongTitle=?",
parameters));

            //update a string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
BandMembers =set_add(BandMembers, <<'newmember'>>) where Artist=? and SongTitle=?",
parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
where Artist=? and SongTitle=?", parameters));

            //delete an item from the Music Table
            processResults(executeStatementRequest(dynamoDB, "DELETE  FROM Music
where Artist=? and SongTitle=?", parameters));
        } catch (Exception e) {
            handleExecuteStatementErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {
        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static List<AttributeValue> getPartiQLParameters() {
        List<AttributeValue> parameters = new ArrayList<AttributeValue>();
        parameters.add(new AttributeValue("Acme Band"));
        parameters.add(new AttributeValue("PartiQL Rocks"));
        return parameters;
    }
```

```
   private static ExecuteStatementResult executeStatementRequest(AmazonDynamoDB
client, String statement, List<AttributeValue> parameters ) {
       ExecuteStatementRequest request = new ExecuteStatementRequest();
       request.setStatement(statement);
       request.setParameters(parameters);
       return client.executeStatement(request);
   }

   private static void processResults(ExecuteStatementResult
executeStatementResult) {
       System.out.println("ExecuteStatement successful: "+
executeStatementResult.toString());

   }

   // Handles errors during ExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
   // your application use-case.
   private static void handleExecuteStatementErrors(Exception exception) {
       try {
           throw exception;
       } catch (ConditionalCheckFailedException ccfe) {
           System.out.println("Condition check specified in the operation failed,
review and update the condition " +
                                       "check before retrying. Error: " +
ccfe.getErrorMessage());
       } catch (TransactionConflictException tce) {
           System.out.println("Operation was rejected because there is an ongoing
transaction for the item, generally " +
                                       "safe to retry with exponential back-off.
Error: " + tce.getErrorMessage());
       } catch (ItemCollectionSizeLimitExceededException icslee) {
           System.out.println("An item collection is too large, you\'re using Local
Secondary Index and exceeded " +
                                       "size limit of items per
partition key. Consider using Global Secondary Index instead. Error: " +
icslee.getErrorMessage());
       } catch (Exception e) {
           handleCommonErrors(e);
       }
   }

   private static void handleCommonErrors(Exception exception) {
       try {
```

```
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
 exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
 your account, increase account level throughput before " +
                                       "retrying. Error: " +
 rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
 retry strategy make sure to retry with exponential back-off. " +
                                       "Otherwise consider reducing frequency of
 requests or increasing provisioned capacity for your table or secondary index.
 Error: " +
                                       ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
 before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                                       "service, but for some reason, the service
 was not able to process it, and returned an error response instead. Investigate and
 " +
                                       "configure retry strategy. Error type: " +
 ase.getErrorType() + ". Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                                       "service, or the client was unable to parse
 the response from the service. Investigate and configure retry strategy. "+
                                       "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

# DynamoDB에 대한 PartiQL 데이터 형식

다음 표에는 DynamoDB용 PartiQL에서 사용할 수 있는 데이터 형식이 나와 있습니다.

| DynamoDB 데이터 형식 | PartiQL 표현 | Notes |
|---|---|---|
| `Boolean` | TRUE \| FALSE | 대/소문자를 구분하지 않습니다. |
| `Binary` | N/A | 코드를 통해서만 지원됩니다. |
| `List` | [ value1, value2,...] | 목록 형식에 저장할 수 있는 데이터 형식에는 제한이 없으며, 목록에 있는 요소의 형식이 달라도 상관없습니다. |
| `Map` | { 'name' : value } | 맵 형식에 저장할 수 있는 데이터 형식에는 제한이 없으며, 맵에 있는 요소의 형식이 달라도 상관없습니다. |
| `Null` | NULL | 대/소문자를 구분하지 않습니다. |
| `Number` | 1, 1.0, 1e0 | 번호는 양수, 음수 또는 0일 수 있습니다. 숫자는 최대 38자릿수까지 지원됩니다. |
| `Number Set` | <<number1, number2>> | 숫자 집합의 요소는 숫자 형식이어야 합니다. |
| `String Set` | <<'string1', 'string2'>> | 문자열 집합의 요소는 문자열 형식이어야 합니다. |
| `String` | 'string value' | 작은 따옴표를 사용하여 문자열 값을 지정해야 합니다. |

예시

다음 문은 String, Number, Map, List, Number Set 및 String Set 데이터 형식을 삽입하는 방법을 보여 줍니다.

```
INSERT INTO TypesTable value {'primarykey':'1',
'NumberType':1,
'MapType' : {'entryname1': 'value', 'entryname2': 4},
'ListType': [1,'stringval'],
'NumberSetType':<<1,34,32,4.5>>,
'StringSetType':<<'stringval','stringval2'>>
}
```

다음 문은 Map, List, Number Set 및 String Set 형식에 새 요소를 삽입하고 Number 형식의 값을 변경하는 방법을 보여 줍니다.

```
UPDATE TypesTable
SET NumberType=NumberType + 100
SET MapType.NewMapEntry=[2020, 'stringvalue', 2.4]
SET ListType = LIST_APPEND(ListType, [4, <<'string1', 'string2'>>])
SET NumberSetType= SET_ADD(NumberSetType, <<345, 48.4>>)
SET StringSetType = SET_ADD(StringSetType, <<'stringsetvalue1', 'stringsetvalue2'>>)
WHERE primarykey='1'
```

다음 문은 Map, List, Number Set 및 String Set 형식에서 요소를 제거하고 Number 형식의 값을 변경하는 방법을 보여 줍니다.

```
UPDATE TypesTable
SET NumberType=NumberType - 1
REMOVE ListType[1]
REMOVE MapType.NewMapEntry
SET NumberSetType = SET_DELETE( NumberSetType, <<345>>)
SET StringSetType = SET_DELETE( StringSetType, <<'stringsetvalue1'>>)
WHERE primarykey='1'
```

자세한 내용은 DynamoDB 데이터 형식 단원을 참조하세요.

## DynamoDB의 PartiQL 문

Amazon DynamoDB에서는 다음 PartiQL 문을 지원합니다.

> **Note**
>
> DynamoDB는 일부 PartiQL 문을 지원하지 않습니다.
> 이 참조에서는 AWS CLI 또는 API를 사용하여 수동으로 실행하는 PartiQL 문의 기본 구문 및 사용 예제를 제공합니다.

데이터 조작 언어(DML)는 DynamoDB 테이블의 데이터를 관리하는 데 사용하는 PartiQL 문 집합입니다. DML 문을 사용하여 테이블의 데이터를 추가, 수정 또는 삭제할 수 있습니다.

지원되는 DML 및 쿼리 언어 문은 다음과 같습니다.

- DynamoDB의 PartiQL select 문
- DynamoDB의 PartiQL update 문
- DynamoDB의 PartiQL insert 문
- DynamoDB의 PartiQL delete 문

DynamoDB용 PartiQL에서 트랜잭션 수행 및 DynamoDB용 PartiQL에서 일괄 작업 실행도 DynamoDB용 PartiQL에서 지원됩니다.

DynamoDB의 PartiQL select 문

SELECT 문을 사용하면 Amazon DynamoDB의 테이블에서 데이터를 검색할 수 있습니다.

SELECT 문을 사용하면 WHERE 절에 파티션 키를 사용한 등식 또는 IN 조건을 지정하지 않은 경우 전체 테이블이 스캔될 수 있습니다. 스캔 작업은 요청한 값을 찾기 위해 전체 항목을 검사하기 때문에 대용량 테이블이나 인덱스일 경우에는 단 한 번의 작업으로 프로비저닝된 처리량을 모두 사용할 수 있습니다.

PartiQL에서 전체 테이블 스캔을 방지하려면 다음과 같이 할 수 있습니다.

- WHERE 절 조건이 적절히 구성되도록 하여 전체 테이블이 스캔되지 않도록 SELECT 문을 작성합니다.
- DynamoDB 개발자 안내서의 예: DynamoDB용 PartiQL에서 select 문은 허용하고 전체 테이블 스캔 문은 거부 단원에 지정된 IAM 정책을 사용하여 전체 테이블 스캔을 사용 중지합니다.

자세한 내용은 DynamoDB 개발자 안내서에서 데이터 쿼리 및 검색 모범 사례를 참조하세요.

주제


- 구문
- 파라미터
- 예시


구문

```
SELECT expression  [, ...]
FROM table[.index]
[ WHERE condition ] [ [ORDER BY key [DESC|ASC] , ...]
```

파라미터

***expression***

(필수) * 와일드카드에서 형성된 프로젝션 또는 결과 집합에 있는 하나 이상의 속성 이름 또는 문서 경로로 구성된 프로젝션 목록입니다. expression은 DynamoDB에서 PartiQL 함수 사용에 대한 호출 또는 DynamoDB용 PartiQL 산술, 비교 및 논리 연산자 에서 수정되는 필드로 구성될 수 있습니다.

***table***

(필수) 쿼리할 테이블 이름입니다.

***#### #### ### ####***

(선택 사항) 쿼리할 인덱스의 이름입니다.

> **Note**
>
> 인덱스를 쿼리할 때 테이블 이름과 인덱스 이름에 큰따옴표를 추가해야 합니다.
>
> ```
> SELECT *
> FROM "TableName"."IndexName"
> ```

***condition***

(선택 사항) 쿼리의 선택 기준입니다.

**Important**

SELECT 문이 전체 테이블 스캔이 되지 않게 하려면 WHERE 절 조건에서 파티션 키를 지정해야 합니다. 등식 또는 IN 연산자를 사용합니다.
예를 들어, Orders 테이블에 OrderID 파티션 키 및 키가 아닌 기타 속성(Address 등)이 있는 경우 다음 문을 사용하면 전체 테이블 스캔이 되지 않습니다.

```
SELECT *
FROM "Orders"
WHERE OrderID = 100

SELECT *
FROM "Orders"
WHERE OrderID = 100 and Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 or OrderID = 200

SELECT *
FROM "Orders"
WHERE OrderID IN [100, 300, 234]
```

그러나 다음 SELECT 문을 사용하면 전체 테이블 스캔이 됩니다.

```
SELECT *
FROM "Orders"
WHERE OrderID > 1

SELECT *
FROM "Orders"
WHERE Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 OR Address='some address'
```

#

(선택 사항) 반환된 결과를 정렬하는 데 사용할 해시 키 또는 정렬 키입니다. 기본 순서는 오름차순(ASC)이며, 결과가 내림차순으로 반환되게 하려면 DESC를 지정합니다.

> **Note**
>
> WHERE 절을 생략하면 테이블의 모든 항목이 검색됩니다.

예시

다음 쿼리는 파티션 키 OrderID를 지정하고 등식 연산자를 사용하여 Orders 테이블에서 항목 하나(있는 경우)를 반환합니다.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1
```

다음 쿼리는 특정 파티션 키 OrderID 값을 OR 연산자로 지정하여 Orders 테이블에서 모든 항목을 반환합니다.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1 OR OrderID = 2
```

다음 쿼리는 특정 파티션 키 OrderID 값을 IN 연산자로 지정하여 Orders 테이블에서 모든 항목을 반환합니다. 반환되는 결과는 OrderID 키 속성 값을 기준으로 내림차순으로 표시됩니다.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID IN [1, 2, 3] ORDER BY OrderID DESC
```

다음 쿼리는 Orders 테이블에서 Total이 500보다 큰 모든 항목을 반환하는 전체 테이블 스캔을 표시합니다. 여기서 Total은 키가 아닌 속성입니다.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total > 500
```

다음 쿼리는 IN 연산자와 키가 아닌 속성 Total을 사용하여 Orders 테이블에서 특정 Total 주문 범위 내의 모든 항목을 반환하는 전체 테이블 스캔을 표시합니다.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total IN [500, 600]
```

다음 쿼리는 BETWEEN 연산자가 키가 아닌 속성 Total을 사용하여 Orders 테이블에서 특정 Total 주문 범위 내의 모든 항목을 반환하는 전체 테이블 스캔을 표시합니다.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total BETWEEN 500 AND 600
```

다음 쿼리는 WHERE 절 조건에 파키션 키 CustomerID 및 정렬 키 MovieID를 지정하고 SELECT 절에 문서 경로를 사용하여 파이어스틱 디바이스를 사용해 시청한 첫 번째 날짜를 반환합니다.

```
SELECT Devices.FireStick.DateWatched[0]
FROM WatchList
WHERE CustomerID= 'C1' AND MovieID= 'M1'
```

다음 쿼리는 WHERE 절 조건에 문서 경로를 사용하여 2019년 12월 24일 이후 파이어스틱 디바이스가 처음 사용된 항목 목록을 반환하는 전체 테이블 스캔을 표시합니다.

```
SELECT Devices
FROM WatchList
WHERE Devices.FireStick.DateWatched[0] >= '12/24/19'
```

DynamoDB의 PartiQL update 문

UPDATE 문을 사용하면 Amazon DynamoDB 테이블에서 항목 내의 특성 하나 이상의 값을 수정할 수 있습니다.

> **Note**
>
> 한 번에 하나의 항목만 업데이트할 수 있어서, 여러 항목을 업데이트하는 단일 DynamoDB PartiQL 문을 실행할 수는 없습니다. 여러 항목을 업데이트하는 방법은 DynamoDB용 PartiQL에서 트랜잭션 수행 또는 DynamoDB용 PartiQL에서 일괄 작업 실행 단원을 참조하세요.

주제


- 구문
- 파라미터
- 반환 값
- 예시


구문

```
UPDATE  table
[SET | REMOVE]  path  [=  data] […]
WHERE condition [RETURNING returnvalues]
<returnvalues>  ::= [ALL OLD | MODIFIED OLD | ALL NEW | MODIFIED NEW] *
```

파라미터

***table***

(필수) 수정할 데이터가 포함된 테이블입니다.

***##***

(필수) 생성하거나 수정할 속성 이름 또는 문서 경로입니다.

***data***

(필수) 속성 값 또는 작업의 결과입니다.

SET과 함께 사용할 수 있는 작업은 다음과 같습니다.

- LIST_APPEND: 목록 형식에 값을 추가합니다.
- SET_ADD: 숫자 또는 문자열 집합에 값을 추가합니다.
- SET_DELETE: 숫자 또는 문자열 집합에서 값을 제거합니다.

***condition***

(필수) 수정할 항목의 선택 기준입니다. 이 조건은 단일 기본 키 값으로 확인되어야 합니다.

***returnvalues***

(선택 사항) 항목 속성이 업데이트되기 전이나 후에 표시되는 해당 속성을 가져오려면 `returnvalues`를 사용합니다. 유효한 값은 다음과 같습니다.

- `ALL OLD *` - 업데이트 작업 전에 표시된 항목 속성을 모두 반환합니다.
- `MODIFIED OLD *` - 업데이트 작업 전에 표시된 업데이트된 속성만 반환합니다.
- `ALL NEW *` - 업데이트 작업 후에 표시되는 항목 속성을 모두 반환합니다.
- `MODIFIED NEW *` - `UpdateItem` 작업 후에 표시되는 업데이트된 속성만 반환합니다.

반환 값

이 문은 `returnvalues` 파라미터를 지정하지 않는 경우 값을 반환하지 않습니다.

> **Note**
>
> DynamoDB 테이블의 모든 항목에 대해 UPDATE 문의 WHERE 절이 true로 평가되지 않으면 `ConditionalCheckFailedException`이 반환됩니다.

예시

기존 항목의 속성 값을 업데이트합니다. 속성이 없으면 새로 생성됩니다.

다음 쿼리는 숫자 형식의 속성(`AwardsWon`)과 맵 형식의 속성(`AwardDetail`)을 추가하여 `"Music"` 테이블에서 항목을 업데이트합니다.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

`RETURNING ALL OLD *`를 추가하여 `Update` 작업 전에 있던 속성을 반환할 수 있습니다.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL OLD *
```

그러면 다음이 반환됩니다.

```
{
    "Items": [
```

```
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

RETURNING ALL NEW *를 추가하여 Update 작업 후에 있던 속성을 반환할 수 있습니다.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL NEW *
```

그러면 다음이 반환됩니다.

```
{
    "Items": [
        {
            "AwardDetail": {
                "M": {
                    "Grammys": {
                        "L": [
                            {
                                "N": "2020"
                            },
                            {
                                "N": "2018"
                            }
                        ]
                    }
                }
            },
            "AwardsWon": {
                "N": "1"
            }
        }
    ]
```

```
}
```

다음 쿼리는 AwardDetail.Grammys 목록에 추가하여 "Music" 테이블에서 항목을 업데이트합니다.

```
UPDATE "Music"
SET AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

다음 쿼리는 AwardDetail.Grammys 목록에서 제거하여 "Music" 테이블에서 항목을 업데이트합니다.

```
UPDATE "Music"
REMOVE AwardDetail.Grammys[2]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

다음 쿼리는 AwardDetail 맵에 "Music"을 추가하여 BillBoard 테이블에서 항목을 업데이트합니다.

```
UPDATE "Music"
SET AwardDetail.BillBoard=[2020]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

다음 쿼리는 문자열 집합 속성 BandMembers를 추가하여 "Music" 테이블에서 항목을 업데이트합니다.

```
UPDATE "Music"
SET BandMembers =<<'member1', 'member2'>>
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

다음 쿼리는 문자열 집합 BandMembers에 newbandmember를 추가하여 "Music" 테이블에서 항목을 업데이트합니다.

```
UPDATE "Music"
SET BandMembers =set_add(BandMembers, <<'newbandmember'>>)
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

### DynamoDB의 PartiQL delete 문

DELETE 문을 사용하면 Amazon DynamoDB 테이블에서 기존 항목을 삭제할 수 있습니다.

> **Note**
>
> 항목은 한 번에 하나만 삭제할 수 있습니다. 여러 항목을 삭제하는 단일 DynamoDB PartiQL 문을 실행할 수는 없습니다. 여러 항목을 삭제하는 방법에 대한 자세한 내용은 DynamoDB용 PartiQL에서 트랜잭션 수행 또는 DynamoDB용 PartiQL에서 일괄 작업 실행 단원을 참조하세요.

주제


- 구문
- 파라미터
- 반환 값
- 예시


구문

```
DELETE FROM table
 WHERE condition [RETURNING returnvalues]
 <returnvalues>  ::= ALL OLD *
```

파라미터

***table***

(필수) 삭제할 항목이 포함된 DynamoDB 테이블입니다.

***condition***

(필수) 삭제할 항목의 선택 기준입니다. 이 조건은 단일 기본 키 값으로 확인되어야 합니다.

***returnvalues***

(선택 사항) 항목 속성이 삭제되기 전에 표시된 해당 속성을 가져오려면 `returnvalues`를 사용합니다. 유효한 값은 다음과 같습니다.

- `ALL OLD *` - 이전 항목의 내용이 반환됩니다.

반환 값

이 문은 `returnvalues` 파라미터를 지정하지 않는 경우 값을 반환하지 않습니다.

> **Note**
>
> DELETE가 실행된 대상 항목과 기본 키가 같은 항목이 DynamoDB 테이블에 없는 경우 삭제된 항목이 0개인 SUCCESS가 반환됩니다. 테이블에 기본 키가 같은 항목이 있지만 DELETE 문의 WHERE 절 조건이 false로 평가되는 경우에는 `ConditionalCheckFailedException`이 반환됩니다.

예시

다음 쿼리는 `"Music"` 테이블의 항목을 삭제합니다.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
```

`RETURNING ALL OLD *` 파라미터를 추가하여 삭제된 데이터를 반환할 수 있습니다.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
 RETURNING ALL OLD *
```

이제 `Delete` 문은 다음을 반환합니다.

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

DynamoDB의 PartiQL insert 문

`INSERT` 문을 사용하면 Amazon DynamoDB의 테이블에 항목을 추가할 수 있습니다.

> **Note**
>
> 한 번에 하나의 항목만 업데이트할 수 있어서, 여러 항목을 삽입하는 단일 DynamoDB PartiQL 문을 실행할 수는 없습니다. 여러 항목을 삽입하는 방법에 대한 자세한 내용은 DynamoDB용 PartiQL에서 트랜잭션 수행 또는 DynamoDB용 PartiQL에서 일괄 작업 실행 단원을 참조하세요.

주제


- 구문
- 파라미터
- 반환 값
- 예시


구문

단일 항목을 삽입합니다.

```
INSERT INTO table VALUE item
```

파라미터

***table***

(필수) 데이터를 삽입할 테이블입니다. 이미 있는 테이블이어야 합니다.

***item***

(필수) PartiQL 튜플로 표시된 유효한 DynamoDB 항목입니다. 항목을 하나만 지정해야 하며 항목의 각 속성 이름은 대/소문자를 구분하고 PartiQL에서 작은 따옴표(`'...'`)로 표시될 수 있습니다.

문자열 값도 PartiQL에서 작은 따옴표(`'...'`)로 표시될 수 있습니다.

반환 값

이 문은 값을 반환하지 않습니다.

> **Note**
>
> DynamoDB 테이블에 삽입되는 항목과 기본 키가 같은 항목이 이미 있는 경우 `DuplicateItemException`이 반환됩니다.

예시

```
INSERT INTO "Music" value {'Artist' : 'Acme Band','SongTitle' : 'PartiQL Rocks'}
```

## DynamoDB에서 PartiQL 함수 사용

Amazon DynamoDB의 PartiQL은 SQL 표준 함수의 다음과 같은 기본 제공 변형을 지원합니다.

> **Note**
>
> 이 목록에 없는 모든 SQL 함수는 현재 DynamoDB에서 지원되지 않습니다.

집계 함수

- Amazon DynamoDB용 PartiQL에서 SIZE 함수 사용

조건 함수

- DynamoDB용 PartiQL에서 EXISTS 함수 사용
- DynamoDB용 PartiQL에서 ATTRIBUTE_TYPE 함수 사용
- DynamoDB용 PartiQL에서 BEGINS_WITH 함수 사용
- DynamoDB용 PartiQL에서 CONTAINS 함수 사용
- DynamoDB용 PartiQL에서 MISSING 함수 사용

### DynamoDB용 PartiQL에서 EXISTS 함수 사용

EXISTS를 사용하면 TransactWriteItems API에서 `ConditionCheck`가 하는 기능을 똑같이 수행할 수 있습니다. EXISTS 함수는 트랜잭션에서만 사용할 수 있습니다.

값이 지정된 경우 값이 비어 있지 않은 컬렉션이면 `TRUE`를 반환합니다. 그렇지 않은 경우 `FALSE`를 반환합니다.

> **Note**
>
> 이 함수는 트랜잭션 작업에서만 사용할 수 있습니다.

구문

```
EXISTS ( statement )
```

인수

*##*

(필수) 함수가 평가하는 SELECT 문입니다.

> **Note**
>
> SELECT 문은 전체 기본 키와 하나의 다른 조건을 지정해야 합니다.

반환 타입

bool

예시

```
EXISTS(
    SELECT * FROM "Music"
    WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks')
```

DynamoDB용 PartiQL에서 BEGINS_WITH 함수 사용

지정된 속성이 특정 하위 문자열로 시작하는 경우 TRUE를 반환합니다.

구문

```
begins_with(path, value )
```

인수

*##*

(필수) 사용할 속성 이름 또는 문서 경로입니다.

*USD ##*

(필수) 검색할 문자열입니다.

반환 타입

bool

예시

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND begins_with("Address", '7834 24th')
```

DynamoDB용 PartiQL에서 MISSING 함수 사용

항목에 지정된 속성이 포함되어 있지 않은 경우 TRUE를 반환합니다. 이 함수에는 등식 및 부등식 연산자만 사용할 수 있습니다.

구문

```
 attributename IS | IS NOT  MISSING
```

인수

*attributename*

(필수) 찾을 속성 이름입니다.

반환 타입

bool

예시

```
SELECT * FROM Music WHERE "Awards" is MISSING
```

### DynamoDB용 PartiQL에서 ATTRIBUTE_TYPE 함수 사용

지정된 경로의 속성이 특정 데이터 형식인 경우 TRUE를 반환합니다.

#### 구문

```
attribute_type( attributename, type )
```

#### 인수

*attributename*

(필수) 사용할 속성 이름입니다.

type

(필수) 확인할 속성 형식입니다. 유효한 값 목록은 DynamoDB attribute_type을 참조하세요.

#### 반환 타입

bool

#### 예시

```
SELECT * FROM "Music" WHERE attribute_type("Artist", 'S')
```

### DynamoDB용 PartiQL에서 CONTAINS 함수 사용

경로에서 지정된 속성이 다음 중 하나인 경우 TRUE를 반환합니다.

- 특정 하위 문자열을 포함하는 문자열
- 특정 요소를 포함하는 집합

자세한 내용은 DynamoDB contains 함수를 참조하세요.

#### 구문

```
contains( path, substring )
```

인수

*##*

(필수) 사용할 속성 이름 또는 문서 경로입니다.

*substring*

(필수) 확인할 속성 하위 문자열 또는 집합 멤버입니다. 자세한 내용은 DynamoDB contains 함수를 참조하세요.

반환 타입

bool

예시

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND contains("Address", 'Kirkland')
```

Amazon DynamoDB용 PartiQL에서 SIZE 함수 사용

속성의 크기(바이트)를 나타내는 숫자가 반환됩니다. Size를 사용하는 데 유효한 데이터 형식은 다음과 같습니다. 자세한 내용은 DynamoDB size 함수를 참조하세요.

구문

```
size( path)
```

인수

*##*

(필수) 속성 이름 또는 문서 경로입니다.

지원되는 형식은 DynamoDB size 함수를 참조하세요.

반환 타입

int

예시

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND size("Image") >300
```

## DynamoDB용 PartiQL 산술, 비교 및 논리 연산자

Amazon DynamoDB의 PartiQL에서는 다음과 같은 SQL 표준 연산자를 지원합니다.

> **Note**
> 이 목록에 없는 모든 SQL 연산자는 현재 DynamoDB에서 지원되지 않습니다.

산술 연산자

| 연산자 | 설명 |
| --- | --- |
| + | 더하기 |
| - | 차감 |

비교 연산자

| 연산자 | 설명 |
| --- | --- |
| = | 같음 |
| <> | 같지 않음 |
| != | 같지 않음 |
| > | 보다 큼 |
| < | 보다 작음 |
| >= | 크거나 같음 |
| <= | 작거나 같음 |

논리 연산자

| 연산자 | 설명 |
| --- | --- |
| AND | AND로 구분된 조건이 모두 TRUE이면 TRUE |
| BETWEEN | 피연산자가 비교 범위 이내이면 TRUE입니다.<br><br>이 연산자는 해당 연산자를 적용하는 피연산자의 하한과 상한을 포함합니다. |
| IN | TRUE 피연산자가 표현식 목록 중 하나와 같은 경우(해시 속성 값 최대 50개 또는 키가 아닌 속성 값 최대 100개) |
| IS | 피연산자가 지정된 PartiQL 데이터 형식(NULL 또는 MISSING 포함)이면 TRUE |
| NOT | 지정된 부울 식의 값을 반대로 바꿈 |
| OR | OR로 구분된 조건 중 하나라도 TRUE이면 TRUE |

논리 연산자 사용에 대한 자세한 내용은 비교 실행 및 논리 평가 섹션을 참조하세요.

## DynamoDB용 PartiQL에서 트랜잭션 수행

이 단원에서는 DynamoDB용 PartiQL에서 트랜잭션을 사용하는 방법을 설명합니다. PartiQL 트랜잭션은 총 100개의 명령문(작업)으로 제한됩니다.

DynamoDB 트랜잭션에 대한 자세한 내용은 DynamoDB Transactions를 사용하여 복잡한 워크플로 관리를 참조하세요.

> **Note**
>
> 전체 트랜잭션은 읽기 또는 쓰기 문으로 구성해야 합니다. 하나의 트랜잭션에서 두 가지를 모두 혼용할 수는 없습니다. EXISTS 함수는 예외입니다. TransactWriteItems API 작업의 ConditionCheck와 유사한 방식으로 특정 항목 속성의 조건을 확인하는 데 사용할 수 있습니다.

주제


- 구문
- 파라미터
- 반환 값
- 예시


구문

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

파라미터

***##***

(필수) DynamoDB용 PartiQL에서 지원되는 문입니다.

> **Note**
>
> 전체 트랜잭션은 읽기 또는 쓰기 문으로 구성해야 합니다. 하나의 트랜잭션에서 두 가지를 모두 혼용할 수는 없습니다.

***parametertype***

(선택 사항) ParitPartiQL 문을 지정할 때 파라미터가 사용된 경우 DynamoDB 형식입니다.

***parametervalue***

(선택 사항) PartiQL 문을 지정할 때 파라미터가 사용된 경우 파라미터 값입니다.

반환 값

이 문은 쓰기 작업(INSERT, UPDATE 또는 DELETE)에 대한 값을 반환하지 않습니다. 그러나 WHERE 절에 지정된 조건에 따라 읽기 작업 (SELECT) 에 대해 서로 다른 값을 반환합니다.

> **Note**
>
> Singleton INSERT, UPDATE 또는 DELETE 작업 중 하나에서 오류를 반환하는 경우 트랜잭션이 `TransactionCanceledException` 예외로 취소되고 취소 이유 코드에 개별 singleton 작업의 오류가 포함됩니다.

예시

다음 예는 여러 문을 트랜잭션으로 실행합니다.

AWS CLI

1. 다음 JSON 코드를 partiql.json이라는 파일에 저장합니다.

```
[
    {
        "Statement": "EXISTS(SELECT * FROM \"Music\" where Artist='No One You
 Know' and SongTitle='Call Me Today' and Awards is  MISSING)"
    },
    {
        "Statement": "INSERT INTO Music value {'Artist':?,'SongTitle':'?'}",
        "Parameters": [{\"S\": \"Acme Band\"}, {\"S\": \"Best Song\"}]
    },
    {
        "Statement": "UPDATE \"Music\" SET AwardsWon=1 SET
 AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
 SongTitle='PartiQL Rocks'"
    }
]
```

2. 명령 프롬프트에서 다음 명령을 실행합니다.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlTransaction {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

        try {
            // Create ExecuteTransactionRequest
            ExecuteTransactionRequest executeTransactionRequest =
 createExecuteTransactionRequest();
            ExecuteTransactionResult executeTransactionResult =
 dynamoDB.executeTransaction(executeTransactionRequest);
            System.out.println("ExecuteTransaction successful.");
            // Handle executeTransactionResult

        } catch (Exception e) {
            handleExecuteTransactionErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {
        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static ExecuteTransactionRequest createExecuteTransactionRequest() {
        ExecuteTransactionRequest request = new ExecuteTransactionRequest();

        // Create statements
        List<ParameterizedStatement> statements = getPartiQLTransactionStatements();

        request.setTransactStatements(statements);
        return request;
    }

    private static List<ParameterizedStatement> getPartiQLTransactionStatements() {
        List<ParameterizedStatement> statements = new
 ArrayList<ParameterizedStatement>();

        statements.add(new ParameterizedStatement()
                               .withStatement("EXISTS(SELECT * FROM "Music" where
 Artist='No One You Know' and SongTitle='Call Me Today' and Awards is  MISSING)"));
```

```
        statements.add(new ParameterizedStatement()
                                .withStatement("INSERT INTO "Music" value
{'Artist':'?','SongTitle':'?'}")
                                .withParameters(new AttributeValue("Acme Band"),new
AttributeValue("Best Song")));

        statements.add(new ParameterizedStatement()
                                .withStatement("UPDATE "Music" SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
SongTitle='PartiQL Rocks'"));

        return statements;
    }

    // Handles errors during ExecuteTransaction execution. Use recommendations in
error messages below to add error handling specific to
    // your application use-case.
    private static void handleExecuteTransactionErrors(Exception exception) {
        try {
            throw exception;
        } catch (TransactionCanceledException tce) {
            System.out.println("Transaction Cancelled, implies a client issue, fix
before retrying. Error: " + tce.getErrorMessage());
        } catch (TransactionInProgressException tipe) {
            System.out.println("The transaction with the given request token is
already in progress, consider changing " +
                "retry strategy for this type of error. Error: " +
tipe.getErrorMessage());
        } catch (IdempotentParameterMismatchException ipme) {
            System.out.println("Request rejected because it was retried with a
different payload but with a request token that was already used, " +
                "change request token for this payload to be accepted. Error: " +
ipme.getErrorMessage());
        } catch (Exception e) {
            handleCommonErrors(e);
        }
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + isee.getErrorMessage());
```

```
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
 your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
 retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing
 provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
 before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
 it, and returned an error response instead. Investigate and " +
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
 Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
 service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

다음 예제에서는 DynamoDB가 WHERE 절에 지정된 조건이 다른 항목을 읽을 때 서로 다른 반환 값을 보여줍니다.

AWS CLI

1. 다음 JSON 코드를 partiql.json이라는 파일에 저장합니다.

```
[
```

```
    // Item exists and projected attribute exists
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One You Know' and
 SongTitle='Call Me Today'"
    },
    // Item exists but projected attributes do not exist
    {
        "Statement": "SELECT non_existent_projected_attribute FROM "Music" WHERE
 Artist='No One You Know' and SongTitle='Call Me Today'"
    },
    // Item does not exist
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One I Know' and
 SongTitle='Call You Today'"
    }
]
```

2. 명령 프롬프트에서 다음 명령.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

3. 다음 응답이 반환됩니다.

```
{
    "Responses": [
        // Item exists and projected attribute exists
        {
            "Item": {
                "Artist":{
                    "S": "No One You Know"
                },
                "SongTitle":{
                    "S": "Call Me Today"
                }
            }
        },
        // Item exists but projected attributes do not exist
        {
            "Item": {}
        },
        // Item does not exist
        {}
    ]
```

```
}
```

## DynamoDB용 PartiQL에서 일괄 작업 실행

이 단원에서는 DynamoDB용 PartiQL에서 배치 문을 사용하는 방법을 설명합니다.

> **Note**
>
> - 전체 배치는 읽기 문이나 쓰기 문 중 하나로 구성해야 하며, 하나의 배치에서 두 문을 함께 사용할 수는 없습니다.
> - `BatchExecuteStatement` 및 `BatchWriteItem`으로 배치당 25개 이하의 문을 실행할 수 있습니다.
> - `BatchExecuteStatement`는 별도의 문에서 프라이머리 키 목록을 가져오는 `BatchGetItem`을 사용합니다.

주제


- 구문
- 파라미터
- 예시


### 구문

```
[
  {
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#9StkWHYTxm7x2AqSXcrfu7' AND
 sk = 'info'"
  },
  {
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#isC2ChceGbxHgESc4szoTE' AND
 sk = 'info'"
  }
]
```

```
[
   {
      "Statement":" statement ",
```

```
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

파라미터

***##***

(필수) DynamoDB용 PartiQL에서 지원되는 문입니다.

> **Note**
>
> - 전체 배치는 읽기 문이나 쓰기 문 중 하나로 구성해야 하며, 하나의 배치에서 두 문을 함께 사용할 수는 없습니다.
> - `BatchExecuteStatement` 및 `BatchWriteItem`으로 배치당 25개 이하의 문을 실행할 수 있습니다.

***parametertype***

(선택 사항) ParitPartiQL 문을 지정할 때 파라미터가 사용된 경우 DynamoDB 형식입니다.

***parametervalue***

(선택 사항) PartiQL 문을 지정할 때 파라미터가 사용된 경우 파라미터 값입니다.

예시

AWS CLI

1. 다음 json을 partiql.json이라는 파일에 저장합니다.

```
[
   {
  "Statement": "INSERT INTO Music VALUE {'Artist':?,'SongTitle':?}",
   "Parameters": [{"S": "Acme Band"}, {"S": "Best Song"}]
 },
 {
```

```
  "Statement": "UPDATE Music SET AwardsWon=1, AwardDetail={'Grammys':[2020,
 2018]} WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
    }
]
```

2. 명령 프롬프트에서 다음 명령을 실행합니다.

```
aws dynamodb batch-execute-statement  --statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlBatch {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

        try {
            // Create BatchExecuteStatementRequest
            BatchExecuteStatementRequest batchExecuteStatementRequest =
 createBatchExecuteStatementRequest();
            BatchExecuteStatementResult batchExecuteStatementResult =
 dynamoDB.batchExecuteStatement(batchExecuteStatementRequest);
            System.out.println("BatchExecuteStatement successful.");
            // Handle batchExecuteStatementResult

        } catch (Exception e) {
            handleBatchExecuteStatementErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {

        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static BatchExecuteStatementRequest createBatchExecuteStatementRequest()
 {
        BatchExecuteStatementRequest request = new BatchExecuteStatementRequest();

        // Create statements
        List<BatchStatementRequest> statements = getPartiQLBatchStatements();
```

```
        request.setStatements(statements);
        return request;
    }

    private static List<BatchStatementRequest> getPartiQLBatchStatements() {
        List<BatchStatementRequest> statements = new
ArrayList<BatchStatementRequest>();

        statements.add(new BatchStatementRequest()
                               .withStatement("INSERT INTO Music value
{'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"));

        statements.add(new BatchStatementRequest()
                               .withStatement("UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist='Acme Band' and SongTitle='PartiQL
Rocks'"));

        return statements;
    }

    // Handles errors during BatchExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
    // your application use-case.
    private static void handleBatchExecuteStatementErrors(Exception exception) {
        try {
            throw exception;
        } catch (Exception e) {
            // There are no API specific errors to handle for BatchExecuteStatement,
common DynamoDB API errors are handled below
            handleCommonErrors(e);
        }
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
```

```
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
 retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing
 provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
 before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
 it, and returned an error response instead. Investigate and " +
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
 Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
 service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

## DynamoDB용 PartiQL을 사용하는 IAM 보안 정책

다음 권한이 필요합니다.

- DynamoDB용 PartiQL을 사용하여 항목을 읽으려면 테이블 또는 인덱스에 대한 `dynamodb:PartiQLSelect` 권한이 있어야 합니다.
- DynamoDB용 PartiQL을 사용하여 항목을 삽입하려면 테이블 또는 인덱스에 대한 `dynamodb:PartiQLInsert` 권한이 있어야 합니다.
- DynamoDB용 PartiQL을 사용하여 항목을 업데이트하려면 테이블 또는 인덱스에 대한 `dynamodb:PartiQLUpdate` 권한이 있어야 합니다.

- DynamoDB용 PartiQL을 사용하여 항목을 삭제하려면 테이블 또는 인덱스에 대한 `dynamodb:PartiQLDelete` 권한이 있어야 합니다.

예: 테이블에서 모든 DynamoDB용 PartiQL 문(Select/Insert/Update/Delete) 허용

다음 IAM 정책은 테이블에서 모든 DynamoDB용 PartiQL 문을 실행할 수 있는 권한을 부여합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

예: 테이블에서 DynamoDB용 PartiQL select 문 허용

다음 IAM 정책은 특정 테이블에서 `select` 문을 실행할 수 있는 권한을 부여합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
```

```
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

예: 테이블에서 DynamoDB용 PartiQL insert 문 허용

다음 IAM 정책은 특정 인덱스에서 `insert` 문을 실행할 수 있는 권한을 부여합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music/index/index1"
         ]
      }
   ]
}
```

예: 테이블에서 DynamoDB용 PartiQL 트랜잭션 문만 허용

다음 IAM 정책은 특정 테이블에서 트래잭션 문만 실행할 수 있는 권한을 부여합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
```

```
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      }
   ]
}
```

예: 테이블에서 DynamoDB용 PartiQL 비트랜잭션 읽기 및 쓰기는 허용하고 PartiQL 트랜잭션 읽기 및 쓰기는 차단

다음 IAM 정책은 DynamoDB용 PartiQL 비트랜잭션 읽기 및 쓰기를 실행할 권한은 부여하고 DynamoDB용 PartiQL 트랜잭션 읽기 및 쓰기는 차단합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
```

```
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

예: DynamoDB용 PartiQL에서 select 문은 허용하고 전체 테이블 스캔 문은 거부

다음 IAM 정책은 특정 테이블에서 `select` 문을 실행할 수 있는 권한은 부여하고 전체 테이블이 스캔되는 `select` 문은 차단합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
         ],
```

```
            "Condition":{
               "Bool":{
                  "dynamodb:FullTableScan":[
                     "true"
                  ]
               }
            }
         },
         {
            "Effect":"Allow",
            "Action":[
               "dynamodb:PartiQLSelect"
            ],
            "Resource":[
               "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
            ]
         }
      ]
   }
```

## 항목 작업: Java

AWS SDK for Java Document API를 사용하여 테이블의 Amazon DynamoDB 항목에 대한 일반적인 생성, 읽기, 업데이트 및 삭제(CRUD) 작업을 수행할 수 있습니다.

> **Note**
>
> 또한 SDK for Java에서는 객체 지속성 모델을 제공하므로 DynamoDB 테이블로 클라이언트 측 클래스를 매핑할 수 있습니다. 이러한 접근 방식을 활용하면 작성해야 할 코드가 줄어듭니다. 자세한 내용은 Java 1.x: DynamoDBMapper 단원을 참조하십시오.

이 단원에는 몇 가지 Java 문서 API 항목 작업을 수행하기 위한 Java 예제와 몇 가지 완전한 작업 예제가 나와 있습니다.

주제

- 항목 추가
- 항목 가져오기
- 일괄 쓰기 - 여러 항목 추가 및 삭제

- 일괄 가져오기: 여러 항목 가져오기
- 항목 업데이트
- 항목 삭제
- 예: AWS SDK for Java 문서 API를 사용하는 CRUD 작업
- 예: AWS SDK for Java Document API를 사용하는 일괄 작업
- 예: AWS SDK for Java 문서 API를 사용하여 이진 형식 속성 처리

## 항목 추가

putItem 메서드는 항목을 테이블에 저장합니다. 항목이 존재하는 경우 전체 항목을 바꿉니다. 전체 항목을 변경하지 않고 특정 속성만 업데이트하는 경우에는 updateItem 메서드를 사용하면 됩니다. 자세한 내용은 항목 업데이트 단원을 참조하십시오.

Java v2

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval> <awards>
<awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is placed
(for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example, Happy
Day).
                **Warning** This program will  place an item that you specify into a
table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
```

```
        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }

    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {

        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle, AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The request
id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
```

```
            System.exit(1);
        }
    }
}
```

Java v1

다음 단계를 따릅니다.

1. DynamoDB 클래스의 인스턴스를 만듭니다.
2. Table 클래스의 인스턴스를 만들어 사용할 테이블을 표시합니다.
3. Item 클래스 인스턴스를 생성하여 새로운 항목을 표시합니다. 새 항목의 기본 키와 그 속성을 지정해야 합니다.
4. 이전 단계에서 생성한 Item을 사용하여 Table 객체의 putItem 메서드를 호출합니다.

다음은 위에서 설명한 작업을 실행하는 Java 코드 예제입니다. 이 코드는 ProductCatalog 테이블에 새 항목을 씁니다.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

// Build a list of related items
List<Number> relatedItems = new ArrayList<Number>();
relatedItems.add(341);
relatedItems.add(472);
relatedItems.add(649);

//Build a map of product pictures
Map<String, String> pictures = new HashMap<String, String>();
pictures.put("FrontView", "http://example.com/products/123_front.jpg");
pictures.put("RearView", "http://example.com/products/123_rear.jpg");
pictures.put("SideView", "http://example.com/products/123_left_side.jpg");

//Build a map of product reviews
Map<String, List<String>> reviews = new HashMap<String, List<String>>();

List<String> fiveStarReviews = new ArrayList<String>();
```

```
fiveStarReviews.add("Excellent! Can't recommend it highly enough!  Buy it!");
fiveStarReviews.add("Do yourself a favor and buy this");
reviews.put("FiveStar", fiveStarReviews);

List<String> oneStarReviews = new ArrayList<String>();
oneStarReviews.add("Terrible product!  Do not buy this.");
reviews.put("OneStar", oneStarReviews);

// Build the item
Item item = new Item()
    .withPrimaryKey("Id", 123)
    .withString("Title", "Bicycle 123")
    .withString("Description", "123 description")
    .withString("BicycleType", "Hybrid")
    .withString("Brand", "Brand-Company C")
    .withNumber("Price", 500)
    .withStringSet("Color",  new HashSet<String>(Arrays.asList("Red", "Black")))
    .withString("ProductCategory", "Bicycle")
    .withBoolean("InStock", true)
    .withNull("QuantityOnHand")
    .withList("RelatedItems", relatedItems)
    .withMap("Pictures", pictures)
    .withMap("Reviews", reviews);

// Write the item to the table
PutItemOutcome outcome = table.putItem(item);
```

위의 예에서 항목에는 형식이 스칼라(`String`, `Number`, `Boolean`, `Null`), 집합(`String Set`) 및 문서(`List`, `Map`)인 속성이 있습니다.

### 옵션 파라미터 지정

필수 파라미터 외에도 `putItem` 메서드에 선택적 파라미터를 지정할 수 있습니다. 예를 들어 다음 Java 코드 예제는 선택적 파라미터를 사용하여 항목을 업로드하기 위한 조건을 지정합니다. 지정하는 조건을 충족하지 못하면 AWS SDK for Java가 `ConditionalCheckFailedException`을 내보냅니다. 이 코드 예제는 `putItem` 메서드에 다음과 같은 선택적 파라미터를 지정합니다.

- 요청에 대한 조건을 정의하는 `ConditionExpression`. 이 코드는 특정 값과 동일한 ISBN 속성을 가진 경우에만 동일한 기본 키를 가진 기존 항목이 대체되는 조건을 정의합니다.
- 조건에 사용되는 `ExpressionAttributeValues`에 대한 지도. 이 경우 단 한 번의 대체만 필요합니다. 런타임에 조건 표현식의 자리 표시자 `:val`이 검사할 실제 ISBN 값으로 대체됩니다.

다음 예제에서는 이 선택적 파라미터를 사용하여 새로운 책 항목을 추가합니다.

Example

```
Item item = new Item()
    .withPrimaryKey("Id", 104)
    .withString("Title", "Book 104 Title")
    .withString("ISBN", "444-4444444444")
    .withNumber("Price", 20)
    .withStringSet("Authors",
        new HashSet<String>(Arrays.asList("Author1", "Author2")));

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val", "444-4444444444");

PutItemOutcome outcome = table.putItem(
    item,
    "ISBN = :val", // ConditionExpression parameter
    null,          // ExpressionAttributeNames parameter - we're not using it for this
 example
    expressionAttributeValues);
```

PutItem 및 JSON 문서

DynamoDB 테이블에 속성으로 JSON 문서를 저장할 수 있습니다. 그러려면 `Item`의 `withJSON` 메서드를 사용합니다. 이 메서드가 JSON 문서를 구문 분석하고 각 요소를 기본 DynamoDB 데이터 형식에 매핑합니다.

특정 제품에 대한 주문을 이행할 수 있는 공급업체를 포함하여 다음 JSON 문서를 저장한다고 가정해 보십시오.

Example

```
{
    "V01": {
        "Name": "Acme Books",
        "Offices": [ "Seattle" ]
    },
    "V02": {
        "Name": "New Publishers, Inc.",
        "Offices": ["London", "New York"
        ]
```

```
    },
    "V03": {
        "Name": "Better Buy Books",
        "Offices": [ "Tokyo", "Los Angeles", "Sydney"
        ]
    }
}
```

withJSON 메서드를 사용하여 이 항목을 ProductCatalog 테이블의 VendorInfo라는 Map 속성에 저장할 수 있습니다. 다음은 이 작업을 수행하는 방법을 보여 주는 Java 코드 예제입니다.

```
// Convert the document into a String.  Must escape all double-quotes.
String vendorDocument = "{"
    + "    \"V01\": {"
    + "        \"Name\": \"Acme Books\","
    + "        \"Offices\": [ \"Seattle\" ]"
    + "    },"
    + "    \"V02\": {"
    + "        \"Name\": \"New Publishers, Inc.\","
    + "        \"Offices\": [ \"London\", \"New York\"" + "]" + "},"
    + "    \"V03\": {"
    + "        \"Name\": \"Better Buy Books\","
    +          "\"Offices\": [ \"Tokyo\", \"Los Angeles\", \"Sydney\""
    + "            ]"
    + "        }"
    + "    }";

Item item = new Item()
    .withPrimaryKey("Id", 210)
    .withString("Title", "Book 210 Title")
    .withString("ISBN", "210-2102102102")
    .withNumber("Price", 30)
    .withJSON("VendorInfo", vendorDocument);

PutItemOutcome outcome = table.putItem(item);
```

## 항목 가져오기

단일 항목을 검색하려면 Table 객체의 getItem 메서드를 사용합니다. 다음 단계를 따릅니다.

1. DynamoDB 클래스의 인스턴스를 만듭니다.
2. Table 클래스의 인스턴스를 만들어 사용할 테이블을 표시합니다.

3. Table 인스턴스의 getItem 메서드를 호출합니다. 검색하려는 항목의 기본 키를 지정해야 합니다.

다음 Java 코드 예는 앞의 단계를 보여줍니다. 이 코드는 지정된 파티션 기본 키가 있는 항목을 가져옵니다.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Item item = table.getItem("Id", 210);
```

옵션 파라미터 지정

필수 파라미터 외에도 getItem 메서드에 대해 선택적 파라미터를 지정할 수 있습니다. 예를 들어 다음 Java 코드 예제는 선택적 메서드를 사용하여 특정 속성 목록만 검색하고 강력하게 일관된 읽기를 지정합니다. 읽기 일관성에 대한 자세한 내용은 DynamoDB 읽기 일관성를 참조하십시오.

ProjectionExpression을 사용하여 전체 항목이 아닌 특정 속성이나 요소만 검색할 수 있습니다. ProjectionExpression은 문서 경로를 사용하여 최상위 또는 중첩된 속성을 지정할 수 있습니다. 자세한 내용은 DynamoDB에서 프로젝션 표현식 사용 단원을 참조하십시오.

getItem 메서드의 파라미터는 읽기 일관성을 지정할 수 없도록 합니다. 그러나 하위 수준 GetItem 작업에 대한 모든 입력에 전체 액세스 권한을 제공하는 GetItemSpec를 생성할 수 있습니다. 아래의 코드 예제에서는 GetItemSpec을 생성하고 그 사양을 getItem 메서드에 대한 입력으로 사용합니다.

Example

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 206)
    .withProjectionExpression("Id, Title, RelatedItems[0], Reviews.FiveStar")
    .withConsistentRead(true);

Item item = table.getItem(spec);

System.out.println(item.toJSONPretty());
```

사람이 읽을 수 있는 형식으로 Item을 인쇄하려면 toJSONPretty 메서드를 사용하십시오. 이전 예제의 출력은 다음과 같습니다.

```
{
  "RelatedItems" : [ 341 ],
  "Reviews" : {
    "FiveStar" : [ "Excellent! Can't recommend it highly enough! Buy it!", "Do yourself
 a favor and buy this" ]
  },
  "Id" : 123,
  "Title" : "20-Bicycle 123"
}
```

### GetItem 및 JSON 문서

PutItem 및 JSON 문서 단원에서는 JSON 문서를 VendorInfo라는 Map 속성에 저장했습니다. getItem 메서드를 사용하여 JSON 형식으로 전체 문서를 검색할 수 있습니다. 또는 문서 경로 표기법을 사용하여 문서에서 일부 요소만 검색할 수도 있습니다. 다음 Java 코드 예제는 이러한 기법을 보여줍니다.

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210);

System.out.println("All vendor info:");
spec.withProjectionExpression("VendorInfo");
System.out.println(table.getItem(spec).toJSON());

System.out.println("A single vendor:");
spec.withProjectionExpression("VendorInfo.V03");
System.out.println(table.getItem(spec).toJSON());

System.out.println("First office location for this vendor:");
spec.withProjectionExpression("VendorInfo.V03.Offices[0]");
System.out.println(table.getItem(spec).toJSON());
```

이전 예제의 출력은 다음과 같습니다.

```
All vendor info:
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]},"V02":{"Name":"New Publishers, Inc.","Offices":["London","New
 York"]},"V01":{"Name":"Acme Books","Offices":["Seattle"]}}}
A single vendor:
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]}}}
```

```
First office location for a single vendor:
{"VendorInfo":{"V03":{"Offices":["Tokyo"]}}}
```

> **Note**
>
> toJSON 메서드를 사용하여 임의의 항목 또는 해당 속성을 JSON 형식 문자열로 변환할 수 있습니다. 다음 코드 예제에서는 여러 최상위 및 중첩 속성을 검색하여 결과를 JSON으로 인쇄합니다.
>
> ```
> GetItemSpec spec = new GetItemSpec()
>     .withPrimaryKey("Id", 210)
>     .withProjectionExpression("VendorInfo.V01, Title, Price");
>
> Item item = table.getItem(spec);
> System.out.println(item.toJSON());
> ```
>
> 출력은 다음과 같습니다.
>
> ```
> {"VendorInfo":{"V01":{"Name":"Acme Books","Offices":
> ["Seattle"]}},"Price":30,"Title":"Book 210 Title"}
> ```

## 일괄 쓰기 - 여러 항목 추가 및 삭제

배치 쓰기란 다수의 항목을 배치로 입력 및 삭제하는 것을 말합니다. batchWriteItem 메서드를 사용하면 단 한 번의 호출로 하나 이상의 테이블에 다수의 항목을 업로드하거나 삭제할 수 있습니다. 다음은 AWS SDK for Java Document API를 사용하여 여러 항목을 일괄하거나 삭제하는 단계입니다.

1. DynamoDB 클래스의 인스턴스를 만듭니다.
2. 테이블의 모든 일괄 및 삭제 작업을 설명하는 TableWriteItems 클래스의 인스턴스를 만듭니다. 단일 일괄 쓰기 작업에서 여러 테이블에 쓰려면 테이블마다 TableWriteItems 인스턴스 하나를 생성해야 합니다.
3. 이전 단계에서 생성한 TableWriteItems 객체를 제공하여 batchWriteItem 메서드를 호출합니다.
4. 응답을 처리합니다. 응답과 함께 반환되는 요청 항목 중 처리하지 않은 항목이 있는지 확인해야 합니다. 프로비저닝된 처리량이 할당량에 이르거나 일시적 오류가 발생하면 이러한 경우가 발생할 수 있습니다. 또한 DynamoDB는 요청 시 지정하는 요청 크기나 작업 수를 제한합니다. 따라

서 이러한 제한을 초과할 경우 DynamoDB가 요청을 거부하기도 합니다. 자세한 내용은 Amazon DynamoDB의 할당량 단원을 참조하십시오.

다음 Java 코드 예는 앞의 단계를 보여줍니다. 예제에서는 두 테이블 `Forum` 및 `Thread`에서 `batchWriteItem` 작업을 수행합니다. 해당 `TableWriteItems` 객체가 다음 작업을 정의합니다.

- `Forum` 테이블에 항목을 업로드합니다.
- `Thread` 테이블에서 항목을 업로드하고 삭제합니다.

그런 다음 코드가 `batchWriteItem`을 호출하여 작업을 수행합니다.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

TableWriteItems forumTableWriteItems = new TableWriteItems("Forum")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("Name", "Amazon RDS")
            .withNumber("Threads", 0));

TableWriteItems threadTableWriteItems = new TableWriteItems("Thread")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("ForumName","Amazon RDS","Subject","Amazon RDS Thread 1")
    .withHashAndRangeKeysToDelete("ForumName","Some partition key value", "Amazon S3",
 "Some sort key value");

BatchWriteItemOutcome outcome = dynamoDB.batchWriteItem(forumTableWriteItems,
 threadTableWriteItems);

// Code for checking unprocessed items is omitted in this example
```

사용 가능한 예제는 예: AWS SDK for Java 문서 API를 사용하는 일괄 쓰기 작업를 참조하세요.

## 일괄 가져오기: 여러 항목 가져오기

`batchGetItem` 메서드를 사용하면 하나 이상의 테이블에서 여러 항목을 검색할 수 있습니다. 단일 항목을 검색하려면 `getItem` 메서드를 사용합니다.

다음 단계를 따릅니다.

1. DynamoDB 클래스의 인스턴스를 만듭니다.
2. 테이블에서 검색할 기본 키 값 목록을 설명하는 TableKeysAndAttributes 클래스의 인스턴스를 만듭니다. 단일 일괄 가져오기 작업에서 여러 테이블로부터 읽으려면 테이블마다 TableKeysAndAttributes 인스턴스 하나를 만들어야 합니다.
3. 이전 단계에서 생성한 TableKeysAndAttributes 객체를 제공하여 batchGetItem 메서드를 호출합니다.

다음 Java 코드 예는 앞의 단계를 보여줍니다. 이 예제에서는 Forum 테이블에서 항목 2개, Thread 테이블에서 항목 3개를 검색합니다.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

    TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes(forumTableName);
    forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

TableKeysAndAttributes threadTableKeysAndAttributes = new
 TableKeysAndAttributes(threadTableName);
threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName", "Subject",
    "Amazon DynamoDB","DynamoDB Thread 1",
    "Amazon DynamoDB","DynamoDB Thread 2",
    "Amazon S3","S3 Thread 1");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(
    forumTableKeysAndAttributes, threadTableKeysAndAttributes);

for (String tableName : outcome.getTableItems().keySet()) {
    System.out.println("Items in table " + tableName);
    List<Item> items = outcome.getTableItems().get(tableName);
    for (Item item : items) {
        System.out.println(item);
    }
}
```

### 옵션 파라미터 지정

batchGetItem을 사용할 때 필수 파라미터 외에도 선택적 파라미터를 지정할 수 있습니다. 예를 들어 TableKeysAndAttributes를 정의할 때마다 ProjectionExpression을 제공할 수 있습니다. 그러면 테이블에서 검색하려는 속성을 지정할 수 있습니다.

다음 코드 예제에서는 Forum 테이블에서 두 개의 항목을 검색합니다. withProjectionExpression 파라미터가 Threads 속성만 검색하도록 지정합니다.

Example

```
TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes("Forum")
    .withProjectionExpression("Threads");

forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(forumTableKeysAndAttributes);
```

## 항목 업데이트

Table 객체의 updateItem 메서드는 기존의 속성 값을 업데이트하거나, 새 속성을 추가하거나, 기존 항목에서 속성을 삭제할 수 있습니다.

updateItem 메서드는 다음과 같이 동작합니다.

- 항목이 없으면(테이블에 지정된 기본 키를 가진 항목이 없음) updateItem이 테이블에 새 항목을 추가합니다.
- 항목이 있으면 UpdateExpression 파라미터가 지정한 대로 updateItem이 업데이트를 수행합니다.

> **Note**
>
> putItem을 사용하여 항목을 "업데이트"할 수도 있습니다. 예를 들어 putItem을 호출하여 테이블에 항목을 추가할 경우 지정된 기본 키를 가진 항목이 이미 있으면 putItem이 전체 항목을 바꿉니다. 입력에 지정되지 않은 속성이 기존 항목에 있으면 putItem이 항목에서 그 속성을 제거합니다.

일반적으로 항목 속성을 수정할 때마다 updateItem을 사용하는 것이 좋습니다. updateItem 메서드는 입력에 지정한 항목 속성만 수정하며, 항목에 있는 다른 속성은 변경되지 않고 남아 있습니다.

다음 단계를 따릅니다.

1. Table 클래스의 인스턴스를 만들어 사용할 테이블을 표시합니다.
2. Table 인스턴스의 updateTable 메서드를 호출합니다. 수정할 속성과 수정 방법을 설명하는 UpdateExpression과 함께 검색하려는 항목의 기본 키를 지정해야 합니다.

다음은 위에서 설명한 작업을 실행하는 Java 코드 예제입니다. 이 코드에서는 ProductCatalog 테이블의 book 항목을 업데이트합니다. 새로운 작성자를 Authors 집합에 추가하고 기존 ISBN 속성을 삭제합니다. 또한 가격을 1만큼 내립니다.

ExpressionAttributeValues 맵이 UpdateExpression에 사용됩니다. 런타임에 자리 표시자 :val1 및 :val2가 Authors 및 Price의 실제 값으로 바뀝니다.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#A", "Authors");
expressionAttributeNames.put("#P", "Price");
expressionAttributeNames.put("#I", "ISBN");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1",
    new HashSet<String>(Arrays.asList("Author YY","Author ZZ")));
expressionAttributeValues.put(":val2", 1);   //Price

UpdateItemOutcome outcome =  table.updateItem(
    "Id",          // key attribute name
    101,           // key attribute value
    "add #A :val1 set #P = #P - :val2 remove #I", // UpdateExpression
    expressionAttributeNames,
    expressionAttributeValues);
```

### 옵션 파라미터 지정

필수 파라미터 외에도 업데이트가 발생하기 위해 충족되어야 하는 조건 등 updateItem 메서드에 대한 선택적 파라미터도 지정할 수 있습니다. 지정하는 조건을 충족하지 못하면 AWS SDK for Java가 ConditionalCheckFailedException을 내보냅니다. 예를 들어 다음 Java 코드 예제는 book 항목 가격을 25로 조건부 업데이트합니다. 기존 가격이 20일 경우에만 가격이 업데이트 되도록 ConditionExpression을 지정합니다.

Example

```
Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#P", "Price");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1", 25);  // update Price to 25...
expressionAttributeValues.put(":val2", 20);  //...but only if existing Price is 20

UpdateItemOutcome outcome = table.updateItem(
    new PrimaryKey("Id",101),
    "set #P = :val1", // UpdateExpression
    "#P = :val2",     // ConditionExpression
    expressionAttributeNames,
    expressionAttributeValues);
```

### 원자성 카운터

updateItem을 사용하여 원자성 카운터를 구현할 수 있습니다. 이는 다른 쓰기 요청과 충돌하지 않고도 기존 속성의 값을 증감합니다. 원자성 카운터를 증가시키려면 UpdateExpression을 set 작업과 함께 사용하여 Number 형식의 기존 속성에 숫자 값을 추가합니다.

다음 예제는 Quantity 속성을 하나씩 늘리며 이를 보여 줍니다. UpdateExpression에서 ExpressionAttributeNames 파라미터가 사용되는 방식도 보여 줍니다.

```
Table table = dynamoDB.getTable("ProductCatalog");

Map<String,String> expressionAttributeNames = new HashMap<String,String>();
expressionAttributeNames.put("#p", "PageCount");

Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", 1);
```

```
UpdateItemOutcome outcome = table.updateItem(
    "Id", 121,
    "set #p = #p + :val",
    expressionAttributeNames,
    expressionAttributeValues);
```

## 항목 삭제

deleteItem 메서드는 테이블에서 항목을 삭제합니다. 삭제할 항목의 기본 키를 제공해야 합니다.

다음 단계를 따릅니다.

1. DynamoDB 클라이언트의 인스턴스를 만듭니다.
2. 삭제하려는 항목의 키를 제공하여 deleteItem 메서드를 호출합니다.

다음은 Java 예제는 이러한 작업을 보여줍니다.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

DeleteItemOutcome outcome = table.deleteItem("Id", 101);
```

옵션 파라미터 지정

deleteItem에 대해 선택적 파라미터를 지정할 수 있습니다. 예를 들어 다음 Java 코드 예제는 책이 더 이상 출판되지 않는 경우에만(InPublication 속성이 false임) ProductCatalog의 book 항목을 삭제하도록 ConditionExpression을 지정합니다.

Example

```
Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", false);

DeleteItemOutcome outcome = table.deleteItem("Id",103,
    "InPublication = :val",
    null, // ExpressionAttributeNames - not used in this example
```

```
    expressionAttributeValues);
```

## 예: AWS SDK for Java 문서 API를 사용하는 CRUD 작업

다음 코드 예제는 Amazon DynamoDB 항목에 대한 CRUD 작업을 보여줍니다. 이 예에서는 항목을 만들고, 검색하고, 다양한 업데이트를 수행하고, 마지막으로 항목을 삭제합니다.

> **Note**
>
> 또한 SDK for Java에서는 객체 지속성 모델을 제공하므로 DynamoDB 테이블로 클라이언트 측 클래스를 매핑할 수 있습니다. 이러한 접근 방식을 활용하면 작성해야 할 코드가 줄어듭니다. 자세한 내용은 Java 1.x: DynamoDBMapper 섹션을 참조하세요.

> **Note**
>
> 아래 코드 예제는 DynamoDB에서 테이블 생성 및 코드 예시에 대한 데이터 로드 단원의 지침에 따라 이미 계정의 DynamoDB에 데이터를 로드하였다고 가정한 것입니다.
> 다음 예제를 실행하기 위한 단계별 지침은 Java 코드 예 섹션을 참조하세요.

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DeleteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.UpdateItemOutcome;
import com.amazonaws.services.dynamodbv2.document.spec.DeleteItemSpec;
import com.amazonaws.services.dynamodbv2.document.spec.UpdateItemSpec;
import com.amazonaws.services.dynamodbv2.document.utils.NameMap;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
```

```
import com.amazonaws.services.dynamodbv2.model.ReturnValue;

public class DocumentAPIItemCRUDExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "ProductCatalog";

    public static void main(String[] args) throws IOException {

        createItems();

        retrieveItem();

        // Perform various updates.
        updateMultipleAttributes();
        updateAddNewAttribute();
        updateExistingAttributeConditionally();

        // Delete the item.
        deleteItem();

    }

    private static void createItems() {

        Table table = dynamoDB.getTable(tableName);
        try {

            Item item = new Item().withPrimaryKey("Id", 120).withString("Title", "Book
 120 Title")
                    .withString("ISBN", "120-1111111111")
                    .withStringSet("Authors", new
 HashSet<String>(Arrays.asList("Author12", "Author22")))
                    .withNumber("Price", 20).withString("Dimensions",
 "8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", false).withString("ProductCategory",
 "Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 121).withString("Title", "Book 121
 Title")
                    .withString("ISBN", "121-1111111111")
```

```
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author21", "Author 22")))
                    .withNumber("Price", 20).withString("Dimensions",
"8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Create items failed.");
            System.err.println(e.getMessage());

        }
    }

    private static void retrieveItem() {
        Table table = dynamoDB.getTable(tableName);

        try {

            Item item = table.getItem("Id", 120, "Id, ISBN, Title, Authors", null);

            System.out.println("Printing item after retrieving it....");
            System.out.println(item.toJSONPretty());

        } catch (Exception e) {
            System.err.println("GetItem failed.");
            System.err.println(e.getMessage());
        }

    }

    private static void updateAddNewAttribute() {
        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
121)
                    .withUpdateExpression("set #na = :val1").withNameMap(new
NameMap().with("#na", "NewAttribute"))
                    .withValueMap(new ValueMap().withString(":val1", "Some value"))
                    .withReturnValues(ReturnValue.ALL_NEW);
```

```
            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after adding new attribute...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to add new attribute in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void updateMultipleAttributes() {

        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
120)
                    .withUpdateExpression("add #a :val1 set #na=:val2")
                    .withNameMap(new NameMap().with("#a", "Authors").with("#na",
"NewAttribute"))
                    .withValueMap(
                            new ValueMap().withStringSet(":val1", "Author YY", "Author
ZZ").withString(":val2",
                                    "someValue"))
                    .withReturnValues(ReturnValue.ALL_NEW);

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after multiple attribute update...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to update multiple attributes in " + tableName);
            System.err.println(e.getMessage());

        }
    }

    private static void updateExistingAttributeConditionally() {
```

```
        Table table = dynamoDB.getTable(tableName);

        try {

            // Specify the desired price (25.00) and also the condition (price =
            // 20.00)

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
120)
                    .withReturnValues(ReturnValue.ALL_NEW).withUpdateExpression("set #p
= :val1")
                    .withConditionExpression("#p = :val2").withNameMap(new
NameMap().with("#p", "Price"))
                    .withValueMap(new ValueMap().withNumber(":val1",
25).withNumber(":val2", 20));

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after conditional update to new
attribute...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Error updating item in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void deleteItem() {

        Table table = dynamoDB.getTable(tableName);

        try {

            DeleteItemSpec deleteItemSpec = new DeleteItemSpec().withPrimaryKey("Id",
120)
                    .withConditionExpression("#ip = :val").withNameMap(new
NameMap().with("#ip", "InPublication"))
                    .withValueMap(new ValueMap().withBoolean(":val",
false)).withReturnValues(ReturnValue.ALL_OLD);

            DeleteItemOutcome outcome = table.deleteItem(deleteItemSpec);
```

```
            // Check the response.
            System.out.println("Printing item that was deleted...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Error deleting item in " + tableName);
            System.err.println(e.getMessage());
        }
    }
}
```

## 예: AWS SDK for Java Document API를 사용하는 일괄 작업

이 단원에서는 AWS SDK for Java 문서 API를 사용하여 Amazon DynamoDB에서 배치 쓰기 및 배치 가져오기를 수행하는 작업의 예를 제공합니다.

> **Note**
>
> 또한 SDK for Java에서는 객체 지속성 모델을 제공하므로 DynamoDB 테이블로 클라이언트 측 클래스를 매핑할 수 있습니다. 이러한 접근 방식을 활용하면 작성해야 할 코드가 줄어듭니다. 자세한 내용은 Java 1.x: DynamoDBMapper 단원을 참조하십시오.

주제

- 예: AWS SDK for Java 문서 API를 사용하는 일괄 쓰기 작업
- 예: AWS SDK for Java 문서 API를 사용하는 일괄 가져오기 작업

### 예: AWS SDK for Java 문서 API를 사용하는 일괄 쓰기 작업

다음 Java 코드 예제에서는 `batchWriteItem` 메서드를 사용하여 다음의 일괄 및 삭제 작업을 수행합니다.

- 한 항목을 `Forum` 테이블에 업로드합니다.
- 한 항목을 `Thread` 테이블에서 업로드하고 삭제합니다.

일괄 쓰기 요청을 생성할 때는 하나 이상의 테이블에 대해 업로드 및 삭제 요청을 얼마든지 지정할 수 있습니다. 하지만 `batchWriteItem`은 단일 일괄 쓰기 작업에서 일괄 쓰기 요청의 크기와 업로드 및 삭제 작업의 수를 제한합니다. 이러한 제한을 초과할 경우에는 요청이 거부됩니다. 이러한 요청을 처리

하는 데 충분한 처리량이 테이블에 할당되어 있지 않은 경우에는 응답 시 처리되지 않은 요청 항목이 반환됩니다.

다음은 응답을 확인하여 처리되지 않은 요청 항목의 유무를 점검하는 예제입니다. 미처리 요청 항목이 있는 경우에는 루프백이 발생하여 미처리 항목이 있는 batchWriteItem 요청을 다시 보냅니다. 이 가이드의 예제를 준수한 경우 이미 Forum 및 Thread 테이블이 생성되어 있습니다. 이러한 테이블은 프로그래밍 방식으로 생성하여 업로드할 수도 있습니다. 자세한 내용은 AWS SDK for Java를 사용한 예시 테이블 생성 및 데이터 업로드 단원을 참조하십시오.

다음 샘플을 테스트하기 위한 단계별 지침은 Java 코드 예 섹션을 참조하세요.

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.BatchWriteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.TableWriteItems;
import com.amazonaws.services.dynamodbv2.model.WriteRequest;

public class DocumentAPIBatchWrite {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {

        writeMultipleItemsBatchWrite();

    }
```

```
   private static void writeMultipleItemsBatchWrite() {
       try {

           // Add a new item to Forum
           TableWriteItems forumTableWriteItems = new
TableWriteItems(forumTableName) // Forum
                   .withItemsToPut(new Item().withPrimaryKey("Name", "Amazon
RDS").withNumber("Threads", 0));

           // Add a new item, and delete an existing item, from Thread
           // This table has a partition key and range key, so need to specify
           // both of them
           TableWriteItems threadTableWriteItems = new
TableWriteItems(threadTableName)
                   .withItemsToPut(
                           new Item().withPrimaryKey("ForumName", "Amazon RDS",
"Subject", "Amazon RDS Thread 1")
                                   .withString("Message", "ElastiCache Thread 1
message")
                                   .withStringSet("Tags", new
HashSet<String>(Arrays.asList("cache", "in-memory"))))
                   .withHashAndRangeKeysToDelete("ForumName", "Subject", "Amazon S3",
"S3 Thread 100");

           System.out.println("Making the request.");
           BatchWriteItemOutcome outcome =
dynamoDB.batchWriteItem(forumTableWriteItems, threadTableWriteItems);

           do {

               // Check for unprocessed keys which could happen if you exceed
               // provisioned throughput

               Map<String, List<WriteRequest>> unprocessedItems =
outcome.getUnprocessedItems();

               if (outcome.getUnprocessedItems().size() == 0) {
                   System.out.println("No unprocessed items found");
               } else {
                   System.out.println("Retrieving the unprocessed items");
                   outcome = dynamoDB.batchWriteItemUnprocessed(unprocessedItems);
               }
```

```
            } while (outcome.getUnprocessedItems().size() > 0);

        } catch (Exception e) {
            System.err.println("Failed to retrieve items: ");
            e.printStackTrace(System.err);
        }

    }

}
```

예: AWS SDK for Java 문서 API를 사용하는 일괄 가져오기 작업

다음 Java 코드 예제에서는 `batchGetItem` 메서드를 사용하여 `Forum` 및 `Thread` 테이블에서 여러 항목을 검색합니다. `BatchGetItemRequest`는 테이블 이름을 비롯해 가져올 각 항목의 키 목록을 지정합니다. 이 예제는 가져온 항목을 출력하여 응답을 처리합니다.

> **Note**
>
> 아래 코드 예제는 DynamoDB에서 테이블 생성 및 코드 예시에 대한 데이터 로드 단원의 지침에 따라 이미 계정의 DynamoDB에 데이터를 로드하였다고 가정한 것입니다.
> 다음 예제를 실행하기 위한 단계별 지침은 Java 코드 예 섹션을 참조하세요.

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.List;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.BatchGetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.TableKeysAndAttributes;
import com.amazonaws.services.dynamodbv2.model.KeysAndAttributes;
```

```
public class DocumentAPIBatchGet {
    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {
        retrieveMultipleItemsBatchGet();
    }

    private static void retrieveMultipleItemsBatchGet() {

        try {

            TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes(forumTableName);
            // Add a partition key
            forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name", "Amazon S3",
 "Amazon DynamoDB");

            TableKeysAndAttributes threadTableKeysAndAttributes = new
 TableKeysAndAttributes(threadTableName);
            // Add a partition key and a sort key
            threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName",
 "Subject", "Amazon DynamoDB",
                    "DynamoDB Thread 1", "Amazon DynamoDB", "DynamoDB Thread 2",
 "Amazon S3", "S3 Thread 1");

            System.out.println("Making the request.");

            BatchGetItemOutcome outcome =
 dynamoDB.batchGetItem(forumTableKeysAndAttributes,
                    threadTableKeysAndAttributes);

            Map<String, KeysAndAttributes> unprocessed = null;

            do {
                for (String tableName : outcome.getTableItems().keySet()) {
                    System.out.println("Items in table " + tableName);
                    List<Item> items = outcome.getTableItems().get(tableName);
                    for (Item item : items) {
                        System.out.println(item.toJSONPretty());
                    }
```

```
                }

                // Check for unprocessed keys which could happen if you exceed
                // provisioned
                // throughput or reach the limit on response size.
                unprocessed = outcome.getUnprocessedKeys();

                if (unprocessed.isEmpty()) {
                    System.out.println("No unprocessed keys found");
                } else {
                    System.out.println("Retrieving the unprocessed keys");
                    outcome = dynamoDB.batchGetItemUnprocessed(unprocessed);
                }

            } while (!unprocessed.isEmpty());

        } catch (Exception e) {
            System.err.println("Failed to retrieve items.");
            System.err.println(e.getMessage());
        }

    }

}
```

## 예: AWS SDK for Java 문서 API를 사용하여 이진 형식 속성 처리

다음은 이진 형식 속성의 처리를 설명하는 Java 코드 예제입니다. 이 예제에서는 항목을 `Reply` 테이블에 추가합니다. 추가된 항목에는 압축 데이터가 저장된 이진수 형식의 속성(`ExtendedMessage`)이 포함되어 있습니다. 그런 다음 항목을 가져와서 모든 속성 값을 출력합니다. 이해를 돕기 위해 예제에서는 `GZIPOutputStream` 클래스를 사용해 샘플 스트림을 압축하여 `ExtendedMessage` 속성에 할당합니다. 이진 속성을 검색하면 `GZIPInputStream` 클래스를 사용하여 이 속성의 압축이 풀립니다.

> **Note**
>
> 또한 SDK for Java에서는 객체 지속성 모델을 제공하므로 DynamoDB 테이블로 클라이언트 측 클래스를 매핑할 수 있습니다. 이러한 접근 방식을 활용하면 작성해야 할 코드가 줄어듭니다. 자세한 내용은 Java 1.x: DynamoDBMapper 단원을 참조하십시오.

DynamoDB에서 테이블 생성 및 코드 예시에 대한 데이터 로드 단원을 따랐다면 이미 Reply 테이블이 생성되어 있습니다. 이 테이블을 프로그래밍 방식으로 생성할 수도 있습니다. 자세한 내용은 AWS SDK for Java를 사용한 예시 테이블 생성 및 데이터 업로드 단원을 참조하십시오.

다음 샘플을 테스트하기 위한 단계별 지침은 Java 코드 예 섹션을 참조하세요.

Example

```
package com.amazonaws.codesamples.document;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import java.util.zip.GZIPInputStream;
import java.util.zip.GZIPOutputStream;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.GetItemSpec;

public class DocumentAPIItemBinaryExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "Reply";
    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    public static void main(String[] args) throws IOException {
        try {

            // Format the primary key values
            String threadId = "Amazon DynamoDB#DynamoDB Thread 2";
```

```
            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));
            String replyDateTime = dateFormatter.format(new Date());

            // Add a new reply with a binary attribute type
            createItem(threadId, replyDateTime);

            // Retrieve the reply with a binary attribute type
            retrieveItem(threadId, replyDateTime);

            // clean up by deleting the item
            deleteItem(threadId, replyDateTime);
        } catch (Exception e) {
            System.err.println("Error running the binary attribute type example: " +
e);
            e.printStackTrace(System.err);
        }
    }

    public static void createItem(String threadId, String replyDateTime) throws
IOException {

        Table table = dynamoDB.getTable(tableName);

        // Craft a long message
        String messageInput = "Long message to be compressed in a lengthy forum reply";

        // Compress the long message
        ByteBuffer compressedMessage = compressString(messageInput.toString());

        table.putItem(new Item().withPrimaryKey("Id",
threadId).withString("ReplyDateTime", replyDateTime)
                .withString("Message", "Long message
follows").withBinary("ExtendedMessage", compressedMessage)
                .withString("PostedBy", "User A"));
    }

    public static void retrieveItem(String threadId, String replyDateTime) throws
IOException {

        Table table = dynamoDB.getTable(tableName);

        GetItemSpec spec = new GetItemSpec().withPrimaryKey("Id", threadId,
"ReplyDateTime", replyDateTime)
                .withConsistentRead(true);
```

```
        Item item = table.getItem(spec);

        // Uncompress the reply message and print
        String uncompressed =
uncompressString(ByteBuffer.wrap(item.getBinary("ExtendedMessage")));

        System.out.println("Reply message:\n" + " Id: " + item.getString("Id") + "\n" +
" ReplyDateTime: "
                + item.getString("ReplyDateTime") + "\n" + " PostedBy: " +
item.getString("PostedBy") + "\n"
                + " Message: "
                + item.getString("Message") + "\n" + " ExtendedMessage (uncompressed):
" + uncompressed + "\n");
    }

    public static void deleteItem(String threadId, String replyDateTime) {

        Table table = dynamoDB.getTable(tableName);
        table.deleteItem("Id", threadId, "ReplyDateTime", replyDateTime);
    }

    private static ByteBuffer compressString(String input) throws IOException {
        // Compress the UTF-8 encoded String into a byte[]
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPOutputStream os = new GZIPOutputStream(baos);
        os.write(input.getBytes("UTF-8"));
        os.close();
        baos.close();
        byte[] compressedBytes = baos.toByteArray();

        // The following code writes the compressed bytes to a ByteBuffer.
        // A simpler way to do this is by simply calling
        // ByteBuffer.wrap(compressedBytes);
        // However, the longer form below shows the importance of resetting the
        // position of the buffer
        // back to the beginning of the buffer if you are writing bytes directly
        // to it, since the SDK
        // will consider only the bytes after the current position when sending
        // data to DynamoDB.
        // Using the "wrap" method automatically resets the position to zero.
        ByteBuffer buffer = ByteBuffer.allocate(compressedBytes.length);
        buffer.put(compressedBytes, 0, compressedBytes.length);
        buffer.position(0); // Important: reset the position of the ByteBuffer
```

```
                            // to the beginning
        return buffer;
    }

    private static String uncompressString(ByteBuffer input) throws IOException {
        byte[] bytes = input.array();
        ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPInputStream is = new GZIPInputStream(bais);

        int chunkSize = 1024;
        byte[] buffer = new byte[chunkSize];
        int length = 0;
        while ((length = is.read(buffer, 0, chunkSize)) != -1) {
            baos.write(buffer, 0, length);
        }

        String result = new String(baos.toByteArray(), "UTF-8");

        is.close();
        baos.close();
        bais.close();

        return result;
    }
}
```

## 항목 작업: .NET

AWS SDK for .NET 하위 수준 API를 사용하여 테이블 항목에 대한 생성, 읽기, 업데이트, 삭제 작업(CRUD)을 수행할 수 있습니다. 다음은 .NET 하위 수준 API를 사용하여 데이터 CRUD를 작업할 때 따라야 할 공통 단계입니다.

1. `AmazonDynamoDBClient` 클래스(클라이언트)의 인스턴스를 만듭니다.
2. 각 요청 객체에서 작업에 따라 필요한 파라미터를 입력합니다.

   예를 들어 항목을 업로드할 때는 `PutItemRequest` 요청 객체를 사용하고, 기존 항목을 가져올 때는 `GetItemRequest` 요청 객체를 사용합니다.

   요청 객체를 사용하여 필수 파라미터와 옵션 파라미터를 모두 입력할 수도 있습니다.
3. 이전 단계에서 생성한 요청 객체를 전달하여 클라이언트 제공 메서드를 실행합니다.

CRUD 작업을 위해 AmazonDynamoDBClient 클라이언트가 제공하는 메서드는 PutItem, GetItem, UpdateItem 및 DeleteItem입니다.

주제


- 항목 추가
- 항목 가져오기
- 항목 업데이트
- 원자성 카운터
- 항목 삭제
- 일괄 쓰기 - 여러 항목 추가 및 삭제
- 일괄 가져오기: 여러 항목 가져오기
- 예: AWS SDK for .NET 하위 수준 API를 사용하는 CRUD 작업
- 예: AWS SDK for .NET 하위 수준 API를 사용하는 일괄 작업
- 예: AWS SDK for .NET 하위 수준 API를 사용하여 이진 형식 속성 처리


## 항목 추가

PutItem 메서드는 항목을 테이블에 업로드합니다. 항목이 존재하는 경우 전체 항목을 바꿉니다.

> **Note**
>
> 전체 항목을 변경하지 않고 특정 속성만 업데이트하는 경우에는 UpdateItem 메서드를 사용하면 됩니다. 자세한 내용은 항목 업데이트 단원을 참조하십시오.

다음은 하위 수준 .NET SDK API를 사용하여 항목을 업로드하는 단계입니다.

1. AmazonDynamoDBClient 클래스의 인스턴스를 만듭니다.
2. PutItemRequest 클래스 인스턴스를 생성하여 필요한 파라미터를 입력합니다.

   항목을 업로드하려면 테이블 이름과 항목을 입력해야 합니다.
3. 이전 단계에서 생성한 PutItemRequest 객체를 입력하여 PutItem 메서드를 실행합니다.

다음 C# 예제에서는 이전 단계를 설명합니다. 이 예제에서는 항목을 ProductCatalog 테이블에 업로드합니다.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new PutItemRequest
{
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
      {
          { "Id", new AttributeValue { N = "201" }},
          { "Title", new AttributeValue { S = "Book 201 Title" }},
          { "ISBN", new AttributeValue { S = "11-11-11-11" }},
          { "Price", new AttributeValue { S = "20.00" }},
          {
            "Authors",
            new AttributeValue
            { SS = new List<string>{"Author1", "Author2"}   }
          }
      }
};
client.PutItem(request);
```

위 예제에서 업로드한 book 항목에는 Id, Title, ISBN 및 Authors 속성이 있습니다. Id는 숫자 형식의 속성이고, 나머지 모든 속성들은 문자열 형식입니다. 작성자는 String 집합입니다.

옵션 파라미터 지정

아래 C# 예제에서와 같이 PutItemRequest 객체를 사용하여 옵션 파라미터를 지정할 수도 있습니다. 이 예제에서 지정하는 옵션 파라미터는 다음과 같습니다.

- ExpressionAttributeNames, ExpressionAttributeValues 및 ConditionExpression 파라미터는 기존 항목에 특정 값의 ISBN 속성이 있는 경우에 한해 항목을 변경할 수 있도록 지정합니다.
- ReturnValues 파라미터는 응답 시 이전 항목을 요청합니다.

Example

```
var request = new PutItemRequest
 {
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
               {
                   { "Id", new AttributeValue { N = "104" }},
                   { "Title", new AttributeValue { S = "Book 104  Title" }},
                   { "ISBN", new AttributeValue { S = "444-4444444444" }},
                   { "Authors",
                     new AttributeValue { SS = new List<string>{"Author3"}}}
               },
    // Optional parameters.
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":isbn",new AttributeValue {S = "444-4444444444"}}
    },
    ConditionExpression = "#I = :isbn"

};
var response = client.PutItem(request);
```

자세한 내용은 PutItem 단원을 참조하세요.

## 항목 가져오기

`GetItem` 메서드는 항목을 가져옵니다.

> **Note**
>
> 다수의 항목을 가져올 때는 `BatchGetItem` 메서드를 사용합니다. 자세한 내용은 일괄 가져오기: 여러 항목 가져오기 단원을 참조하십시오.

다음은 하위 수준 AWS SDK for .NET API를 사용하여 기존 항목을 가져오는 단계입니다.

1. `AmazonDynamoDBClient` 클래스의 인스턴스를 만듭니다.

2. `GetItemRequest` 클래스 인스턴스를 생성하여 필요한 파라미터를 입력합니다.

   항목을 가져오려면 테이블 이름과 항목의 기본 키를 입력해야 합니다.
3. 이전 단계에서 생성한 `GetItemRequest` 객체를 입력하여 `GetItem` 메서드를 실행합니다.

다음 C# 예제에서는 이전 단계를 설명합니다. 이 예제에서는 `ProductCatalog` 테이블에서 항목을 검색합니다.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
   Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
 };
 var response = client.GetItem(request);

// Check the response.
var result = response.GetItemResult;
var attributeMap = result.Item; // Attribute list in the response.
```

### 옵션 파라미터 지정

아래 C# 예제에서와 같이 `GetItemRequest` 객체를 사용하여 옵션 파라미터를 지정할 수도 있습니다. 이 샘플에서 지정하는 옵션 파라미터는 다음과 같습니다.

- `ProjectionExpression` 파라미터는 가져올 속성을 지정합니다.
- `ConsistentRead` 파라미터는 Strongly Consistent Read를 실행합니다. 읽기 일관성에 대한 자세한 내용은 DynamoDB 읽기 일관성 단원을 참조하십시오.

### Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
```

```
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
    // Optional parameters.
    ProjectionExpression = "Id, ISBN, Title, Authors",
    ConsistentRead = true
 };

 var response = client.GetItem(request);

// Check the response.
var result = response.GetItemResult;
var attributeMap = result.Item;
```

자세한 내용은 GetItem을 참조하세요.

## 항목 업데이트

UpdateItem 메서드는 기존 항목이 있는 경우 이를 업데이트합니다. UpdateItem 작업을 사용하면 기존 속성 값을 업데이트하거나, 새로운 속성을 추가하거나, 혹은 기존 속성 컬렉션에서 속성을 삭제할 수도 있습니다. 기본 키를 지정한 항목이 없는 경우에는 새로운 항목을 추가합니다.

UpdateItem 작업은 다음 지침을 사용합니다.

- 항목이 없는 경우 UpdateItem은 입력에서 지정한 기본 키를 사용하여 새 항목을 추가합니다.
- 항목이 있는 경우 UpdateItem은 다음과 같이 업데이트를 적용합니다.
  - 기존 속성 값을 업데이트 값으로 변경합니다.
  - 입력한 속성이 없는 경우에는 새로운 속성을 항목에 추가합니다.
  - 입력 속성이 null인 경우에는 입력한 속성이 있으면 삭제합니다.
  - Action에서 ADD를 사용하면 기존 집합(문자열 또는 숫자 집합)에 값을 추가하거나, 기존 숫자 속성 값에서 수치적으로 더하거나(양수 사용) 뺄 수 있습니다(음수 사용).

> **Note**
>
> PutItem 작업은 업데이트도 가능합니다. 자세한 내용은 항목 추가 단원을 참조하십시오. 예를 들어 PutItem을 호출하여 항목을 업데이트할 때 기본 키가 있을 경우 PutItem 작업이 전체 항목을 변경합니다. 기존 항목에 속성이 있으나 그러한 속성이 입력에 지정되지 않은 경우, PutItem 작업은 그러한 속성을 삭제합니다. 그러나 UpdateItem은 지정된 입력 속성만 업데이트합니다. 해당 항목의 다른 모든 기존 속성은 변경되지 않고 그대로 유지됩니다.

다음은 하위 수준 .NET SDK API를 사용하여 기존 항목을 업데이트하는 단계입니다.

1. AmazonDynamoDBClient 클래스의 인스턴스를 만듭니다.
2. UpdateItemRequest 클래스 인스턴스를 생성하여 필요한 파라미터를 입력합니다.

   이 인스턴스는 속성 추가, 기존 속성 업데이트 또는 속성 삭제 등과 같이 모든 업데이트를 설명하는 요청 객체입니다. 기존 속성을 삭제하려면 null 값으로 속성 이름을 지정합니다.
3. 이전 단계에서 생성한 UpdateItemRequest 객체를 입력하여 UpdateItem 메서드를 실행합니다.

다음 C# 코드 예제에서는 이전 단계를 설명합니다. 이 예제에서는 ProductCatalog 테이블의 book 항목을 업데이트합니다. 새로운 작성자를 Authors 컬렉션에 추가하고 기존 ISBN 속성을 삭제합니다. 또한 가격을 1만큼 내립니다.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#A", "Authors"},
        {"#P", "Price"},
        {"#NA", "NewAttribute"},
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":auth",new AttributeValue { SS = {"Author YY","Author ZZ"}}},
        {":p",new AttributeValue {N = "1"}},
        {":newattr",new AttributeValue {S = "someValue"}},
    },

    // This expression does the following:
    // 1) Adds two new authors to the list
    // 2) Reduces the price
    // 3) Adds a new attribute to the item
```

```
    // 4) Removes the ISBN attribute from the item
    UpdateExpression = "ADD #A :auth SET #P = #P - :p, #NA = :newattr REMOVE #I"
};
var response = client.UpdateItem(request);
```

옵션 파라미터 지정

아래 C# 예제에서와 같이 UpdateItemRequest 객체를 사용하여 옵션 파라미터를 지정할 수도 있습니다. 이 예제에서 지정하는 옵션 파라미터는 다음과 같습니다.

- ExpressionAttributeValues 및 ConditionExpression 파라미터는 기존 가격이 20.00인 경우에만 가격을 업데이트하도록 지정합니다.
- ReturnValues 파라미터는 응답 시 업데이트된 항목을 요청합니다.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },

    // Update price only if the current price is 20.00.
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#P", "Price"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":newprice",new AttributeValue {N = "22"}},
        {":currprice",new AttributeValue {N = "20"}}
    },
    UpdateExpression = "SET #P = :newprice",
    ConditionExpression = "#P = :currprice",
    TableName = tableName,
    ReturnValues = "ALL_NEW" // Return all the attributes of the updated item.
};

var response = client.UpdateItem(request);
```

자세한 내용은 UpdateItem 단원을 참조하세요.

## 원자성 카운터

updateItem을 사용하여 원자성 카운터를 구현할 수 있습니다. 이는 다른 쓰기 요청과 충돌하지 않고도 기존 속성의 값을 증감합니다. 원자성 카운터를 업데이트하려면 UpdateExpression 파라미터에서 Number 형식의 속성을 가진 updateItem을 사용하고, ADD에서는 Action을 사용합니다.

다음 예제는 Quantity 속성을 하나씩 늘리며 이를 보여 줍니다.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    Key = new Dictionary<string, AttributeValue>() { { "Id", new AttributeValue { N =
 "121" } } },
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#Q", "Quantity"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":incr",new AttributeValue {N = "1"}}
    },
    UpdateExpression = "SET #Q = #Q + :incr",
    TableName = tableName
};

var response = client.UpdateItem(request);
```

## 항목 삭제

DeleteItem 메서드는 테이블에서 항목을 삭제합니다.

다음은 하위 수준 .NET SDK API를 사용하여 항목을 삭제하는 단계입니다.

1. AmazonDynamoDBClient 클래스의 인스턴스를 만듭니다.
2. DeleteItemRequest 클래스 인스턴스를 생성하여 필요한 파라미터를 입력합니다.

   항목을 삭제하려면 테이블 이름과 항목의 기본 키가 필요합니다.

3. 이전 단계에서 생성한 DeleteItemRequest 객체를 입력하여 DeleteItem 메서드를 실행합니다.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },
};

var response = client.DeleteItem(request);
```

옵션 파라미터 지정

아래 C# 코드 예제에서와 같이 DeleteItemRequest 객체를 사용하여 옵션 파라미터를 지정할 수도 있습니다. 이 예제에서 지정하는 옵션 파라미터는 다음과 같습니다.

- ExpressionAttributeValues 및 ConditionExpression 파라미터는 서적이 더 이상 출판되지 않는 경우에만(InPublication 속성 값 false) 서적 항목을 삭제할 수 있도록 지정합니다.
- ReturnValues 파라미터는 응답 시 삭제된 항목을 요청합니다.

Example

```
var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },

    // Optional parameters.
    ReturnValues = "ALL_OLD",
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#IP", "InPublication"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
```

```
    {
        {":inpub",new AttributeValue {BOOL = false}}
    },
    ConditionExpression = "#IP = :inpub"
};

var response = client.DeleteItem(request);
```

자세한 내용은 DeleteItem을 참조하세요.

## 일괄 쓰기 - 여러 항목 추가 및 삭제

배치 쓰기란 다수의 항목을 배치로 입력 및 삭제하는 것을 말합니다. BatchWriteItem 메서드를 사용하면 단 한 번의 호출로 하나 이상의 테이블에 다수의 항목을 업로드하거나 삭제할 수 있습니다. 다음은 하위 수준 .NET SDK API를 사용하여 다수의 항목을 가져오는 단계입니다.

1. AmazonDynamoDBClient 클래스의 인스턴스를 만듭니다.
2. BatchWriteItemRequest 클래스 인스턴스를 생성하여 모든 업로드 및 삭제 작업을 설명합니다.
3. 이전 단계에서 생성한 BatchWriteItemRequest 객체를 입력하여 BatchWriteItem 메서드를 실행합니다.
4. 응답을 처리합니다. 응답과 함께 반환되는 요청 항목 중 처리하지 않은 항목이 있는지 확인해야 합니다. 프로비저닝된 처리량이 할당량에 이르거나 일시적 오류가 발생하면 이러한 경우가 발생할 수 있습니다. 또한 DynamoDB는 요청 시 지정하는 요청 크기나 작업 수를 제한합니다. 따라서 이러한 제한을 초과할 경우 DynamoDB가 요청을 거부하기도 합니다. 자세한 내용은 BatchWriteItem을 참조하세요.

다음 C# 코드 예제에서는 이전 단계를 설명합니다. 아래 예제는 BatchWriteItemRequest를 생성하여 다음 쓰기 작업을 실행합니다.

- Forum 테이블에 항목을 업로드합니다.
- Thread 테이블에서 항목을 업로드하고 삭제합니다.

그런 다음 코드가 BatchWriteItem을 실행하여 배치 작업을 시작합니다.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";
```

```
string table2Name = "Thread";

var request = new BatchWriteItemRequest
 {
   RequestItems = new Dictionary<string, List<WriteRequest>>
    {
      {
        table1Name, new List<WriteRequest>
        {
          new WriteRequest
          {
             PutRequest = new PutRequest
             {
                Item = new Dictionary<string,AttributeValue>
                {
                  { "Name", new AttributeValue { S = "Amazon S3 forum" } },
                  { "Threads", new AttributeValue { N = "0" }}
                }
             }
          }
        }
      } ,
      {
        table2Name, new List<WriteRequest>
        {
          new WriteRequest
          {
            PutRequest = new PutRequest
            {
               Item = new Dictionary<string,AttributeValue>
               {
                 { "ForumName", new AttributeValue { S = "Amazon S3 forum" } },
                 { "Subject", new AttributeValue { S = "My sample question" } },
                 { "Message", new AttributeValue { S = "Message Text." } },
                 { "KeywordTags", new AttributeValue { SS = new List<string> { "Amazon
 S3", "Bucket" }  } }
               }
            }
          },
          new WriteRequest
          {
             DeleteRequest = new DeleteRequest
             {
                Key = new Dictionary<string,AttributeValue>()
```

```
                {
                   { "ForumName", new AttributeValue { S = "Some forum name" } },
                   { "Subject", new AttributeValue { S = "Some subject" } }
                }
             }
          }
        }
      }
    }
 };
response = client.BatchWriteItem(request);
```

사용 가능한 예제는 [예: AWS SDK for .NET 하위 수준 API를 사용하는 일괄 작업](#)를 참조하세요.

## 일괄 가져오기: 여러 항목 가져오기

`BatchGetItem` 메서드를 사용하면 하나 이상의 테이블에서 여러 항목을 검색할 수 있습니다.

> **Note**
>
> 단일 항목을 검색하려면 `GetItem` 메서드를 사용합니다.

다음은 하위 수준 AWS SDK for .NET API를 사용하여 다수의 항목을 가져오는 단계입니다.

1. `AmazonDynamoDBClient` 클래스의 인스턴스를 만듭니다.
2. `BatchGetItemRequest` 클래스 인스턴스를 생성하여 필요한 파라미터를 입력합니다.

   다수의 항목을 가져오려면 테이블 이름과 기본 키 값의 목록이 필요합니다.
3. 이전 단계에서 생성한 `BatchGetItemRequest` 객체를 입력하여 `BatchGetItem` 메서드를 실행합니다.
4. 응답을 처리합니다. 처리하지 않은 키가 있는지 확인해야 합니다. 프로비저닝된 처리량이 할당량에 이르거나 일시적 오류가 발생하면 이러한 경우가 발생할 수 있습니다.

다음 C# 코드 예제에서는 이전 단계를 설명합니다. 아래 예제에서는 `Forum` 및 `Thread` 테이블에서 항목을 가져옵니다. 요청에 따라 지정하는 항목 수는 `Forum` 테이블에서 2개, 그리고 `Thread` 테이블에서 3개입니다. 따라서 이 요청에는 두 테이블의 항목이 모두 포함됩니다. 코드를 보면 응답의 처리 방식을 알 수 있습니다.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";
string table2Name = "Thread";

var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      }
    },
    {
      table2Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 1" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 2" } }
          },
          new Dictionary<string, AttributeValue>()
          {
```

```
            { "ForumName", new AttributeValue { S = "Amazon S3" } },
            { "Subject", new AttributeValue { S = "Amazon S3 Thread 1" } }
          }
        }
      }
    }
  }
};

var response = client.BatchGetItem(request);

// Check the response.
var result = response.BatchGetItemResult;
var responses = result.Responses; // The attribute list in the response.

var table1Results = responses[table1Name];
Console.WriteLine("Items in table {0}" + table1Name);
foreach (var item1 in table1Results.Items)
{
  PrintItem(item1);
}

var table2Results = responses[table2Name];
Console.WriteLine("Items in table {1}" + table2Name);
foreach (var item2 in table2Results.Items)
{
  PrintItem(item2);
}
// Any unprocessed keys? could happen if you exceed ProvisionedThroughput or some other
 error.
Dictionary<string, KeysAndAttributes> unprocessedKeys = result.UnprocessedKeys;
foreach (KeyValuePair<string, KeysAndAttributes> pair in unprocessedKeys)
{
    Console.WriteLine(pair.Key, pair.Value);
}
```

### 옵션 파라미터 지정

아래 C# 코드 예제에서와 같이 `BatchGetItemRequest` 객체를 사용하여 옵션 파라미터를 지정할 수도 있습니다. 이 예제에서는 `Forum` 테이블에서 두 개의 항목을 가져옵니다. 이 예제에서 지정하는 옵션 파라미터는 다음과 같습니다.

- `ProjectionExpression` 파라미터는 가져올 속성을 지정합니다.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";

var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      },
      // Optional - name of an attribute to retrieve.
      ProjectionExpression = "Title"
    }
  }
};

var response = client.BatchGetItem(request);
```

자세한 내용은 BatchGetItem을 참조하세요.

## 예: AWS SDK for .NET 하위 수준 API를 사용하는 CRUD 작업

다음 C# 코드 예제는 Amazon DynamoDB 항목에 대한 CRUD 작업을 보여줍니다. 여기 예제에서는 항목을 `ProductCatalog` 테이블에 추가하고, 항목을 가져오고, 다수의 업데이트를 실행하고, 그리고 마지막으로 항목을 삭제합니다. 이 테이블을 만들지 않은 경우 프로그래밍 방식으로 만들 수도 있습니다. 자세한 내용은 AWS SDK for .NET를 사용한 예시 테이블 생성 및 데이터 업로드 단원을 참조하십시오.

다음 샘플을 테스트하기 위한 단계별 지침은 .NET 코드 예시 섹션을 참조하세요.

### Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelItemCRUDExample
    {
        private static string tableName = "ProductCatalog";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                CreateItem();
                RetrieveItem();

                // Perform various updates.
                UpdateMultipleAttributes();
                UpdateExistingAttributeConditionally();

                // Delete item.
                DeleteItem();
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
        }

        private static void CreateItem()
        {
```

```
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  }},
                { "Title", new AttributeValue {
                      S = "Book 201 Title"
                  }},
                { "ISBN", new AttributeValue {
                      S = "11-11-11-11"
                  }},
                { "Authors", new AttributeValue {
                      SS = new List<string>{"Author1", "Author2" }
                  }},
                { "Price", new AttributeValue {
                      N = "20.00"
                  }},
                { "Dimensions", new AttributeValue {
                      S = "8.5x11.0x.75"
                  }},
                { "InPublication", new AttributeValue {
                      BOOL = false
                  } }
            }
            };
            client.PutItem(request);
        }

        private static void RetrieveItem()
        {
            var request = new GetItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                ProjectionExpression = "Id, ISBN, Title, Authors",
                ConsistentRead = true
```

```
            };
            var response = client.GetItem(request);

            // Check the response.
            var attributeList = response.Item; // attribute list in the response.
            Console.WriteLine("\nPrinting item after retrieving it ............");
            PrintItem(attributeList);
        }

        private static void UpdateMultipleAttributes()
        {
            var request = new UpdateItemRequest
            {
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                // Perform the following updates:
                // 1) Add two new authors to the list
                // 1) Set a new attribute
                // 2) Remove the ISBN attribute
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#A","Authors"},
                {"#NA","NewAttribute"},
                {"#I","ISBN"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":auth",new AttributeValue {
                     SS = {"Author YY", "Author ZZ"}
                 }},
                {":new",new AttributeValue {
                     S = "New Value"
                 }}
            },

                UpdateExpression = "ADD #A :auth SET #NA = :new REMOVE #I",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
```

```
            var response = client.UpdateItem(request);

            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
                                                     // print attributeList.
            Console.WriteLine("\nPrinting item after multiple attribute
update ............");
            PrintItem(attributeList);
        }

        private static void UpdateExistingAttributeConditionally()
        {
            var request = new UpdateItemRequest
            {
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#P", "Price"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":newprice",new AttributeValue {
                     N = "22.00"
                 }},
                {":currprice",new AttributeValue {
                     N = "20.00"
                 }}
            },
                // This updates price only if current price is 20.00.
                UpdateExpression = "SET #P = :newprice",
                ConditionExpression = "#P = :currprice",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
            var response = client.UpdateItem(request);

            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
```

```
            Console.WriteLine("\nPrinting item after updating price value
conditionally .....");
            PrintItem(attributeList);
        }

        private static void DeleteItem()
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },

                // Return the entire item as it appeared before the update.
                ReturnValues = "ALL_OLD",
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#IP", "InPublication"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":inpub",new AttributeValue {
                     BOOL = false
                 }}
            },
                ConditionExpression = "#IP = :inpub"
            };

            var response = client.DeleteItem(request);

            // Check the response.
            var attributeList = response.Attributes; // Attribute list in the response.
                                                     // Print item.
            Console.WriteLine("\nPrinting item that was just deleted .....");
            PrintItem(attributeList);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
```

```
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
    }
}
```

## 예: AWS SDK for .NET 하위 수준 API를 사용하는 일괄 작업

주제

- 예: AWS SDK for .NET 하위 수준 API를 사용하는 일괄 쓰기 작업
- 예: AWS SDK for .NET 하위 수준 API를 사용하는 일괄 가져오기 작업

이번 단원에서는 배치 쓰기, 배치 가져오기 같이 Amazon DynamoDB에서 지원되는 배치 작업의 예제를 제공합니다.

예: AWS SDK for .NET 하위 수준 API를 사용하는 일괄 쓰기 작업

다음 C# 코드는 `BatchWriteItem` 메서드를 사용해 업로드 및 삭제 작업을 실행하는 예제입니다.

- 한 항목을 `Forum` 테이블에 업로드합니다.
- 한 항목을 `Thread` 테이블에서 업로드하고 삭제합니다.

일괄 쓰기 요청을 생성할 때는 하나 이상의 테이블에 대해 업로드 및 삭제 요청을 얼마든지 지정할 수 있습니다. 하지만 DynamoDB `BatchWriteItem`은 단일 배치 쓰기 작업에서 배치 쓰기 요청의 크기와 업로드 및 삭제 작업의 수를 제한합니다. 자세한 내용은 BatchWriteItem을 참조하세요. 이러한 제한을 초과할 경우에는 요청이 거부됩니다. 이러한 요청을 처리하는 데 충분한 처리량이 테이블에 할당되어 있지 않은 경우에는 응답 시 처리되지 않은 요청 항목이 반환됩니다.

다음은 응답을 확인하여 처리되지 않은 요청 항목의 유무를 점검하는 예제입니다. 미처리 요청 항목이 있는 경우에는 루프백이 발생하여 미처리 항목이 있는 BatchWriteItem 요청을 다시 보냅니다. 이러한 샘플 테이블은 프로그래밍 방식으로 생성하여 업로드할 수도 있습니다. 자세한 내용은 AWS SDK for .NET를 사용한 예시 테이블 생성 및 데이터 업로드 단원을 참조하십시오.

다음 샘플을 테스트하기 위한 단계별 지침은 .NET 코드 예시 섹션을 참조하세요.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelBatchWrite
    {
        private static string table1Name = "Forum";
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                TestBatchWrite();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void TestBatchWrite()
        {
            var request = new BatchWriteItemRequest
            {
                ReturnConsumedCapacity = "TOTAL",
                RequestItems = new Dictionary<string, List<WriteRequest>>
            {
```

```
                {
                    table1Name, new List<WriteRequest>
                    {
                        new WriteRequest
                        {
                            PutRequest = new PutRequest
                            {
                                Item = new Dictionary<string, AttributeValue>
                                {
                                    { "Name", new AttributeValue {
                                          S = "S3 forum"
                                      } },
                                    { "Threads", new AttributeValue {
                                          N = "0"
                                      }}
                                }
                            }
                        }
                    }
                },
                {
                    table2Name, new List<WriteRequest>
                    {
                        new WriteRequest
                        {
                            PutRequest = new PutRequest
                            {
                                Item = new Dictionary<string, AttributeValue>
                                {
                                    { "ForumName", new AttributeValue {
                                          S = "S3 forum"
                                      } },
                                    { "Subject", new AttributeValue {
                                          S = "My sample question"
                                      } },
                                    { "Message", new AttributeValue {
                                          S = "Message Text."
                                      } },
                                    { "KeywordTags", new AttributeValue {
                                          SS = new List<string> { "S3", "Bucket" }
                                      } }
                                }
                            }
                        },
```

```
                        new WriteRequest
                        {
                            // For the operation to delete an item, if you provide a
primary key value
                            // that does not exist in the table, there is no error, it
is just a no-op.
                            DeleteRequest = new DeleteRequest
                            {
                                Key = new Dictionary<string, AttributeValue>()
                                {
                                    { "ForumName",  new AttributeValue {
                                          S = "Some partition key value"
                                      } },
                                    { "Subject", new AttributeValue {
                                          S = "Some sort key value"
                                      } }
                                }
                            }
                        }
                    }
                }
            }
            };

            CallBatchWriteTillCompletion(request);
        }

        private static void CallBatchWriteTillCompletion(BatchWriteItemRequest request)
        {
            BatchWriteItemResponse response;

            int callCount = 0;
            do
            {
                Console.WriteLine("Making request");
                response = client.BatchWriteItem(request);
                callCount++;

                // Check the response.

                var tableConsumedCapacities = response.ConsumedCapacity;
                var unprocessed = response.UnprocessedItems;

                Console.WriteLine("Per-table consumed capacity");
```

```
                foreach (var tableConsumedCapacity in tableConsumedCapacities)
                {
                    Console.WriteLine("{0} - {1}", tableConsumedCapacity.TableName,
 tableConsumedCapacity.CapacityUnits);
                }

                Console.WriteLine("Unprocessed");
                foreach (var unp in unprocessed)
                {
                    Console.WriteLine("{0} - {1}", unp.Key, unp.Value.Count);
                }
                Console.WriteLine();

                // For the next iteration, the request will have unprocessed items.
                request.RequestItems = unprocessed;
            } while (response.UnprocessedItems.Count > 0);

            Console.WriteLine("Total # of batch write API calls made: {0}", callCount);
        }
    }
}
```

예: AWS SDK for .NET 하위 수준 API를 사용하는 일괄 가져오기 작업

다음 C# 코드 예제에서는 `BatchGetItem` 메서드를 사용해 Amazon DynamoDB의 `Forum` 및 `Thread` 테이블에서 다수의 항목을 가져옵니다. `BatchGetItemRequest`는 테이블 이름을 비롯해 각 테이블의 기본 키 목록을 지정합니다. 이 예제는 가져온 항목을 출력하여 응답을 처리합니다.

다음 샘플을 테스트하기 위한 단계별 지침은 .NET 코드 예시 섹션을 참조하세요.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelBatchGet
    {
        private static string table1Name = "Forum";
```

```
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                RetrieveMultipleItemsBatchGet();

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void RetrieveMultipleItemsBatchGet()
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
            {
                { table1Name,
                  new KeysAndAttributes
                  {
                      Keys = new List<Dictionary<string, AttributeValue> >()
                      {
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                                S = "Amazon DynamoDB"
                            } }
                          },
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                                S = "Amazon S3"
                            } }
                          }
                      }
                }},
                {
                    table2Name,
                    new KeysAndAttributes
```

```
                {
                    Keys = new List<Dictionary<string, AttributeValue> >()
                    {
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon DynamoDB"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "DynamoDB Thread 1"
                              } }
                        },
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon DynamoDB"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "DynamoDB Thread 2"
                              } }
                        },
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon S3"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "S3 Thread 1"
                              } }
                        }
                    }
                }
            }
        }
        };

        BatchGetItemResponse response;
        do
        {
            Console.WriteLine("Making request");
            response = client.BatchGetItem(request);

            // Check the response.
            var responses = response.Responses; // Attribute list in the response.
```

```
                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
                }

                // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +
```

```
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("*************************************************");
        }
    }
}
```

## 예: AWS SDK for .NET 하위 수준 API를 사용하여 이진 형식 속성 처리

다음 C# 코드는 이진수 형식 속성의 처리 방법을 나타낸 예제입니다. 이 예제에서는 항목을 Reply 테이블에 추가합니다. 추가된 항목에는 압축 데이터가 저장된 이진수 형식의 속성(ExtendedMessage)이 포함되어 있습니다. 그런 다음 항목을 가져와서 모든 속성 값을 출력합니다. 이해를 돕기 위해 예제에서는 GZipStream 클래스를 사용해 샘플 스트림을 압축하여 ExtendedMessage 속성에 할당한 다음 이후 속성 값을 출력할 때 압축을 풉니다.

다음 예제를 테스트하기 위한 단계별 지침은 .NET 코드 예시 단원을 참조하세요.

### Example

```
using System;
using System.Collections.Generic;
using System.IO;
using System.IO.Compression;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelItemBinaryExample
    {
        private static string tableName = "Reply";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            // Reply table primary key.
            string replyIdPartitionKey = "Amazon DynamoDB#DynamoDB Thread 1";
            string replyDateTimeSortKey = Convert.ToString(DateTime.UtcNow);

            try
```

```
            {
                CreateItem(replyIdPartitionKey, replyDateTimeSortKey);
                RetrieveItem(replyIdPartitionKey, replyDateTimeSortKey);
                // Delete item.
                DeleteItem(replyIdPartitionKey, replyDateTimeSortKey);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void CreateItem(string partitionKey, string sortKey)
        {
            MemoryStream compressedMessage = ToGzipMemoryStream("Some long extended
message to compress.");
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  }},
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  }},
                { "Subject", new AttributeValue {
                      S = "Binary type "
                  }},
                { "Message", new AttributeValue {
                      S = "Some message about the binary type"
                  }},
                { "ExtendedMessage", new AttributeValue {
                      B = compressedMessage
                  }}
            }
            };
            client.PutItem(request);
        }

        private static void RetrieveItem(string partitionKey, string sortKey)
        {
```

```
            var request = new GetItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  } },
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  } }
            },
                ConsistentRead = true
            };
            var response = client.GetItem(request);

            // Check the response.
            var attributeList = response.Item; // attribute list in the response.
            Console.WriteLine("\nPrinting item after retrieving it ............");

            PrintItem(attributeList);
        }

        private static void DeleteItem(string partitionKey, string sortKey)
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  } },
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  } }
            }
            };
            var response = client.DeleteItem(request);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
```

```
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]") +
                    (value.B == null ? "" : "B=[" + FromGzipMemoryStream(value.B) +
 "]")
                    );
            }
            Console.WriteLine("************************************************");
        }

        private static MemoryStream ToGzipMemoryStream(string value)
        {
            MemoryStream output = new MemoryStream();
            using (GZipStream zipStream = new GZipStream(output,
 CompressionMode.Compress, true))
            using (StreamWriter writer = new StreamWriter(zipStream))
            {
                writer.Write(value);
            }
            return output;
        }

        private static string FromGzipMemoryStream(MemoryStream stream)
        {
            using (GZipStream zipStream = new GZipStream(stream,
 CompressionMode.Decompress))
            using (StreamReader reader = new StreamReader(zipStream))
            {
                return reader.ReadToEnd();
            }
        }
    }
}
```

# DynamoDB에서 보조 인덱스를 사용하여 데이터 액세스 개선

Amazon DynamoDB는 기본 키 값을 지정하여 테이블의 항목에 신속하게 액세스할 수 있습니다. 그러나 많은 애플리케이션에서는 주요 키가 아닌 속성을 가진 데이터에 효율적으로 액세스할 수 있다는 장점을 활용하기 위해 하나 이상의 보조(또는 대체) 키를 사용하기도 합니다. 이를 위해 테이블에서 하나 이상의 보조 인덱스를 생성하고 이 인덱스에 대해 `Query` 또는 `Scan` 요청을 발행할 수 있습니다.

보조 인덱스는 `Query` 작업을 지원하는 대체 키와 함께 테이블의 속성 하위 집합을 포함하는 데이터 구조입니다. 테이블에서 `Query`를 사용하는 것과 거의 같은 방식으로 `Query`를 사용하여 인덱스에서 데이터를 가져올 수 있습니다. 한 테이블에 여러 개의 보조 인덱스를 포함할 수 있어서 애플리케이션이 서로 다른 여러 쿼리 패턴에 액세스할 수 있습니다.

> **Note**
>
> 또한 테이블을 `Scan`하는 것과 거의 동일한 방식으로 인덱스를 `Scan`할 수 있습니다.
> 보조 인덱스 스캔 작업에 대한 교차 계정 액세스는 현재 리소스 기반 정책에서 지원되지 않습니다.

모든 보조 인덱스는 정확히 하나의 테이블과 연결되어 해당 테이블에서 데이터를 가져옵니다. 이를 인덱스의 기본 테이블이라고 합니다. 인덱스를 생성할 때 인덱스 파티션 및 정렬 키로 대체 키를 정의할 수 있습니다. 또한 기본 테이블에서 인덱스로 프로젝션하거나 복사하려는 속성을 정의합니다. DynamoDB는 이 속성을 기본 테이블의 기본 키 속성과 함께 인덱스로 복사합니다. 그러면 테이블을 쿼리하거나 스캔하는 것처럼 인덱스를 쿼리하거나 스캔할 수 있습니다.

모든 보조 인덱스는 DynamoDB에서 자동으로 유지 관리됩니다. 기본 테이블의 항목을 추가, 수정 또는 삭제할 때 해당 테이블의 모든 인덱스도 업데이트되어 이 변경 사항이 반영됩니다.

DynamoDB는 두 종류의 보조 인덱스를 지원합니다.

- 글로벌 보조 인덱스 - 파티션 키와 정렬 키와 기본 테이블과 다를 수 있는 인덱스입니다. 모든 파티션에서 인덱스의 쿼리가 기본 테이블의 모든 데이터에 적용될 수 있으므로 글로벌 보조 인덱스는 글로벌하게 간주됩니다. 글로벌 보조 인덱스는 기본 테이블과 별개로 자체 파티션 공간에 저장되며 기본 테이블과는 별도로 크기 조정됩니다.
- 로컬 보조 인덱스 - 기본 테이블과 파티션 키는 동일하지만 정렬 키가 다른 인덱스입니다. 로컬 보조 인덱스는 로컬 보조 인덱스의 모든 파티션이 동일한 파티션 키 값을 갖는 기본 테이블 파티션으로 한정된다는 의미에서 "로컬"이라고 합니다.

글로벌 보조 인덱스와 로컬 보조 인덱스에 대한 비교는 이 비디오를 참조하세요.

GSI와 LSI 사이에서 올바른 선택하기

주제

- DynamoDB에서 글로벌 보조 인덱스 사용
- DynamoDB의 로컬 보조 인덱스

사용할 인덱스 유형을 결정할 때 애플리케이션의 요구 사항을 고려해야 합니다. 다음 표에서는 글로벌 보조 인덱스와 로컬 보조 인덱스 간의 주된 차이점을 보여 줍니다.

| 기능 | 글로벌 보조 인덱스 | 로컬 보조 인덱스 |
| --- | --- | --- |
| 키 스키마 | 글로벌 보조 인덱스의 기본 키는 단순 기본 키(파티션 키)이거나 복합 기본 키(파티션 키 및 정렬 키)일 수 있습니다. | 로컬 보조 인덱스의 기본 키는 반드시 복합 기본 키(파티션 키 및 정렬 키)여야 합니다. |
| 키 속성 | 인덱스 파티션 키 및 정렬 키(있을 경우)는 문자열, 숫자 또는 이진수 형식의 기본 테이블 속성일 수 있습니다. | 인덱스의 파티션 키는 기본 테이블의 파티션 키와 동일한 속성입니다. 정렬 키는 문자열, 숫자 또는 이진수 형식의 기본 테이블 속성일 수 있습니다. |
| 파티션 키 값당 크기 제한 | 글로벌 보조 인덱스에는 크기 제한이 없습니다. | 파티션 키 값마다 인덱싱된 모든 항목의 전체 크기가 10GB 이하여야 합니다. |
| 온라인 인덱스 작업 | 글로벌 보조 인덱스는 테이블을 생성할 때 동시에 생성될 수 있습니다. 새 글로벌 보조 인덱스를 기존 테이블에 추가하거나 기존 글로벌 보조 인덱스를 삭제할 수도 있습니다. 자세한 내용은 DynamoDB에서 글로벌 보조 인덱스 관리 섹션을 참조하세요. | 로컬 보조 인덱스는 테이블을 생성할 때 동시에 생성됩니다. 기존 테이블에 로컬 보조 인덱스를 추가할 수도 없고 현재 있는 로컬 보조 인덱스를 삭제할 수도 없습니다. |

| 기능 | 글로벌 보조 인덱스 | 로컬 보조 인덱스 |
| --- | --- | --- |
| 쿼리 및 파티션 | 글로벌 보조 인덱스를 사용하면 모든 파티션에서 전체 테이블을 쿼리할 수 있습니다. | 로컬 보조 인덱스를 사용하면 쿼리에서 파티션 키 값으로 지정하는 단일 파티션을 쿼리할 수 있습니다. |
| 읽기 일관성 | 글로벌 보조 인덱스의 쿼리는 최종 일관성만 지원합니다. | 로컬 보조 인덱스를 쿼리할 때는 최종 일관성 또는 강력한 일관성을 선택할 수 있습니다. |
| 프로비저닝된 처리량 소비 | 글로벌 보조 인덱스마다 읽기 및 쓰기 활동에 대한 고유한 프로비저닝된 처리량 설정이 있습니다. 글로벌 보조 인덱스의 쿼리 및 스캔은 기본 테이블이 아니라 인덱스의 용량 단위를 소비합니다. 테이블 쓰기로 인한 글로벌 보조 인덱스 업데이트의 경우도 마찬가지입니다. 글로벌 테이블과 연결된 글로벌 보조 인덱스는 쓰기 용량 단위를 소비합니다. | 로컬 보조 인덱스의 쿼리 또는 스캔은 기본 테이블의 읽기 용량 단위를 소비합니다. 테이블에 쓸 때 해당 로컬 보조 인덱스도 업데이트됩니다. 이 업데이트는 기본 테이블의 쓰기 용량 단위를 소비합니다. 글로벌 테이블과 연결된 로컬 보조 인덱스는 복제된 쓰기 용량 단위를 소비합니다. |
| 프로젝션 속성 | 글로벌 보조 인덱스 쿼리 또는 스캔에서는 인덱스로 프로젝션되는 속성만 요청할 수 있습니다. DynamoDB가 테이블에서는 어떤 속성도 가져오지 않습니다. | 로컬 보조 인덱스를 쿼리 또는 스캔하는 경우에는 인덱스로 프로젝션되지 않는 속성만 요청할 수 있습니다. DynamoDB가 테이블에서 이러한 속성을 자동으로 가져옵니다. |

보조 인덱스가 있는 테이블을 2개 이상 생성하려면 순차적으로 해야 합니다. 예를 들어 첫 번째 테이블을 만들고 테이블이 `ACTIVE`가 될 때까지 기다리고, 다음 테이블을 만들고 이 테이블이 `ACTIVE`가 될 때까지 기다리는 방식으로 진행됩니다. 보조 인덱스가 있는 테이블을 2개 이상 동시에 생성하면 DynamoDB에서 `LimitExceededException`을 반환합니다.

각 보조 인덱스는 연결된 기본 테이블과 동일한 테이블 클래스 및 용량 모드를 사용합니다. 각 보조 인덱스에 대해 다음을 지정해야 합니다.

- 생성할 인덱스의 유형, 즉 글로벌 보조 인덱스 또는 로컬 보조 인덱스.
- 인덱스의 이름. 인덱스의 이름 지정 규칙은 Amazon DynamoDB의 할당량에 나온 대로 테이블의 경우와 동일합니다. 이름은 연관된 기본 테이블에 대해 고유해야 하지만 다른 기본 테이블과 연관된 인덱스에 동일한 이름을 사용할 수 있습니다.
- 인덱스의 키 스키마. 인덱스 키 스키마의 모든 속성은 `String`, `Number` 또는 `Binary` 형식의 최상위 속성이어야 합니다. 문서, 집합 등 다른 데이터 형식은 허용되지 않습니다. 키 스키마의 다른 요구 사항은 인덱스 유형에 따라 결정됩니다.
  - 글로벌 보조 인덱스의 경우 파티션 키는 기본 테이블의 스칼라 속성일 수 있습니다. 정렬 키는 선택적이며 역시 기본 테이블의 스칼라 속성일 수 있습니다.
  - 로컬 보조 인덱스의 경우 파티션 키는 기본 테이블의 파티션 키와 동일해야 하며 정렬 키는 키가 아닌 기본 테이블 속성이어야 합니다.
- 기본 테이블에서 인덱스로 프로젝션할 추가 속성(있는 경우). 테이블 키 속성 외에 이 속성도 모든 인덱스에 자동으로 프로젝션됩니다. 스칼라, 문서, 집합을 포함하여 모든 데이터 형식의 속성을 프로젝션할 수 있습니다.
- 필요할 경우 인덱스의 프로비저닝 처리량 설정:
  - 글로벌 보조 인덱스의 경우 읽기 및 쓰기 용량 단위 설정을 지정해야 합니다. 이 할당 처리량 설정은 기본 테이블의 설정과 관련이 없습니다.
  - 로컬 보조 인덱스의 경우 읽기 및 쓰기 용량 단위 설정을 지정할 필요가 없습니다. 로컬 보조 인덱스의 읽기 및 쓰기 작업은 기본 테이블의 프로비저닝된 처리량 설정을 활용합니다.

쿼리 유연성을 극대화하도록 테이블당 최대 20개의 글로벌 보조 인덱스(기본 할당량)와 최대 5개의 로컬 보조 인덱스를 생성할 수 있습니다.

테이블의 보조 인덱스에 대한 상세 목록을 가져오려면 `DescribeTable` 작업을 사용합니다. `DescribeTable`은 테이블에 있는 모든 보조 인덱스의 이름, 스토리지 크기 및 항목 수를 반환합니다. 이 값은 실시간으로 업데이트되지 않지만 거의 6시간마다 새로 고쳐집니다.

`Query` 또는 `Scan` 작업을 사용하여 보조 인덱스의 데이터에 액세스할 수 있습니다. 사용하려는 인덱스의 이름과 기본 테이블의 이름, 결과에 반환할 속성, 적용할 조건 표현식 또는 필터를 지정해야 합니다. DynamoDB는 결과를 오름차순 또는 내림차순으로 반환할 수 있습니다.

테이블을 삭제할 때 테이블과 연관된 모든 인덱스도 삭제됩니다.

모범 사례는 DynamoDB의 보조 인덱스 사용에 대한 모범 사례 섹션을 참조하세요.

# DynamoDB에서 글로벌 보조 인덱스 사용

일부 애플리케이션은 서로 다른 속성을 쿼리 기준으로 사용하여 다양한 유형의 쿼리를 수행해야 할 수 있습니다. 이러한 요구 사항을 지원하기 위해 하나 이상의 글로벌 보조 인덱스를 생성하고 Amazon DynamoDB에서 이 인덱스에 대해 `Query` 요청을 실행할 수 있습니다.

주제


- 시나리오: 글로벌 보조 인덱스 사용
- 속성 프로젝션
- 글로벌 보조 인덱스에서 데이터 읽기
- 테이블과 글로벌 보조 인덱스 간 데이터 동기화
- 글로벌 보조 인덱스를 사용하는 테이블 클래스
- 글로벌 보조 인덱스에 대한 프로비저닝된 처리량 고려 사항
- 글로벌 보조 인덱스에 대한 스토리지 고려 사항
- DynamoDB에서 글로벌 보조 인덱스 관리
- DynamoDB에서 인덱스 키 위반 검색 및 수정
- 글로벌 보조 인덱스로 작업: Java
- 글로벌 보조 인덱스로 작업: .NET
- AWS CLI를 사용하여 DynamoDB에서 글로벌 보조 인덱스 작업


## 시나리오: 글로벌 보조 인덱스 사용

모바일 게임 애플리케이션에서 사용자와 점수를 추적하는 `GameScores`라는 테이블을 예로 들어 설명해 보겠습니다. `GameScores`의 각 항목은 파티션 키(`UserId`)와 정렬 키(`GameTitle`)로 식별됩니다. 다음 그림은 테이블에서 항목을 구성하는 방식을 보여 줍니다. (일부 속성만 표시)

GameScores

| UserId | GameTitle | TopScore | TopScoreDateTime | Wins | Losses | |
|---|---|---|---|---|---|---|
| "101" | "Galaxy Invaders" | 5842 | "2015-09-15:17:24:31" | 21 | 72 | ... |
| "101" | "Meteor Blasters" | 1000 | "2015-10-22:23:18:01" | 12 | 3 | ... |
| "101" | "Starship X" | 24 | "2015-08-31:13:14:21" | 4 | 9 | ... |
| "102" | "Alien Adventure" | 192 | "2015-07-12:11:07:56" | 32 | 192 | ... |
| "102" | "Galaxy Invaders" | 0 | "2015-09-18:07:33:42" | 0 | 5 | ... |
| "103" | "Attack Ships" | 3 | "2015-10-19:01:13:24" | 1 | 8 | ... |
| "103" | "Galaxy Invaders" | 2317 | "2015-09-11:06:53:00" | 40 | 3 | ... |
| "103" | "Meteor Blasters" | 723 | "2015-10-19:01:13:24" | 22 | 12 | ... |
| "103" | "Starship X" | 42 | "2015-07-11:06:53:00" | 4 | 19 | ... |
| ... | ... | ... | ... | ... | ... | |

이제 각 게임의 최고 점수를 표시하는 순위표 애플리케이션을 작성하려는 경우를 가정하겠습니다. 키 속성(`UserId` 및 `GameTitle`)을 지정한 쿼리는 매우 효율적일 것입니다. 그러나 애플리케이션에서 `GameTitle`만을 토대로 `GameScores`에서 데이터를 검색해야 할 경우에는 `Scan` 작업을 사용해야 할 수 있습니다. 테이블에 항목을 추가할수록 전체 데이터 스캔이 느려지고 효율성이 저하되면서 다음과 같은 질문에 응답하기가 어려워집니다.

- Meteor Blasters 게임의 최고 기록 점수
- Galaxy Invaders의 최고 점수 기록 보유자
- 최고 승률

키가 아닌 속성에 대한 쿼리 속도를 높이기 위해 글로벌 보조 인덱스를 만들 수 있습니다. 글로벌 보조 인덱스는 기본 테이블의 속성 중 일부를 포함하지만 테이블과 다른 기본 키를 기준으로 구성됩니다. 인덱스 키는 테이블의 키 속성을 가져야 할 필요가 없습니다. 테이블과 동일한 키 스키마를 가져야 할 필요도 없습니다.

예를 들어 파티션 키가 GameTitle이고 정렬 키가 TopScore인 GameTitleIndex라는 글로벌 보조 인덱스를 생성할 수 있습니다. 기본 테이블의 기본 키 속성이 언제나 인덱스로 프로젝션되므로 UserId 속성이 함께 표시됩니다. 다음 그림은 GameTitleIndex 인덱스를 나타냅니다.

*GameTitleIndex*

| GameTitle | TopScore | UserId |
|---|---|---|
| "Alien Adventure" | 192 | "102" |
| "Attack Ships" | 3 | "103" |
| "Galaxy Invaders" | 0 | "102" |
| "Galaxy Invaders" | 2317 | "103" |
| "Galaxy Invaders" | 5842 | "101" |
| "Meteor Blasters" | 723 | "103" |
| "Meteor Blasters" | 1000 | "101" |
| "Starship X" | 24 | "101" |
| "Starship X" | 42 | "103" |
| ... | ... | ... |

이제 GameTitleIndex를 쿼리하여 Meteor Blasters의 점수를 쉽게 가져올 수 있습니다. 결과는 정렬 키 값 TopScore로 정렬됩니다. ScanIndexForward 파라미터를 false로 설정하는 경우 결과가 내림차순으로 반환되어 가장 높은 점수가 가장 먼저 반환됩니다.

모든 글로벌 보조 인덱스에는 파티션 키가 있어야 하며 선택 사항으로 정렬 키가 있을 수 있습니다. 인덱스 키 스키마는 기본 테이블 스키마와 다를 수 있습니다. 단순 기본 키(파티션 키)를 사용하는 테이블이 있을 때 복합 기본 키(파티션 키 및 정렬 키)를 사용하여 글로벌 보조 인덱스를 만들거나 그 반대도 가능합니다. 인덱스 키 속성은 기본 테이블에서 나온 모든 최상위 수준 String, Number 또는 Binary 속성으로 구성될 수 있습니다. 다른 스칼라 유형, 문서 유형 및 집합 유형은 허용되지 않습니다.

다른 기본 테이블 속성을 인덱스로 프로젝션할 수도 있습니다. 인덱스를 쿼리할 경우 DynamoDB에서 이와 같이 프로젝션된 속성을 효율적으로 가져올 수 있습니다. 그러나 글로벌 보조 인덱스 쿼리는 기본 테이블에서 속성을 가져올 수 없습니다. 예를 들어 이전 다이어그램에 나와 있듯이

GameTitleIndex을 쿼리하는 경우, TopScore 이외의 어떤 키가 아닌 속성에도 액세스할 수 없습니다(키 속성 GameTitle와 UserId은 자동으로 프로젝션됨에도 불구하고).

DynamoDB 테이블에서 각 키 값은 고유해야 합니다. 하지만 글로벌 보조 인덱스의 키 값은 고유할 필요가 없습니다. 예를 들어 많은 새로운 사용자들이 도전하지만 0점을 넘지 못하는 특히 어려운 Comet Quest라는 게임이 있다고 가정해 보겠습니다. 아래에는 이를 표현할 수 있는 몇 가지 데이터가 나와 있습니다.

| UserId | GameTitle | TopScore |
|---|---|---|
| 123 | Comet Quest | 0 |
| 201 | Comet Quest | 0 |
| 301 | Comet Quest | 0 |

이 데이터가 GameScores 테이블에 추가되면 DynamoDB가 이를 GameTitleIndex에 전파합니다. GameTitle에 Comet Quest를 사용하고 TopScore에 0을 사용하여 인덱스를 쿼리하면 다음 데이터가 반환됩니다.

| *GameTitle* | *TopScore* | *UserId* |
|---|---|---|
| *"Comet Quest"* | *0* | *"123"* |
| *"Comet Quest"* | *0* | *"201"* |
| *"Comet Quest"* | *0* | *"301"* |

지정된 키 값을 가진 항목만 응답에 나타납니다. 이러한 데이터 집합 내에서 항목들은 특정한 순서를 따르지 않습니다.

글로벌 보조 인덱스는 해당 키 속성이 실제로 존재하는 데이터 항목만 추적합니다. 예를 들어 GameScores 테이블에 새 항목을 하나 더 추가하되 필요한 기본 키 속성만 제공한 경우를 가정하겠습니다.

| UserId | GameTitle |
|---|---|
| 400 | Comet Quest |

TopScore 속성을 지정하지 않았으므로 DynamoDB가 이 항목을 GameTitleIndex에 전파하지 않습니다. 라서, 모든 Comet Quest 항목에 대해 GameScores를 쿼리하면 다음 4개 항목이 반환됩니다.

| UserId | GameTitle | TopScore |
|---|---|---|
| "123" | "Comet Quest" | 0 |
| "201" | "Comet Quest" | 0 |
| "301" | "Comet Quest" | 0 |
| "400" | "Comet Quest" | |

그러나 GameTitleIndex로 유사한 쿼리를 실행할 경우 4개가 아닌 3개 항목이 반환됩니다. 그 이유는 존재하지 않는 TopScore를 보유한 항목은 인덱스로 적용되지 않기 때문입니다.

| *GameTitle* | *TopScore* | *UserId* |
|---|---|---|
| *"Comet Quest"* | *0* | *"123"* |
| *"Comet Quest"* | *0* | *"201"* |
| *"Comet Quest"* | *0* | *"301"* |

## 속성 프로젝션

프로젝션은 테이블에서 보조 인덱스로 복사되는 속성 집합입니다. 테이블의 파티션 키와 정렬 키는 항상 인덱스로 프로젝션되지만, 다른 속성을 프로젝션하여 애플리케이션의 쿼리 요건을 지원하는 것도 가능합니다. 따라서 인덱스에 쿼리를 실행할 때는 마치 속성이 자체 테이블에 저장되어 있는 것처럼 Amazon DynamoDB가 프로젝션의 모든 속성에 액세스할 수 있습니다.

보조 인덱스를 생성할 때 인덱스에 프로젝션될 속성을 지정해야 합니다. DynamoDB는 이를 위해 다음과 같은 세 가지 옵션을 제공합니다.

- KEYS_ONLY - 인덱스의 각 항목은 테이블 파티션 키 및 정렬 키 값, 그리고 인덱스 키 값으로만 구성됩니다. KEYS_ONLY 옵션은 보조 인덱스의 크기를 최소화합니다.
- INCLUDE – KEYS_ONLY에서 설명한 속성 외에도 지정하는 키가 아닌 다른 속성이 보조 인덱스에 포함됩니다.
- ALL – 보조 인덱스에 소스 테이블의 모든 속성이 포함됩니다. 모든 테이블 데이터가 인덱스에 복제되므로 ALL 프로젝션은 보조 인덱스의 크기를 최대화합니다.

이전 다이어그램에서 GameTitleIndex에는 프로젝션된 하나의 속성 UserId만 포함되어 있습니다. 애플리케이션이 쿼리의 GameTitle와 TopScore를 사용하여 각 게임의 최고 득점자의 UserId를 효과적으로 결정할 수 있지만 최고 득점자의 승패 격차의 최대 비율을 효율적으로 확인하지는 못합니다. 확인하려면 기본 테이블에서 추가 쿼리를 실행하여 각 최고 득점자의 승패를 가져와야 합니다. 이 데이터에서 쿼리를 지원하는 더욱 효율적인 방법은 다음 그림과 같은 속성을 기본 테이블에서 글로벌 보조 인덱스로 프로젝션하는 것입니다.

*GameTitleIndex*

| GameTitle | TopScore | UserId | Wins | Losses |
|---|---|---|---|---|
| "Alien Adventure" | 192 | "102" | 32 | 192 |
| "Attack Ships" | 3 | "103" | 1 | 8 |
| "Galaxy Invaders" | 0 | "102" | 0 | 5 |
| "Galaxy Invaders" | 2317 | "103" | 40 | 3 |
| "Galaxy Invaders" | 5842 | "101" | 21 | 72 |
| "Meteor Blasters" | 723 | "103" | 22 | 12 |
| "Meteor Blasters" | 1000 | "101" | 12 | 3 |
| "Starship X" | 24 | "101" | 4 | 9 |
| "Starship X" | 42 | "103" | 4 | 19 |
| ... | ... | ... | ... | ... |

키가 아닌 속성 Wins과 Losses는 인덱스로 프로젝션되기 때문에 애플리케이션으로 모든 게임 또는 모든 게임과 사용자 ID 조합에 대해 승률을 확인할 수 있습니다.

속성을 글로벌 보조 인덱스로 프로젝션할 경우에는 프로비저닝된 처리량 비용과 스토리지 비용 간 균형을 고려해야 합니다.

- 대기 시간이 가장 낮은 몇 개의 속성만 액세스해야 할 경우 해당 속성만 글로벌 보조 인덱스로 프로젝션하는 방법을 고려해 볼 수 있습니다. 인덱스가 작을수록 스토리지 비용과 쓰기 비용이 절감됩니다.

- 애플리케이션이 키가 아닌 일부 속성에 빈번하게 액세스할 경우 해당 속성을 글로벌 보조 인덱스로 프로젝션하는 방법을 고려해야 합니다. 글로벌 보조 인덱스의 추가 스토리지 비용이 빈번한 테이블 스캔 수행으로 발생하는 비용보다 경제적입니다.
- 키가 아닌 속성 대부분에 자주 액세스해야 하는 경우 해당 속성이나 심지어 전체 기본 테이블을 글로벌 보조 인덱스로 프로젝션할 수 있습니다. 이렇게 하면 유연성이 극대화합니다. 그러나 스토리지 비용이 심지어 2배로 증가하게 됩니다.
- 애플리케이션이 테이블에 빈번하게 쿼리하지 않지만 테이블 데이터에 대해 많은 쓰기 또는 업데이트 작업을 수행해야 할 경우 KEYS_ONLY를 프로젝션할 수 있습니다. 이 경우 글로벌 보조 인덱스는 크기는 최소화되지만 쿼리 작업에 필요할 경우 계속 사용할 수 있습니다.

## 글로벌 보조 인덱스에서 데이터 읽기

Query 및 Scan 작업을 사용하여 글로벌 보조 인덱스에서 항목을 가져올 수 있습니다. GetItem 및 BatchGetItem 작업은 글로벌 보조 인덱스에 사용할 수 없습니다.

### 글로벌 보조 인덱스 쿼리

Query 작업을 사용하여 글로벌 보조 인덱스에서 하나 이상의 항목에 액세스할 수 있습니다. 쿼리에서는 사용하려는 인덱스의 이름과 기본 테이블의 이름, 쿼리 결과에 반환할 속성, 적용할 쿼리 조건을 지정해야 합니다. DynamoDB는 결과를 오름차순 또는 내림차순으로 반환할 수 있습니다.

순위표 애플리케이션에 대해 게임 데이터를 요청하는 Query에서 다음과 같은 데이터가 반환된 경우를 가정해 보세요.

```
{
    "TableName": "GameScores",
    "IndexName": "GameTitleIndex",
    "KeyConditionExpression": "GameTitle = :v_title",
    "ExpressionAttributeValues": {
        ":v_title": {"S": "Meteor Blasters"}
    },
    "ProjectionExpression": "UserId, TopScore",
    "ScanIndexForward": false
}
```

이 쿼리에서

- DynamoDB는 GameTitle 파티션 키로 Meteor Blassts의 인덱스 항목을 찾아 GameTitleIndex에 액세스합니다. 이 키가 있는 모든 인덱스 항목은 빠른 검색을 위해 인접한 상태로 저장됩니다.

- 이 게임에서 DynamoDB는 인덱스를 사용하여 이 게임의 모든 사용자 ID와 최고 점수에 액세스합니다.
- 결과가 반환되고, `ScanIndexForward` 파라미터가 false로 설정되어 있으므로 내림차순으로 정렬됩니다.

### 글로벌 보조 인덱스 검색

`Scan` 작업을 사용하여 글로벌 보조 인덱스에서 모든 데이터를 검색할 수 있습니다. 요청에 기본 테이블 이름과 인덱스 이름을 제공해야 합니다. DynamoDB는 `Scan`을 사용하여 인덱스에 있는 모든 데이터를 읽고 애플리케이션에 반환합니다. 또한 일부 데이터만 반환하고 나머지 데이터를 무시하도록 요청할 수 있습니다. 그러려면 `Scan` 작업의 `FilterExpression` 파라미터를 사용합니다. 자세한 내용은 스캔에 대한 필터 표현식 단원을 참조하십시오.

## 테이블과 글로벌 보조 인덱스 간 데이터 동기화

DynamoDB는 각 글로벌 보조 인덱스를 기본 테이블과 자동으로 동기화합니다. 애플리케이션이 테이블에서 항목을 쓰거나 삭제하면 해당 테이블의 글로벌 보조 인덱스가 최종적으로 일관된 모델을 사용하여 비동기적으로 업데이트됩니다. 애플리케이션이 인덱스에 직접 쓰기를 수행하는 경우는 없습니다. 하지만, DynamoDB가 이러한 인덱스를 유지하는 방식이 어떤 영향을 미치는지 이해하는 것이 중요합니다.

글로벌 보조 인덱스는 기본 테이블에서 읽기/쓰기 용량 모드를 상속합니다. 자세한 내용은 DynamoDB에서 용량 모드 전환 시 고려 사항 단원을 참조하십시오.

글로벌 보조 인덱스를 생성할 때 하나 이상의 인덱스 키 속성과 해당 데이터 형식을 지정합니다. 다시 말해, 기본 테이블에 항목을 쓸 때마다 해당 속성의 데이터 형식이 인덱스 키 스키마의 데이터 형식과 일치해야 합니다. `GameTitleIndex`의 경우, 인덱스의 `GameTitle` 파티션 키가 `String` 데이터 유형으로 정의됩니다. 인덱스의 `TopScore` 정렬 키는 유형이 `Number`입니다. `GameScores` 테이블에 항목을 추가하고 `GameTitle` 또는 `TopScore`에 다른 데이터 형식을 지정할 경우 DynamoDB에서 데이터 형식 불일치로 인해 `ValidationException`을 반환합니다.

테이블에 항목을 추가하거나 삭제하면 해당 테이블의 글로벌 보조 인덱스가 최종 일관성 방식으로 업데이트됩니다. 정상적인 조건에서 해당 테이블 데이터의 변경 사항은 거의 밀리초 수준으로 글로벌 보조 인덱스에 적용됩니다. 하지만, 간혹 장애가 발생할 경우 긴 적용 지연이 발생할 수 있습니다. 따라서, 애플리케이션은 글로벌 보조 인덱스의 쿼리가 최신이 아닌 결과를 반환할 수 있음을 예상하고 그러한 상황을 처리할 수 있어야 합니다.

테이블에 항목을 쓸 경우 모든 글로벌 보조 인덱스 정렬 키의 속성을 지정하지 않아도 됩니다. 예를 들어 GameTitleIndex의 경우, GameScores 테이블에 새 항목을 쓰기 위해 TopScore 속성의 값을 지정할 필요가 없습니다. 이 경우 DynamoDB는 이 특정 항목의 인덱스에 데이터를 쓰지 않습니다.

글로벌 보조 인덱스가 많은 테이블은 인덱스가 적은 테이블보다 쓰기 작업에 더 높은 비용이 발생합니다. 자세한 내용은 글로벌 보조 인덱스에 대한 프로비저닝된 처리량 고려 사항 단원을 참조하십시오.

## 글로벌 보조 인덱스를 사용하는 테이블 클래스

글로벌 보조 인덱스는 항상 기본 테이블과 동일한 테이블 클래스를 사용합니다. 테이블에 대해 새 글로벌 보조 인덱스가 추가될 때마다 새 인덱스는 기본 테이블과 동일한 테이블 클래스를 사용합니다. 테이블의 테이블 클래스가 업데이트되면 연결된 모든 글로벌 보조 인덱스도 업데이트됩니다.

## 글로벌 보조 인덱스에 대한 프로비저닝된 처리량 고려 사항

프로비저닝된 모드 테이블에서 글로벌 보조 인덱스를 생성할 때는 해당 인덱스에 예상되는 워크로드에 대한 읽기 및 쓰기 용량 단위를 지정해야 합니다. 글로벌 보조 인덱스의 프로비저닝된 처리량 설정 값은 기본 테이블의 설정 값과 별개입니다. 글로벌 보조 인덱스의 Query 작업은 기본 테이블이 아닌 인덱스에서 읽기 용량 단위를 소비합니다. 테이블에 항목을 추가, 업데이트 또는 삭제하면 해당 테이블의 글로벌 보조 인덱스도 업데이트됩니다. 이러한 인덱스 업데이트에서는 기본 테이블이 아닌 인덱스의 용량 단위가 사용됩니다.

예를 들어 글로벌 보조 인덱스를 Query하고 프로비저닝된 읽기 용량을 초과할 경우 요청이 제한됩니다. 테이블에 많은 쓰기 작업을 수행하지만 해당 테이블의 글로벌 보조 인덱스의 쓰기 용량이 부족할 경우 테이블의 쓰기 작업이 제한됩니다.

> **Important**
>
> 잠재적 조절을 피하기 위해서 글로벌 보조 인덱스용 프로비저닝된 쓰기 용량은 새로운 업데이트로 기본 테이블과 글로벌 보조 인덱스가 모두 쓰여지기 때문에 기본 테이블의 쓰기 용량 이상이어야 합니다.

글로벌 보조 인덱스의 프로비저닝된 처리량 설정을 확인하려면 DescribeTable 작업을 사용합니다. 테이블의 모든 글로벌 보조 인덱스에 대한 세부 정보가 반환됩니다.

### 읽기 용량 단위

글로벌 보조 인덱스는 최종적으로 일관된 읽기를 지원하며, 각각 읽기 용량 단위의 절반을 소비합니다. 즉, 단일 글로벌 보조 인덱스 쿼리는 읽기 용량 단위당 최대 2 × 4KB = 8KB까지 검색할 수 있습니다.

글로벌 보조 인덱스 쿼리의 경우 DynamoDB는 테이블에 대한 쿼리와 같은 방식으로 프로비저닝된 읽기 작업을 계산합니다. 유일한 차이는 기본 테이블 항목의 크기가 아닌 인덱스 항목의 크기를 기준으로 계산이 이루어진다는 점입니다. 읽기 용량 단위의 수는 반환된 모든 항목에서 프로젝션된 모든 속성 크기의 합계입니다. 그런 다음 결과가 다음 4KB 경계로 반올림됩니다. DynamoDB에서 프로비저닝된 처리량을 계산하는 방법에 대한 자세한 내용은 DynamoDB 프로비저닝된 용량 모드 단원을 참조하세요.

`Query` 작업에서 반환된 결과의 최대 크기는 1MB입니다. 여기에는 반환된 모든 항목의 속성 이름 및 값의 크기가 포함됩니다.

예를 들어 각 항목에 2,000바이트의 데이터가 포함된 글로벌 보조 인덱스를 가정해 보겠습니다. 이제 이 인덱스를 `Query`했을 때 쿼리의 `KeyConditionExpression`이 8개 항목과 일치한다고 가정해 봅니다. 일치하는 항목의 총 크기는 2,000바이트 × 8개 항목 = 16,000바이트입니다. 그런 다음 결과가 가장 가까운 4KB 경계로 반올림됩니다. 글로벌 보조 인덱스 쿼리는 최종적으로 일관되므로 총 비용은 0.5 × (16KB/4KB), 즉 2 읽기 용량 단위입니다.

쓰기 용량 단위

테이블의 항목을 추가, 업데이트 또는 삭제하고 이로 인해 글로벌 보조 인덱스가 영향을 받을 경우 글로벌 보조 인덱스에서 작업에 프로비저닝된 쓰기 용량 단위를 소비합니다. 쓰기의 총 할당된 처리량 비용은 기본 테이블에 쓰기 작업으로 소비된 쓰기 용량 단위와 글로벌 보조 인덱스를 업데이트하여 소비된 쓰기 용량 단위의 합계로 구성됩니다. 글로벌 보조 인덱스 업데이트가 필요하지 않은 테이블 쓰기의 경우 인덱스에서 쓰기 용량이 소비되지 않습니다.

테이블 쓰기가 성공하기 위해서는 테이블 및 모든 글로벌 보조 인덱스의 프로비저닝 처리량 설정에 쓰기를 수용할 수 있을 만큼 충분한 쓰기 용량이 있어야 합니다. 그렇지 않을 경우 테이블에 대한 쓰기가 제한됩니다.

글로벌 보조 인덱스에 항목을 쓰는 비용은 몇 가지 요인에 따라 달라집니다.

- 인덱싱된 속성을 정의하는 테이블에 새 항목을 쓰거나 이전에 정의되지 않은 인덱싱된 속성을 정의하도록 기존 항목을 업데이트하는 경우 항목을 인덱스에 넣으려면 1번의 쓰기 작업이 필요합니다.
- 테이블을 업데이트하여 인덱스 키 속성 값이 A에서 B로 변경될 경우 두 번의 쓰기, 즉, 인덱스에서 이전 항목을 삭제하는 쓰기와 새 항목을 인덱스에 추가하는 쓰기가 필요합니다.
- 항목이 인덱스에 있지만 테이블 쓰기로 인해 인덱스 속성이 삭제된 경우 인덱스에서 기존 항목 프로젝션을 삭제하는 한 번의 쓰기가 필요합니다.
- 항목이 업데이트되기 전후에 인덱스에 항목이 없을 경우 인덱스에 추가 쓰기 비용이 발생하지 않습니다.
- 테이블을 업데이트하여 인덱스 키 스키마에 프로젝션된 속성 값이 변경되었지만 인덱스 키 속성 값이 변경되지 않은 경우 인덱스에 프로젝션된 속성 값을 업데이트하는 한 번의 쓰기가 필요합니다.

이러한 모든 요인은 인덱스에 있는 각 항목의 크기가 쓰기 용량 단위를 계산하기 위한 1KB 항목 크기보다 작거나 같은 경우를 가정합니다. 이보다 큰 인덱스 항목에서는 추가 쓰기 용량 단위가 필요합니다. 쿼리에서 어떤 속성을 반환해야 하는지 고려하고 해당 속성만 인덱스에 프로젝션함으로써 쓰기 비용을 최소화할 수 있습니다.

## 글로벌 보조 인덱스에 대한 스토리지 고려 사항

애플리케이션에서 테이블에 항목을 쓰는 경우 DynamoDB는 해당 속성이 표시되어야 하는 모든 글로벌 보조 인덱스에 올바른 속성 하위 집합을 자동으로 복사합니다. AWS 계정에는 기본 테이블의 항목 스토리지 및 해당 테이블에 있는 글로벌 보조 인덱스의 속성 스토리지 비용이 청구됩니다.

인덱스 항목에서 사용하는 공간의 양은 다음의 합계입니다.

- 기본 테이블 기본 키(파티션 및 정렬 키)의 크기(바이트)
- 인덱스 키 속성의 크기(바이트)
- 프로젝션된 속성(있는 경우)의 크기(바이트)
- 인덱스 항목당 오버헤드의 100바이트

글로벌 보조 인덱스의 스토리지 요구 사항을 추정하려면 인덱스의 평균 항목 크기를 추정한 다음 기본 테이블에서 글로벌 보조 인덱스 키 속성이 있는 항목의 수를 곱합니다.

테이블에 특정 속성이 정의되지 않은 항목이 포함되어 있지만 해당 속성이 인덱스 파티션 키 또는 정렬 키로 정의된 경우 DynamoDB에서 해당 항목의 데이터를 인덱스에 쓰지 않습니다.

## DynamoDB에서 글로벌 보조 인덱스 관리

이 단원에서는 Amazon DynamoDB에서 글로벌 보조 인덱스를 생성, 수정 및 삭제하는 방법을 설명합니다.

주제

- 글로벌 보조 인덱스가 있는 테이블 생성
- 테이블의 글로벌 보조 인덱스 설명
- 기존 테이블에 글로벌 보조 인덱스 추가
- 글로벌 보조 인덱스 삭제
- 생성 중 글로벌 보조 인덱스 수정

### 글로벌 보조 인덱스가 있는 테이블 생성

하나 이상의 글로벌 보조 인덱스를 사용하는 테이블을 생성하려면 `CreateTable` 작업에 `GlobalSecondaryIndexes` 파라미터를 추가하면 됩니다. 쿼리 유연성을 극대화하도록 테이블당 최대 20개의 글로벌 보조 인덱스(기본 할당량)를 생성할 수 있습니다.

보조 파티션 키의 역할을 할 속성 한 가지를 지정해야 합니다. 선택 사항으로 인덱스 정렬 키에 또 하나의 속성을 지정할 수 있습니다. 그렇다고 두 가지 키 속성 중 하나가 테이블의 키 속성과 반드시 같을 필요는 없습니다. 예를 들어 GameScores 테이블(DynamoDB에서 글로벌 보조 인덱스 사용 참조)에서는 `TopScore`와 `TopScoreDateTime` 둘 다 키 속성이 아닙니다. 파티션 키가 `TopScore`이고 정렬 키가 `TopScoreDateTime`인 글로벌 보조 인덱스를 생성할 수 있습니다. 이러한 인덱스는 최고 점수와 경기 시간 사이의 상관관계 여부를 판단할 때 사용됩니다.

각 인덱스 키 속성은 `String`, `Number` 또는 `Binary` 형식의 스칼라여야 합니다. (문서 또는 집합일 수 없습니다.) 모든 데이터 형식의 속성을 글로벌 보조 인덱스로 프로젝션할 수 있습니다. 여기에는 스칼라, 문서, 집합이 포함됩니다. 데이터 형식에 대한 전체 목록은 데이터 타입 단원을 참조하세요.

프로비저닝된 모드를 사용할 경우 인덱스의 `ProvisionedThroughput` 설정값은 `ReadCapacityUnits`와 `WriteCapacityUnits`로 구성하여 입력해야 합니다. 여기에서 프로비저닝 처리량 설정 값은 테이블의 설정 값과 다르지만 동작 방식은 비슷합니다. 자세한 내용은 글로벌 보조 인덱스에 대한 프로비저닝된 처리량 고려 사항 단원을 참조하십시오.

글로벌 보조 인덱스는 기본 테이블에서 읽기/쓰기 용량 모드를 상속합니다. 자세한 내용은 DynamoDB에서 용량 모드 전환 시 고려 사항 단원을 참조하십시오.

> **Note**
>
> 채우기 작업과 진행 중인 쓰기 작업은 글로벌 보조 인덱스 내의 쓰기 처리량을 공유합니다. 새 GSI를 생성할 때 파티션 키를 선택하면 새 인덱스의 파티션 키 값에서 균일하지 않거나 좁은 데이터 또는 트래픽 분산이 생성되는지 확인하는 것이 중요할 수 있습니다. 이 경우 채우기 및 쓰기 작업이 동시에 발생하고 쓰기가 기본 테이블로 제한될 수 있습니다. 이 서비스는 이 시나리오의 잠재력을 최소화하는 조치를 취하지만 인덱스 파티션 키, 선택한 프로젝션 또는 인덱스 기본 키의 희소성과 관련하여 고객 데이터의 형태를 파악하지 못합니다.
> 새 글로벌 보조 인덱스에서 파티션 키 값의 데이터 또는 분포가 좁거나 왜곡되어 있다고 의심되는 경우 운영상 중요한 테이블에 새 인덱스를 추가하기 전에 다음을 고려하세요.
>
> - 애플리케이션에서 최소 양의 트래픽을 유도하고 있을 때 인덱스를 추가하는 것이 가장 안전할 수 있습니다.

- 기본 테이블 및 인덱스에서 CloudWatch Contributor Insights를 활성화하는 것이 좋습니다. 이렇게 하면 트래픽 분포에 대한 중요한 정보를 얻을 수 있습니다.
- 프로비저닝된 용량 모드 기본 테이블 및 인덱스의 경우 새 인덱스의 프로비저닝된 쓰기 용량을 기본 테이블의 두 배 이상으로 설정합니다. 프로세스 전체에 `WriteThrottleEvents`, `ThrottledRequests`, `OnlineIndexPercentageProgress`, `OnlineIndexConsumedWriteCapacity` 및 `OnlineIndexThrottleEvents` CloudWatch 지표를 감시합니다. 진행 중인 작업을 크게 제한하지 않고 채우기를 적절한 시간에 완료하려면 프로비저닝된 쓰기 용량을 필요에 따라 조정합니다.
- 쓰기 제한으로 인해 작업에 영향을 받고 새 GSI에서 프로비저닝된 쓰기 용량을 늘려도 문제가 해결되지 않는 경우 인덱스 생성을 취소할 수 있어야 합니다.

### 테이블의 글로벌 보조 인덱스 설명

테이블에 속한 모든 글로벌 보조 인덱스의 상태를 확인하려면 `DescribeTable` 작업을 사용합니다. 응답에서 `GlobalSecondaryIndexes` 부분은 현재 각 인덱스 상태(`IndexStatus`)와 함께 테이블에 속한 모든 인덱스를 나타냅니다.

글로벌 보조 인덱스의 `IndexStatus`는 다음 중 하나입니다.

- `CREATING` - 현재 인덱스를 생성 중이며, 아직 사용할 수 없습니다.
- `ACTIVE` - 현재 인덱스가 사용 대기 상태이며, 애플리케이션이 인덱스에 대한 `Query` 작업을 실행할 수 있습니다.
- `UPDATING` - 인덱스의 프로비저닝된 처리량 설정을 변경 중입니다.
- `DELETING` - 현재 인덱스를 삭제하고 있으며 더 이상 사용할 수 없습니다.

DynamoDB가 글로벌 보조 인덱스 빌드를 마치면 인덱스 상태는 `CREATING`에서 `ACTIVE`로 바뀝니다.

### 기존 테이블에 글로벌 보조 인덱스 추가

글로벌 보조 인덱스를 기존 테이블에 추가하려면 `UpdateTable` 작업에 `GlobalSecondaryIndexUpdates` 파라미터를 사용합니다. 이때 입력해야 하는 정보는 다음과 같습니다.

- 인덱스 이름. 인덱스 이름은 모든 테이블 인덱스 중에서 식별할 수 있도록 고유해야 합니다.
- 인덱스의 키 스키마. 인덱스 파티션 키로 한 가지 속성을 반드시 지정해야 하며, 인덱스 정렬 키는 옵션으로 다른 속성을 지정할 수 있습니다. 그렇다고 두 가지 키 속성 중 하나가 테이블의 키 속성

과 반드시 같을 필요는 없습니다. 각 스키마 속성의 데이터 형식은 스칼라(String, Number 또는 Binary)가 되어야 합니다.

- 테이블에서 인덱스로 가져올 속성:
  - KEYS_ONLY - 인덱스의 각 항목은 테이블 파티션 키 및 정렬 키 값, 그리고 인덱스 키 값으로만 구성됩니다.
  - INCLUDE - KEYS_ONLY에서 설명한 속성 외에도 지정하는 키가 아닌 다른 속성이 보조 인덱스에 포함됩니다.
  - ALL - 인덱스에 소스 테이블의 모든 속성이 포함됩니다.
- 인덱스의 할당 처리량 설정 값(ReadCapacityUnits와 WriteCapacityUnits로 구성). 이러한 프로비저닝된 처리량 설정은 테이블의 설정과 별개입니다.

UpdateTable 작업당 글로벌 보조 인덱스를 1개만 생성할 수 있습니다.

인덱스 생성 단계

새로운 글로벌 보조 인덱스를 기존 테이블에 추가하는 경우 인덱스 빌드 중에도 테이블을 계속 사용할 수 있습니다. 하지만 쿼리 작업 중에는 상태가 CREATING에서 ACTIVE로 바뀔 때까지 새로운 인덱스를 사용할 수 없습니다.

> **Note**
>
> 글로벌 보조 인덱스 생성에는 Application Auto Scaling이 사용되지 않습니다. MIN Application Auto Scaling 용량을 늘려도 글로벌 보조 인덱스의 생성 시간이 단축되지는 않습니다.

내부적으로 DynamoDB는 다음 두 단계로 인덱스를 빌드합니다.

리소스 할당

DynamoDB가 인덱스 빌드에 필요한 컴퓨팅 및 스토리지 리소스를 할당합니다.

리소스 할당 단계에서는 IndexStatus 속성이 CREATING이고, Backfilling 속성이 false입니다. DescribeTable 작업으로 테이블과 테이블에 속한 모든 보조 인덱스의 상태를 가져옵니다.

인덱스가 리소스 할당 단계에 있는 동안에는 인덱스를 삭제하거나 상위 테이블을 삭제할 수 없습니다. 또한 인덱스 또는 테이블의 프로비저닝된 처리량을 수정할 수 없습니다. 테이블에 다른 인덱스를 추가하거나 삭제할 수 없습니다. 그러나 이와 같은 다른 인덱스의 프로비저닝된 처리양을 수정할 수는 있습니다.

### 채우기 작업(Backfilling)

테이블의 각 항목에 대해 DynamoDB는 프로젝션(KEYS_ONLY, INCLUDE 또는 ALL)에 따라 인덱스에 쓸 속성 집합을 결정합니다. 그런 다음 결정된 속성을 인덱스에 기록합니다. DynamoDB가 채우기(backfill) 단계에 들어서면 테이블에서 추가, 삭제 또는 업데이트되는 항목을 추적합니다. 그리고, 이러한 항목들의 속성이 인덱스에서도 추가, 삭제 또는 업데이트됩니다.

채우기 단계에서는 `IndexStatus` 속성이 `CREATING`으로 설정되어 있고 `Backfilling` 속성은 true입니다. `DescribeTable` 작업으로 테이블과 테이블에 속한 모든 보조 인덱스의 상태를 가져옵니다.

인덱스 채우기 중에는 상위 테이블을 삭제할 수 없습니다. 단, 인덱스를 삭제하거나 테이블과 테이블에 속한 글로벌 보조 인덱스의 프로비저닝된 처리량을 변경하는 것은 가능합니다.

> **Note**
>
> 채우기 단계에서 불일치 항목이 있을 경우 쓰기 작업이 성공하는 경우도 있고 거부될 수도 있습니다. 채우기 단계가 끝나면 새 인덱스의 키 스키마를 위반하는 모든 항목에 대한 쓰기 작업은 거부됩니다. 따라서 채우기 단계 이후에는 위반 감지기 도구를 실행하여 발생했을 수도 있는 키 위반을 찾아내 해결하는 것이 좋습니다. 자세한 내용은 DynamoDB에서 인덱스 키 위반 검색 및 수정 단원을 참조하십시오.

리소스 할당 및 채우기 단계가 진행 중일 때 인덱스는 `CREATING` 상태입니다. 이 단계에서 DynamoDB는 테이블에서 읽기 작업을 수행합니다. 글로벌 보조 인덱스를 채우기 위한 기본 테이블의 읽기 작업에는 요금이 부과되지 않습니다. 그러나 새로 생성된 글로벌 보조 인덱스를 채우기 위한 쓰기 작업에는 요금이 청구됩니다.

인덱스 빌드를 마치면 상태가 `ACTIVE`로 바뀝니다. 인덱스가 `ACTIVE` 상태가 되기 전에는 `Query` 또는 `Scan` 작업을 할 수 없습니다.

> **Note**
>
> 경우에 따라 인덱스 키 위반으로 인해 DynamoDB가 테이블의 데이터를 인덱스에 쓰지 못할 수도 있습니다. 이러한 상황은 다음과 같은 경우에 발생할 수 있습니다.
>
> - 속성 값의 데이터 형식이 인덱스 키 스키마 데이터 형식의 데이터 형식과 일치하지 않습니다.
> - 속성 크기가 인덱스 키 속성의 최대 길이를 초과합니다.

- 인덱스 키 속성에 빈 문자열 또는 빈 이진 속성 값이 있습니다.

인덱스 키 위반이 발생해도 글로벌 보조 인덱스 생성은 방해를 받지 않습니다. 그러나 인덱스가 `ACTIVE` 상태가 되면 인덱스에는 위반 키가 존재하지 않습니다.
DynamoDB는 이러한 문제를 찾아 해결하기 위한 독립형 도구를 제공합니다. 자세한 내용은 DynamoDB에서 인덱스 키 위반 검색 및 수정 단원을 참조하십시오.

### 큰 테이블에 글로벌 보조 인덱스 추가

글로벌 보조 인덱스 빌드에 소요되는 시간은 다음과 같은 몇 가지 요인에 따라 달라집니다.

- 테이블 크기
- 인덱스에 추가할 수 있는 테이블 항목 수
- 인덱스로 가져올 속성 수
- 인덱스에 할당된 쓰기 용량
- 인덱스 빌드 중 메인 테이블에 대한 쓰기 작업

아주 큰 테이블에 글로벌 보조 인덱스를 추가할 때는 생성 프로세스가 완료되는 데 오래 걸릴 수 있습니다. 이때 진행 상황을 모니터링하여 인덱스의 쓰기 용량이 충분한지 확인하려면 다음 Amazon CloudWatch 지표를 참조하세요.

- `OnlineIndexPercentageProgress`
- `OnlineIndexConsumedWriteCapacity`
- `OnlineIndexThrottleEvents`

DynamoDB와 관련된 CloudWatch 지표에 대한 자세한 내용은 DynamoDB 지표 단원을 참조하세요.

**Important**

글로벌 보조 인덱스를 생성하거나 업데이트하기 전에 먼저 매우 큰 테이블 허용 목록이 필요할 수 있습니다. 테이블 허용 목록을 추가하려면 AWS 지원 팀에 문의하시기 바랍니다.

인덱스에 프로비저닝된 쓰기 처리량 설정 값이 낮을 수록 인덱스 빌드에 걸리는 시간이 길어집니다. 따라서 새로운 글로벌 보조 인덱스의 빌드 시간을 단축하려면 프로비저닝된 쓰기 용량을 일시적으로 높여야 합니다.

> **Note**
>
> 일반적으로 인덱스에 할당된 쓰기 용량은 테이블에 할당된 쓰기 용량에 비해 1.5배로 높여 설정하는 것이 좋습니다. 이는 많은 사용 사례에 유용하지만, 실제 요건은 더 높거나 낮을 수도 있습니다.

인덱스 채움 단계에서는 DynamoDB가 내부 시스템 용량을 사용하여 테이블에서 읽어옵니다. 이는 인덱스 생성에 미치는 영향을 최소화하는 동시에 테이블의 읽기 용량이 고갈되는 것을 방지하기 위해서입니다.

하지만 수신되는 쓰기 작업 볼륨이 인덱스에 할당된 쓰기 용량을 초과할 수는 있습니다. 이는 인덱스에 대한 쓰기 작업이 몰려들면서 인덱스 생성 시간이 늘어나는 병목 현상 시나리오입니다. 따라서 인덱스 빌드 중에는 인덱스의 Amazon CloudWatch 지표를 모니터링하면서 소비되는 쓰기 용량이 프로비저닝된 용량을 초과하지 않는지 살펴보는 것이 바람직합니다. 병목 현상 시나리오에서는 인덱스에 할당된 쓰기 용량을 늘려서 채움 단계에서 쓰기 작업이 몰리는 것을 방지해야 합니다.

인덱스 생성을 마치면 정상적인 애플리케이션 사용량을 반영하여 할당된 쓰기 용량을 설정해야 합니다.

글로벌 보조 인덱스 삭제

글로벌 보조 인덱스가 더 이상 필요하지 않으면 `UpdateTable` 작업으로 삭제할 수 있습니다.

`UpdateTable` 작업당 글로벌 보조 인덱스를 하나만 삭제할 수 있습니다.

글로벌 보조 인덱스 삭제 중에도 상위 테이블의 읽기 또는 쓰기 작업에 미치는 영향은 전혀 없습니다. 삭제가 진행되는 동안에도 다른 인덱스의 할당 처리량은 변경 가능합니다.

> **Note**
>
> - `DeleteTable` 작업을 사용하여 테이블을 삭제할 때는 해당 테이블의 글로벌 보조 인덱스가 모두 삭제됩니다.
> - 글로벌 보조 인덱스의 삭제 작업에 대해서는 계정에 요금이 부과되지 않습니다.

### 생성 중 글로벌 보조 인덱스 수정

인덱스 빌드 중에도 `DescribeTable` 작업을 사용하여 현재 진행 중인 단계를 확인할 수 있습니다. 인덱스의 설명에 포함되는 부울 속성인 `Backfilling`은 DynamoDB가 현재 테이블의 항목과 함께 인덱스를 로드하고 있는지 여부를 나타냅니다. `Backfilling`이 true이면 리소스 할당 단계가 끝나고 인덱스 채움 단계가 시작된 것을 의미합니다.

채움 단계가 진행 중일 때는 인덱스의 할당 처리량 파라미터를 업데이트할 수 있습니다. 이러한 파라미터 업데이트의 목적은 인덱스의 빌드 속도를 높이는 데 있습니다. 즉, 인덱스 빌드 중 인덱스의 쓰기 용량을 늘렸다가 빌드가 완료되면 용량을 줄입니다. 인덱스의 할당 처리량 설정 값을 변경하려면 `UpdateTable` 작업을 사용합니다. 그러면 인덱스 상태가 `UPDATING`으로 바뀌고, `Backfilling`은 인덱스를 사용할 준비가 될 때까지 true를 유지합니다.

채우기 단계 중에는 생성되고 있는 인덱스를 삭제할 수 있습니다. 이 단계에서는 테이블에(서) 다른 인덱스를 추가하거나 삭제할 수 없습니다.

> **Note**
>
> 인덱스가 `CreateTable` 작업의 일환으로 생성된 경우에는 `Backfilling` 속성이 `DescribeTable` 출력에 표시되지 않습니다. 자세한 내용은 인덱스 생성 단계 단원을 참조하십시오.

## DynamoDB에서 인덱스 키 위반 검색 및 수정

Amazon DynamoDB는 글로벌 보조 인덱스 생성의 채우기 단계 중 테이블의 각 항목을 검사하여 인덱스에 포함하기에 적합한지 여부를 결정합니다. 일부 항목은 인덱스 키 위반을 유발하여 적합하지 않을 수도 있습니다. 이러한 경우에는 항목이 테이블에서는 유지되지만 인덱스에 해당 항목에 상응하는 키가 없습니다.

인덱스 키 위반은 다음 상황에서 발생합니다.

- 속성 값과 인덱스 키 스키마 데이터 형식 간에 데이터 형식이 일치하지 않습니다. 예를 들어 `GameScores` 테이블의 항목 중 하나가 형식이 `String`인 `TopScore` 값을 하나 가지고 있었다고 가정해 보겠습니다. 파티션 키 `TopScore`이고 형식이 `Number`인 글로벌 보조 인덱스를 추가했다면 테이블의 해당 항목이 인덱스 키를 위반하게 됩니다.
- 테이블의 속성 값이 인덱스 키 속성의 최대 길이를 초과합니다. 파티션 키의 최대 길이는 2,048바이트이고 정렬 키의 최대 길이는 1,024바이트입니다. 테이블에서 해당 속성 값 중 하나가 이러한 제한을 초과하는 경우 테이블에서 이 항목은 인덱스 키를 위반합니다.

> **Note**
>
> 인덱스 키로 사용되는 속성에 대해 문자열 또는 이진 속성 값이 설정된 경우 속성 값의 길이는 0보다 커야 합니다. 그렇지 않으면 테이블의 항목이 인덱스 키를 위반하게 됩니다.
> 이 도구는 현재 이 인덱스 키 위반에 플래그를 지정하지 않습니다.

인덱스 키 위반이 발생하면 중단 없이 채우기 단계가 계속되지만, 위반 항목이 인덱스에 포함되지 않습니다. 채우기 단계가 끝나면 새 인덱스의 키 스키마를 위반하는 모든 항목 쓰기 작업은 거부됩니다.

테이블에서 인덱스 키를 위반하는 속성 값을 식별하여 수정하려면 위반 감지기 도구를 사용합니다. 위반 감지기를 실행하려면 스캔할 테이블의 이름, 글로벌 보조 인덱스 파티션 키 및 정렬 키의 이름과 데이터 형식, 인덱스 키 위반이 발생하는 경우 취할 조치를 지정하는 구성 파일을 생성합니다. 위반 감지기는 다음 두 가지 모드 중 하나로 실행할 수 있습니다.

- 감지 모드 - 인덱스 키 위반을 감지합니다. 감지 모드를 사용하여 테이블에서 글로벌 보조 인덱스의 키 위반을 초래하는 항목을 보고합니다. 원할 경우 이러한 위반 테이블 항목을 발견되는 즉시 삭제하도록 요청할 수도 있습니다. 검색 모드의 출력이 파일에 작성되어 추후 분석용으로 사용할 수 있습니다.
- 수정 모드 - 인덱스 키 위반을 수정합니다. 수정 모드에서는 위반 감지기가 감지 모드의 출력 파일과 동일한 형식의 입력 파일을 읽습니다. 수정 모드에서는 입력 파일의 레코드를 읽고 각 레코드마다 테이블에서 해당 항목을 삭제하거나 업데이트합니다. 항목을 업데이트하도록 선택하는 경우 입력 파일을 편집하여 이러한 업데이트에 대한 알맞은 값을 설정해야 합니다.

### 위반 감지기 다운로드 및 실행

위반 감지기는 실행 가능한 Java 아카이브(`.jar` 파일)로 제공되며 Windows, macOS 또는 Linux 컴퓨터에서 실행됩니다. 위반 감지기를 사용하려면 Java 1.7 이상 및 Apache Maven이 필요합니다.

- GitHub에서 위반 감지기 다운로드

`README.md` 파일의 지침에 따라 Mavn을 사용하여 위반 감지기를 다운로드 및 설치합니다.

위반 감지기를 시작하려면 `ViolationDetector.java`를 작성했던 디렉터리로 이동하여 다음 명령을 입력합니다.

```
java -jar ViolationDetector.jar [options]
```

위반 감지기 명령줄에서는 다음 옵션을 사용할 수 있습니다.

- `-h | --help` - 위반 감지기의 사용법 요약 및 옵션을 출력합니다.
- `-p | --configFilePath value` - 위반 감지기 구성 파일의 정규화된 이름입니다. 자세한 내용은 위반 감지기 구성 파일 단원을 참조하십시오.
- `-t | --detect value` - 테이블에서 인덱스 키 위반을 감지하고 위반 감지기 출력 파일에 기록합니다. 이 파라미터의 값이 `keep`으로 설정되면 키 위반 항목이 수정되지 않습니다. 값이 `delete`로 설정되면 키 위반 항목이 테이블에서 삭제됩니다.
- `-c | --correct value` - 입력 파일에서 인덱스 키 위반을 읽고 테이블에서 이러한 항목에 대한 수정 작업을 수행합니다. 이 파라미터의 값이 `update`로 설정되면 키 위반 항목이 위반하지 않는 새 값으로 업데이트됩니다. 값이 `delete`로 설정되면 키 위반 항목이 테이블에서 삭제됩니다.

위반 감지기 구성 파일

런타임 시, 위반 감지기 도구에는 구성 파일이 필요합니다. 이 파일의 파라미터는 위반 감지기에서 액세스할 수 있는 DynamoDB 리소스와 이 도구에서 사용할 수 있는 프로비저닝된 처리량을 결정합니다. 다음 표는 이러한 파라미터에 대해 설명합니다.

| 파라미터 이름 | 설명 | 필수? |
|---|---|---|
| `awsCredentialsFile` | AWS 자격 증명을 포함하는 파일의 정규화된 이름입니다. 자격 증명 파일은 다음 형식이어야 합니다.<br>`accessKey = access_key_id_goes_here`<br>`secretKey = secret_key_goes_here` | 예 |
| `dynamoDBRegion` | 테이블이 상주하는 AWS 리전입니다. 예: `us-west-2` . | 예 |
| `tableName` | 스캔할 DynamoDB 테이블의 이름입니다. | 예 |

| 파라미터 이름 | 설명 | 필수? |
| --- | --- | --- |
| gsiHashKeyName | 인덱스 파티션 키의 이름입니다. | 예 |
| gsiHashKeyType | 인덱스 파티션 키의 데이터 형식(String, Number 또는 Binary).<br><br>S \| N \| B | 예 |
| gsiRangeKeyName | 인덱스 정렬 키의 이름입니다. 인덱스에 단순 기본 키(파티션 키)만 있는 경우 이 파라미터를 지정하지 마세요. | 아니요 |
| gsiRangeKeyType | 인덱스 정렬 키의 데이터 형식입니다(String, Number 또는 Binary).<br><br>S \| N \| B<br><br>인덱스에 단순 기본 키(파티션 키)만 있는 경우 이 파라미터를 지정하지 마세요. | 아니요 |
| recordDetails | 인덱스 키 위반에 대한 자세한 정보를 출력 파일에 작성할지 여부를 나타냅니다. true(기본값)로 설정하면 위반 항목에 대한 전체 정보가 보고됩니다. false로 설정하면 위반 수만 보고됩니다. | 아니요 |

| 파라미터 이름 | 설명 | 필수? |
| --- | --- | --- |
| `recordGsiValueInViolationRecord` | 위반 인덱스 키의 값을 출력 파일에 작성할지 여부를 나타냅니다. `true`(기본값)로 설정하면 키 값이 보고됩니다. `false`로 설정하면 키 값이 보고되지 않습니다. | 아니요 |
| `detectionOutputPath` | 위반 감지기 출력 파일의 전체 경로입니다. 이 파라미터는 로컬 디렉터리 또는 Amazon Simple Storage Service(Amazon S3)에 쓰기를 지원합니다. 예를 들어, 다음과 같습니다.<br><br>`detectionOutputPath = //local/path/filename.csv`<br><br>`detectionOutputPath = s3://bucket/filename.csv`<br><br>출력 파일의 정보가 CSV 형식(쉼표로 구분된 값)으로 표시됩니다. `detectionOutputPath` 를 설정하지 않는 경우 출력 파일의 이름은 `violation_detection.csv` 이고 현재 작업 디렉터리에 작성됩니다. | 아니요 |

| 파라미터 이름 | 설명 | 필수? |
| --- | --- | --- |
| numOfSegments | 위반 감지기에서 테이블을 스캔할 때 사용할 병렬 스캔 세그먼트 수입니다. 기본값은 1이며, 이는 테이블이 순차적으로 스캔됨을 뜻합니다. 값이 2 이상이면 위반 감지기가 테이블을 여러 논리 세그먼트, 그리고 이와 동일한 스캔 스레드 수로 나눕니다.<br><br>numOfSegments 의 최대 설정은 4096입니다.<br><br>대형 테이블의 경우 병렬 스캔이 일반적으로 순차 스캔보다 빠릅니다. 또한, 테이블이 매우 커서 여러 파티션에 걸쳐 있는 경우 병렬 스캔을 수행하면 읽기 작업이 여러 파티션에 고르게 분산됩니다.<br>DynamoDB의 병렬 스캔에 대한 자세한 내용은 병렬 스캔 단원을 참조하세요. | 아니요 |
| numOfViolations | 출력 파일에 작성할 인덱스 키 위반의 상한 값입니다. -1(기본값)로 설정하면 전체 테이블이 스캔됩니다. 양의 정수로 설정하면 위반 감지기가 해당 위반 수에 도달한 후 중지됩니다. | 아니요 |

| 파라미터 이름 | 설명 | 필수? |
| --- | --- | --- |
| numOfRecords | 테이블에서 스캔할 항목 수입니다. -1(기본값)로 설정하면 전체 테이블이 스캔됩니다. 양의 정수로 설정하면 위반 감지기가 테이블에서 해당 개수의 항목을 스캔한 후 중지됩니다. | 아니요 |
| readWriteIOPSPercent | 테이블 스캔 중에 사용되는 프로비저닝된 읽기 용량 단위의 백분율을 조정합니다. 사용할 수 있는 값은 1~100입니다. 기본값(25)을 사용하면 위반 감지기가 테이블의 프로비저닝된 읽기 처리량 중 25% 이하만 사용합니다. | 아니요 |
| correctionInputPath | 위반 감지기 수정 입력 파일의 전체 경로입니다. 위반 감지기를 수정 모드로 실행하면 이 파일의 내용이 테이블에서 글로벌 보조 인덱스를 위반하는 데이터 항목을 수정하거나 삭제하는 데 사용됩니다.<br><br>correctionInputPath 파일의 형식은 detectionOutputPath 파일의 형식과 동일합니다. 따라서 검색 모드의 출력을 수정 모드의 입력으로 처리할 수 있습니다. | 아니요 |

| 파라미터 이름 | 설명 | 필수? |
| --- | --- | --- |
| correctionOutputPath | 위반 감지기 수정 출력 파일의 전체 경로입니다. 이 파일은 업데이트 오류가 있는 경우에만 생성됩니다.<br><br>이 파라미터는 로컬 디렉터리 또는 Amazon S3에 쓰기를 지원합니다. 예를 들어, 다음과 같습니다.<br><br>`correctionOutputPath = //local/path/filename.csv`<br><br>`correctionOutputPath = s3://bucket/filename.csv`<br><br>출력 파일의 정보가 CSV 형식으로 표시됩니다. `correctionOutputPath` 를 설정하지 않는 경우 출력 파일의 이름은 `violation_update_errors.csv` 이고 현재 작업 디렉터리에 작성됩니다. | 아니요 |

### 탐지

인덱스 키 위반을 감지하려면 위반 감지기를 `--detect` 명령줄 옵션과 함께 사용합니다. 이 옵션의 작동 방식을 알아보기 위해 `ProductCatalog` 테이블을 살펴봅니다. 다음은 테이블의 항목 목록입니다. 기본 키(`Id`)와 `Price` 속성만 표시됩니다.

| Id(프라이머리 키) | 가격 |
|---|---|
| 101 | 5 |
| 102 | 20 |
| 103 | 200 |
| 201 | 100 |
| 202 | 200 |
| 203 | 300 |
| 204 | 400 |
| 205 | 500 |

Price의 모든 값은 Number 형식입니다. 하지만 DynamoDB에는 스키마가 없으므로 숫자가 아닌 Price 값이 지정된 항목을 추가할 수 있습니다. 예를 들어 또 하나의 항목을 ProductCatalog 테이블에 추가한다고 가정해 보겠습니다.

| Id(프라이머리 키) | 가격 |
|---|---|
| 999 | "Hello" |

이제 테이블에는 총 9개 항목이 있습니다.

이제 테이블에 새 글로벌 보조 인덱스 PriceIndex를 추가합니다. 이 인덱스의 기본 키는 Number 형식의 파티션 키 Price입니다. 인덱스가 빌드되면 인덱스에 8개 항목이 포함되지만 ProductCatalog 테이블에는 9개 항목이 있습니다. 이러한 불일치가 발생한 이유는 "Hello" 값은 String 형식이지만, PriceIndex의 기본 키는 Number 형식이기 때문입니다. String 값은 글로벌 보조 인덱스 키를 위반하므로 인덱스에 없습니다.

이 시나리오에서 위반 감지기를 사용하려면 먼저 다음과 같은 구성 파일을 만듭니다.

```
# Properties file for violation detection tool configuration.
```

```
# Parameters that are not specified will use default values.

awsCredentialsFile = /home/alice/credentials.txt
dynamoDBRegion = us-west-2
tableName = ProductCatalog
gsiHashKeyName = Price
gsiHashKeyType = N
recordDetails = true
recordGsiValueInViolationRecord = true
detectionOutputPath = ./gsi_violation_check.csv
correctionInputPath = ./gsi_violation_check.csv
numOfSegments = 1
readWriteIOPSPercent = 40
```

그 다음으로, 다음 예제와 같이 위반 감지기를 실행합니다.

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --detect keep

Violation detection started: sequential scan, Table name: ProductCatalog, GSI name:
 PriceIndex
Progress: Items scanned in total: 9,    Items scanned by this thread: 9,    Violations
 found by this thread: 1, Violations deleted by this thread: 0
Violation detection finished: Records scanned: 9, Violations found: 1, Violations
 deleted: 0, see results at: ./gsi_violation_check.csv
```

`recordDetails` 구성 파라미터를 `true`로 설정하면 다음 예에서처럼 위반 감지기에서 각 위반에 대한 자세한 정보를 출력 파일에 기록합니다.

```
Table Hash Key,GSI Hash Key Value,GSI Hash Key Violation Type,GSI Hash Key Violation
 Description,GSI Hash Key Update Value(FOR USER),Delete Blank Attributes When Updating?
(Y/N)

999,"{""S"":""Hello""}",Type Violation,Expected: N Found: S,,
```

출력 파일은 CSV 형식입니다. 이 파일의 첫 번째 줄은 헤더이며, 그 뒤에는 인덱스 키를 위반하는 항목당 한 개 레코드가 이어집니다. 이러한 위반 레코드의 필드는 다음과 같습니다.

- Table hash key - 테이블에 있는 항목의 파티션 키 값입니다.
- Table range key - 테이블에 있는 항목의 정렬 키 값입니다.
- GSI hash key value - 글로벌 보조 인덱스의 파티션 키 값입니다.
- GSI hash key violation type - `Type Violation` 또는 `Size Violation`입니다.

- GSI hash key violation description - 위반 원인입니다.
- GSI hash key update Value(FOR USER) - 수정 모드에서 해당 속성에 대한 새로운 사용자 제공 값입니다.
- GSI range key value - 글로벌 보조 인덱스의 정렬 키 값입니다.
- GSI range key violation type - `Type Violation` 또는 `Size Violation`입니다.
- GSI range key violation description - 위반 원인입니다.
- GSI range key update Value(FOR USER) - 수정 모드에서 해당 속성에 대한 새로운 사용자 제공 값입니다.
- Delete blank attribute when Updating(Y/N) - 수정 모드에서 다음 필드 중 하나가 비어 있는 경우에만 테이블에서 위반 항목을 삭제할지(Y) 또는 유지할지(N)를 결정합니다.
  - `GSI Hash Key Update Value(FOR USER)`
  - `GSI Range Key Update Value(FOR USER)`

  이러한 필드 중 하나가 비어 있지 않으면 `Delete Blank Attribute When Updating(Y/N)`이 적용되지 않습니다.

> **Note**
>
> 출력 형식은 구성 파일 및 명령줄 옵션에 따라 다를 수 있습니다. 예를 들어 테이블이 단순 기본 키(정렬 키 없음)를 가질 경우 출력에 정렬 키 필드가 없습니다.
> 파일의 위반 레코드는 순서가 정렬되어 있지 않을 수도 있습니다.

수정

인덱스 키 위반을 수정하려면 위반 감지기를 `--correct` 명령줄 옵션과 함께 사용합니다. 수정 모드에서는 위반 감지기가 `correctionInputPath` 파라미터에 지정된 입력 파일을 읽습니다. 이 파일의 형식은 `detectionOutputPath` 파일과 동일하므로 검색의 출력을 수정의 입력으로 사용할 수 있습니다.

위반 감지기에서는 인덱스 키 위반을 수정할 수 있는 두 가지 방법을 제공합니다.

- 위반 삭제 - 위반하는 속성 값이 있는 테이블 항목을 삭제합니다.
- 위반 업데이트 - 위반하는 속성 값을 위반하지 않는 값으로 바꾸어 테이블 항목을 업데이트합니다.

어느 경우든, 검색 모드의 출력 파일을 수정 모드의 입력으로 사용할 수 있습니다.

ProductCatalog 예를 계속 살펴보면서, 이번에는 테이블에서 위반 항목을 삭제하려 한다고 가정해 보겠습니다. 이 작업을 수행하려면 다음 명령줄을 사용합니다.

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --correct delete
```

이 때 위반 항목을 삭제할지 묻는 확인 메시지가 표시됩니다.

```
Are you sure to delete all violations on the table?y/n
y
Confirmed, will delete violations on the table...
Violation correction from file started: Reading records from file: ./
gsi_violation_check.csv, will delete these records from table.
Violation correction from file finished: Violations delete: 1, Violations Update: 0
```

이제는 ProductCatalog와 PriceIndex의 항목 수가 동일합니다.

## 글로벌 보조 인덱스로 작업: Java

AWS SDK for Java Document API를 사용하여 하나 이상의 글로벌 보조 인덱스가 포함된 Amazon DynamoDB 테이블을 만들고, 테이블의 인덱스를 설명하고, 인덱스를 사용하여 쿼리를 수행할 수 있습니다.

다음은 테이블 작업의 일반적인 단계입니다.

1. DynamoDB 클래스의 인스턴스를 만듭니다.
2. 해당하는 요청 객체를 만들어 작업의 필수 및 선택적 파라미터를 제공합니다.
3. 이전 단계에서 만든 클라이언트가 제공한 적절한 메서드를 호출합니다.

주제

- 글로벌 보조 인덱스가 있는 테이블 생성
- 글로벌 보조 인덱스가 있는 테이블 설명
- 글로벌 보조 인덱스 쿼리
- 예: AWS SDK for Java 문서 API를 사용하는 글로벌 보조 인덱스

### 글로벌 보조 인덱스가 있는 테이블 생성

글로벌 보조 인덱스는 테이블을 만들 때 동시에 만들 수 있습니다. 이렇게 하려면 CreateTable을 사용하고 하나 이상의 글로벌 보조 인덱스에 대한 사양을 제공합니다. 다음에 예로 나오는 Java 코드

는 날씨 데이터에 대한 정보를 담고 있는 테이블을 만듭니다. 파티션 키는 Location이고 정렬 키는 Date입니다. PrecipIndex라는 이름의 글로벌 보조 인덱스를 사용하면 다양한 위치의 강수 데이터에 빠르게 액세스할 수 있습니다.

다음은 DynamoDB Document API를 사용하여 글로벌 보조 인덱스가 있는 테이블을 생성하는 단계입니다.

1. DynamoDB 클래스의 인스턴스를 만듭니다.
2. CreateTableRequest 클래스 인스턴스를 만들어 요청 정보를 입력합니다.

   이때 입력해야 하는 정보는 테이블 이름, 기본 키, 그리고 프로비저닝된 처리량 값입니다. 글로벌 보조 인덱스의 경우 인덱스 이름, 프로비저닝된 처리량 설정, 인덱스 정렬 키의 속성 정의, 인덱스의 키 스키마, 속성 프로젝션을 제공해야 합니다.
3. 요청 객체를 파라미터로 입력하여 createTable 메서드를 호출합니다.

다음 Java 코드 예는 앞의 단계를 보여줍니다. 이 코드는 글로벌 보조 인덱스(PrecipIndex)가 있는 테이블(WeatherData)을 생성합니다. 인덱스 파티션 키는 Date이고 정렬 키는 Precipitation입니다. 모든 테이블 속성이 인덱스로 프로젝션되지 않습니다. 이 인덱스를 쿼리하여 특정 날짜에 대한 날씨 데이터를 가져와서 필요에 따라 강수량을 기준으로 데이터를 정렬할 수 있습니다.

Precipitation은 해당 테이블의 키 속성이 아니므로 필요하지 않습니다. 그러나 Precipitation이 없는 WeatherData 항목은 PrecipIndex에 표시되지 않습니다.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

// Attribute definitions
ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();

attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Location")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Date")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Precipitation")
    .withAttributeType("N"));
```

```
// Table key schema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Location")
    .withKeyType(KeyType.HASH));  //Partition key
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
    .withKeyType(KeyType.RANGE));  //Sort key

// PrecipIndex
GlobalSecondaryIndex precipIndex = new GlobalSecondaryIndex()
    .withIndexName("PrecipIndex")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 10)
        .withWriteCapacityUnits((long) 1))
        .withProjection(new Projection().withProjectionType(ProjectionType.ALL));

ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();

indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
    .withKeyType(KeyType.HASH));  //Partition key
indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Precipitation")
    .withKeyType(KeyType.RANGE));  //Sort key

precipIndex.setKeySchema(indexKeySchema);

CreateTableRequest createTableRequest = new CreateTableRequest()
    .withTableName("WeatherData")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 5)
        .withWriteCapacityUnits((long) 1))
    .withAttributeDefinitions(attributeDefinitions)
    .withKeySchema(tableKeySchema)
    .withGlobalSecondaryIndexes(precipIndex);

Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

DynamoDB에서 테이블을 만들고 테이블 상태가 ACTIVE로 설정될 때까지 기다려야 합니다. 그런 다음 테이블에 데이터 항목을 입력할 수 있습니다.

### 글로벌 보조 인덱스가 있는 테이블 설명

테이블의 글로벌 보조 인덱스에 관한 자세한 내용은 `DescribeTable` 단원을 참조하세요. 각 인덱스에 대해 인덱스의 이름, 키 스키마 및 프로젝션된 속성에 액세스할 수 있습니다.

다음은 테이블의 글로벌 보조 인덱스 정보에 액세스하는 단계입니다.

1. `DynamoDB` 클래스의 인스턴스를 만듭니다.
2. `Table` 클래스의 인스턴스를 만들어 사용할 인덱스를 표시합니다.
3. `Table` 객체의 `describe` 메서드를 호출합니다.

다음 Java 코드 예는 앞의 단계를 보여줍니다.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
TableDescription tableDesc = table.describe();


Iterator<GlobalSecondaryIndexDescription> gsiIter =
 tableDesc.getGlobalSecondaryIndexes().iterator();
while (gsiIter.hasNext()) {
    GlobalSecondaryIndexDescription gsiDesc = gsiIter.next();
    System.out.println("Info for index "
         + gsiDesc.getIndexName() + ":");

    Iterator<KeySchemaElement> kseIter = gsiDesc.getKeySchema().iterator();
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = gsiDesc.getProjection();
    System.out.println("\tThe projection type is: "
        + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: "
            + projection.getNonKeyAttributes());
    }
}
```

### 글로벌 보조 인덱스 쿼리

테이블을 Query할 때와 거의 동일한 방식으로 글로벌 보조 인덱스에서 Query를 사용할 수 있습니다. 인덱스 이름, 인덱스 파티션 키 및 정렬 키(있는 경우)의 쿼리 기준, 반환하려는 속성을 지정해야 합니다. 이 예제에서 인덱스는 PrecipIndex이고, 해당 파티션 키는 Date이고, 정렬 키는 Precipitation입니다. 인덱스 쿼리에서는 특정 날짜에 대해 강수량이 0보다 큰 모든 날씨 데이터를 반환합니다.

다음은 AWS SDK for Java Document API를 사용하여 글로벌 보조 인덱스를 쿼리하는 단계입니다.

1. DynamoDB 클래스의 인스턴스를 만듭니다.
2. Table 클래스의 인스턴스를 만들어 사용할 인덱스를 표시합니다.
3. 쿼리할 인덱스에 대한 Index 클래스의 인스턴스를 만듭니다.
4. Index 객체의 query 메서드를 호출합니다.

속성 이름 Date는 DynamoDB 예약어입니다. 따라서 KeyConditionExpression에서 표현식 속성 이름을 자리 표시자로 사용해야 합니다.

다음 Java 코드 예는 앞의 단계를 보여줍니다.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
Index index = table.getIndex("PrecipIndex");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("#d = :v_date and Precipitation = :v_precip")
    .withNameMap(new NameMap()
        .with("#d", "Date"))
    .withValueMap(new ValueMap()
        .withString(":v_date","2013-08-10")
        .withNumber(":v_precip",0));

ItemCollection<QueryOutcome> items = index.query(spec);
Iterator<Item> iter = items.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next().toJSONPretty());
```

```
}
```

예: AWS SDK for Java 문서 API를 사용하는 글로벌 보조 인덱스

다음 Java 코드 예제는 글로벌 보조 인덱스로 작업하는 방법을 보여줍니다. 이 예에서는 `Issues`라는 테이블을 만듭니다. 이 테이블은 소프트웨어 개발을 위한 간단한 버그 추적 시스템에서 사용할 수 있습니다. 파티션 키는 `IssueId`이고 정렬 키는 `Title`입니다. 이 테이블에는 세 개의 글로벌 보조 인덱스가 있습니다.

- `CreateDateIndex` - 파티션 키는 `CreateDate`이고 정렬 키는 `IssueId`입니다. 테이블 키 외에도 `Description` 및 `Status` 속성도 인덱스로 프로젝션됩니다.
- `TitleIndex` - 파티션 키는 `Title`이고 정렬 키는 `IssueId`입니다. 테이블 키 이외의 속성은 인덱스로 프로젝션되지 않습니다.
- `DueDateIndex` - 파티션 키는 `DueDate`이고 정렬 키는 없습니다. 모든 테이블 속성이 인덱스로 프로젝션되지 않습니다.

`Issues` 테이블이 생성되면 프로그램에서 소프트웨어 버그 보고서를 나타내는 데이터와 함께 테이블을 로드합니다. 그런 다음 글로벌 보조 인덱스를 사용하여 데이터를 쿼리합니다. 마지막으로 프로그램에서 `Issues` 테이블을 삭제합니다.

다음 예제를 테스트하기 위한 단계별 지침은 Java 코드 예 단원을 참조하세요.

Example

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
```

```
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.GlobalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class DocumentAPIGlobalSecondaryIndexExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    public static String tableName = "Issues";

    public static void main(String[] args) throws Exception {

        createTable();
        loadData();

        queryIndex("CreateDateIndex");
        queryIndex("TitleIndex");
        queryIndex("DueDateIndex");

        deleteTable(tableName);

    }

    public static void createTable() {

        // Attribute definitions
        ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();

        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("IssueId").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Title").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("CreateDate").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("DueDate").withAttributeType("S"));

        // Key schema for table
        ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
```

```
        tableKeySchema.add(new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.HASH)); //
Partition

                    // key
        tableKeySchema.add(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.RANGE)); // Sort

                   // key

        // Initial provisioned throughput settings for the indexes
        ProvisionedThroughput ptIndex = new
ProvisionedThroughput().withReadCapacityUnits(1L)
                .withWriteCapacityUnits(1L);

        // CreateDateIndex
        GlobalSecondaryIndex createDateIndex = new
GlobalSecondaryIndex().withIndexName("CreateDateIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("CreateDate").withKeyType(KeyType.HASH), //
Partition

                          // key
                        new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort

                 // key
                .withProjection(
                        new
Projection().withProjectionType("INCLUDE").withNonKeyAttributes("Description",
"Status"));

        // TitleIndex
        GlobalSecondaryIndex titleIndex = new
GlobalSecondaryIndex().withIndexName("TitleIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.HASH), // Partition

                     // key
                        new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort
```

```
                // key
               .withProjection(new Projection().withProjectionType("KEYS_ONLY"));

       // DueDateIndex
       GlobalSecondaryIndex dueDateIndex = new
GlobalSecondaryIndex().withIndexName("DueDateIndex")
               .withProvisionedThroughput(ptIndex)
               .withKeySchema(new
KeySchemaElement().withAttributeName("DueDate").withKeyType(KeyType.HASH)) //
Partition

                     // key
               .withProjection(new Projection().withProjectionType("ALL"));

       CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
               .withProvisionedThroughput(
                       new ProvisionedThroughput().withReadCapacityUnits((long)
1).withWriteCapacityUnits((long) 1))

.withAttributeDefinitions(attributeDefinitions).withKeySchema(tableKeySchema)
               .withGlobalSecondaryIndexes(createDateIndex, titleIndex, dueDateIndex);

       System.out.println("Creating table " + tableName + "...");
       dynamoDB.createTable(createTableRequest);

       // Wait for table to become active
       System.out.println("Waiting for " + tableName + " to become ACTIVE...");
       try {
           Table table = dynamoDB.getTable(tableName);
           table.waitForActive();
       } catch (InterruptedException e) {
           e.printStackTrace();
       }
   }

   public static void queryIndex(String indexName) {

       Table table = dynamoDB.getTable(tableName);


System.out.println("\n***********************************************************\n");
       System.out.print("Querying index " + indexName + "...");
```

```
        Index index = table.getIndex(indexName);

        ItemCollection<QueryOutcome> items = null;

        QuerySpec querySpec = new QuerySpec();

        if (indexName == "CreateDateIndex") {
            System.out.println("Issues filed on 2013-11-01");
            querySpec.withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
                    .withValueMap(new ValueMap().withString(":v_date",
"2013-11-01").withString(":v_issue", "A-"));
            items = index.query(querySpec);
        } else if (indexName == "TitleIndex") {
            System.out.println("Compilation errors");
            querySpec.withKeyConditionExpression("Title = :v_title and
begins_with(IssueId, :v_issue)")
                    .withValueMap(
                            new ValueMap().withString(":v_title", "Compilation
error").withString(":v_issue", "A-"));
            items = index.query(querySpec);
        } else if (indexName == "DueDateIndex") {
            System.out.println("Items that are due on 2013-11-30");
            querySpec.withKeyConditionExpression("DueDate = :v_date")
                    .withValueMap(new ValueMap().withString(":v_date", "2013-11-30"));
            items = index.query(querySpec);
        } else {
            System.out.println("\nNo valid index name provided");
            return;
        }

        Iterator<Item> iterator = items.iterator();

        System.out.println("Query: printing results...");

        while (iterator.hasNext()) {
            System.out.println(iterator.next().toJSONPretty());
        }

    }

    public static void deleteTable(String tableName) {
```

```
        System.out.println("Deleting table " + tableName + "...");

        Table table = dynamoDB.getTable(tableName);
        table.delete();

        // Wait for table to be deleted
        System.out.println("Waiting for " + tableName + " to be deleted...");
        try {
            table.waitForDelete();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static void loadData() {

        System.out.println("Loading data into table " + tableName + "...");

        // IssueId, Title,
        // Description,
        // CreateDate, LastUpdateDate, DueDate,
        // Priority, Status

        putItem("A-101", "Compilation error", "Can't compile Project X - bad version
number. What does this mean?",
                "2013-11-01", "2013-11-02", "2013-11-10", 1, "Assigned");

        putItem("A-102", "Can't read data file", "The main data file is missing, or the
permissions are incorrect",
                "2013-11-01", "2013-11-04", "2013-11-30", 2, "In progress");

        putItem("A-103", "Test failure", "Functional test of Project X produces
errors", "2013-11-01", "2013-11-02",
                "2013-11-10", 1, "In progress");

        putItem("A-104", "Compilation error", "Variable 'messageCount' was not
initialized.", "2013-11-15",
                "2013-11-16", "2013-11-30", 3, "Assigned");

        putItem("A-105", "Network issue", "Can't ping IP address 127.0.0.1. Please fix
this.", "2013-11-15",
                "2013-11-16", "2013-11-19", 5, "Assigned");

    }
```

```
    public static void putItem(

            String issueId, String title, String description, String createDate, String
 lastUpdateDate, String dueDate,
            Integer priority, String status) {

        Table table = dynamoDB.getTable(tableName);

        Item item = new Item().withPrimaryKey("IssueId", issueId).withString("Title",
 title)
                .withString("Description", description).withString("CreateDate",
 createDate)
                .withString("LastUpdateDate", lastUpdateDate).withString("DueDate",
 dueDate)
                .withNumber("Priority", priority).withString("Status", status);

        table.putItem(item);
    }

}
```

## 글로벌 보조 인덱스로 작업: .NET

AWS SDK for .NET 하위 수준 API를 사용하여 하나 이상의 글로벌 보조 인덱스가 포함된 Amazon DynamoDB 테이블을 만들고, 테이블의 인덱스를 설명하고, 인덱스를 사용하여 쿼리를 수행할 수 있습니다. 이러한 작업은 해당 DynamoDB 작업에 매핑됩니다. 자세한 내용은 Amazon DynamoDB API 참조를 참조하세요.

다음은 .NET 하위 수준 API를 사용하여 테이블 작업을 할 때 따라야 할 공통 단계입니다.

1. `AmazonDynamoDBClient` 클래스의 인스턴스를 만듭니다.
2. 해당하는 요청 객체를 만들어 작업의 필수 및 선택적 파라미터를 제공합니다.

   예를 들어 `CreateTableRequest` 객체를 만들어 테이블을 생성하거나 `QueryRequest` 객체를 만들어 테이블 또는 인덱스를 쿼리합니다.
3. 이전 단계에서 만든 클라이언트가 제공한 적절한 메서드를 실행합니다.

### 주제

- 글로벌 보조 인덱스가 있는 테이블 생성

- 글로벌 보조 인덱스가 있는 테이블 설명
- 글로벌 보조 인덱스 쿼리
- 예: AWS SDK for .NET 하위 수준 API를 사용하는 글로벌 보조 인덱스

### 글로벌 보조 인덱스가 있는 테이블 생성

글로벌 보조 인덱스는 테이블을 만들 때 동시에 만들 수 있습니다. 이렇게 하려면 `CreateTable`을 사용하고 하나 이상의 글로벌 보조 인덱스에 대한 사양을 제공합니다. 다음에 예로 나오는 C# 코드는 날씨 데이터에 대한 정보를 담고 있는 테이블을 만듭니다. 파티션 키는 `Location`이고 정렬 키는 `Date`입니다. `PrecipIndex`라는 이름의 글로벌 보조 인덱스를 사용하면 다양한 위치의 강수 데이터에 빠르게 액세스할 수 있습니다.

다음은 .NET 하위 수준 API를 사용하여 글로벌 보조 인덱스가 포함된 테이블을 생성하는 단계입니다.

1. `AmazonDynamoDBClient` 클래스의 인스턴스를 만듭니다.
2. `CreateTableRequest` 클래스 인스턴스를 만들어 요청 정보를 입력합니다.

   이때 입력해야 하는 정보는 테이블 이름, 기본 키, 그리고 프로비저닝된 처리량 값입니다. 글로벌 보조 인덱스의 경우 인덱스 이름, 프로비저닝된 처리량 설정, 인덱스 정렬 키의 속성 정의, 인덱스의 키 스키마, 속성 프로젝션을 제공해야 합니다.
3. 요청 객체를 파라미터로 입력하여 `CreateTable` 메서드를 실행합니다.

다음 C# 코드 예제에서는 이전 단계를 설명합니다. 이 코드는 글로벌 보조 인덱스(`PrecipIndex`)가 있는 테이블(`WeatherData`)을 생성합니다. 인덱스 파티션 키는 `Date`이고 정렬 키는 `Precipitation`입니다. 모든 테이블 속성이 인덱스로 프로젝션되지 않습니다. 이 인덱스를 쿼리하여 특정 날짜에 대한 날씨 데이터를 가져와서 필요에 따라 강수량을 기준으로 데이터를 정렬할 수 있습니다.

`Precipitation`은 해당 테이블의 키 속성이 아니므로 필요하지 않습니다. 그러나 `Precipitation`이 없는 `WeatherData` 항목은 `PrecipIndex`에 표시되지 않습니다.

```
client = new AmazonDynamoDBClient();
string tableName = "WeatherData";

// Attribute definitions
var attributeDefinitions = new List<AttributeDefinition>()
{
    {new AttributeDefinition{
```

```
        AttributeName = "Location",
        AttributeType = "S"}},
    {new AttributeDefinition{
        AttributeName = "Date",
        AttributeType = "S"}},
    {new AttributeDefinition(){
        AttributeName = "Precipitation",
        AttributeType = "N"}
    }
};

// Table key schema
var tableKeySchema = new List<KeySchemaElement>()
{
    {new KeySchemaElement {
        AttributeName = "Location",
        KeyType = "HASH"}},  //Partition key
    {new KeySchemaElement {
        AttributeName = "Date",
        KeyType = "RANGE"}  //Sort key
    }
};

// PrecipIndex
var precipIndex = new GlobalSecondaryIndex
{
    IndexName = "PrecipIndex",
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)10,
        WriteCapacityUnits = (long)1
    },
    Projection = new Projection { ProjectionType = "ALL" }
};

var indexKeySchema = new List<KeySchemaElement> {
    {new KeySchemaElement { AttributeName = "Date", KeyType = "HASH"}},  //Partition
 key
    {new KeySchemaElement{AttributeName = "Precipitation",KeyType = "RANGE"}}  //Sort
 key
};

precipIndex.KeySchema = indexKeySchema;
```

```
CreateTableRequest createTableRequest = new CreateTableRequest
{
    TableName = tableName,
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)5,
        WriteCapacityUnits = (long)1
    },
    AttributeDefinitions = attributeDefinitions,
    KeySchema = tableKeySchema,
    GlobalSecondaryIndexes = { precipIndex }
};

CreateTableResponse response = client.CreateTable(createTableRequest);
Console.WriteLine(response.CreateTableResult.TableDescription.TableName);
Console.WriteLine(response.CreateTableResult.TableDescription.TableStatus);
```

DynamoDB에서 테이블을 만들고 테이블 상태가 ACTIVE로 설정될 때까지 기다려야 합니다. 그런 다음 테이블에 데이터 항목을 입력할 수 있습니다.

글로벌 보조 인덱스가 있는 테이블 설명

테이블의 글로벌 보조 인덱스에 관한 자세한 내용은 DescribeTable 단원을 참조하세요. 각 인덱스에 대해 인덱스의 이름, 키 스키마 및 프로젝션된 속성에 액세스할 수 있습니다.

다음은 .NET 하위 수준 API를 사용하여 테이블의 글로벌 보조 인덱스 정보에 액세스하는 단계입니다.

1. AmazonDynamoDBClient 클래스의 인스턴스를 만듭니다.
2. 요청 객체를 파라미터로 입력하여 describeTable 메서드를 실행합니다.

   DescribeTableRequest 클래스 인스턴스를 만들어 요청 정보를 입력합니다. 테이블 이름을 입력해야 합니다.

3.

다음 C# 코드 예제에서는 이전 단계를 설명합니다.

Example

```
client = new AmazonDynamoDBClient();
string tableName = "WeatherData";
```

```
DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest
 { TableName = tableName});

List<GlobalSecondaryIndexDescription> globalSecondaryIndexes =
response.DescribeTableResult.Table.GlobalSecondaryIndexes;

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.

foreach (GlobalSecondaryIndexDescription gsiDescription in globalSecondaryIndexes) {
     Console.WriteLine("Info for index " + gsiDescription.IndexName + ":");

     foreach (KeySchemaElement kse in gsiDescription.KeySchema) {
          Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
     }

      Projection projection = gsiDescription.Projection;
      Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

      if (projection.ProjectionType.ToString().Equals("INCLUDE")) {
           Console.WriteLine("\t\tThe non-key projected attributes are: "
                + projection.NonKeyAttributes);
      }
}
```

글로벌 보조 인덱스 쿼리

테이블을 Query할 때와 거의 동일한 방식으로 글로벌 보조 인덱스에서 Query를 사용할 수 있습니다. 인덱스 이름, 인덱스 파티션 키 및 정렬 키(있는 경우)의 쿼리 기준, 반환하려는 속성을 지정해야 합니다. 이 예제에서 인덱스는 PrecipIndex이고, 해당 파티션 키는 Date이고, 정렬 키는 Precipitation입니다. 인덱스 쿼리에서는 특정 날짜에 대해 강수량이 0보다 큰 모든 날씨 데이터를 반환합니다.

다음은 .NET 하위 수준 API를 사용하여 글로벌 보조 인덱스를 쿼리하는 단계입니다.

1. AmazonDynamoDBClient 클래스의 인스턴스를 만듭니다.
2. QueryRequest 클래스 인스턴스를 만들어 요청 정보를 입력합니다.
3. 요청 객체를 파라미터로 입력하여 query 메서드를 실행합니다.

속성 이름 Date는 DynamoDB 예약어입니다. 따라서 KeyConditionExpression에서 표현식 속성 이름을 자리 표시자로 사용해야 합니다.

다음 C# 코드 예제에서는 이전 단계를 설명합니다.

### Example

```
client = new AmazonDynamoDBClient();

QueryRequest queryRequest = new QueryRequest
{
    TableName = "WeatherData",
    IndexName = "PrecipIndex",
    KeyConditionExpression = "#dt = :v_date and Precipitation > :v_precip",
    ExpressionAttributeNames = new Dictionary<String, String> {
        {"#dt", "Date"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
        {":v_date", new AttributeValue { S =  "2013-08-01" }},
        {":v_precip", new AttributeValue { N =  "0" }}
    },
    ScanIndexForward = true
};

var result = client.Query(queryRequest);

var items = result.Items;
foreach (var currentItem in items)
{
    foreach (string attr in currentItem.Keys)
    {
        Console.Write(attr + "---> ");
        if (attr == "Precipitation")
        {
            Console.WriteLine(currentItem[attr].N);
    }
    else
    {
        Console.WriteLine(currentItem[attr].S);
    }

         }
     Console.WriteLine();
}
```

### 예: AWS SDK for .NET 하위 수준 API를 사용하는 글로벌 보조 인덱스

다음 C# 코드 예제는 글로벌 보조 인덱스로 작업하는 방법을 보여줍니다. 이 예에서는 `Issues`라는 테이블을 만듭니다. 이 테이블은 소프트웨어 개발을 위한 간단한 버그 추적 시스템에서 사용할 수 있습니다. 파티션 키는 `IssueId`이고 정렬 키는 `Title`입니다. 이 테이블에는 세 개의 글로벌 보조 인덱스가 있습니다.

- `CreateDateIndex` - 파티션 키는 `CreateDate`이고 정렬 키는 `IssueId`입니다. 테이블 키 외에도 `Description` 및 `Status` 속성도 인덱스로 프로젝션됩니다.
- `TitleIndex` - 파티션 키는 `Title`이고 정렬 키는 `IssueId`입니다. 테이블 키 이외의 속성은 인덱스로 프로젝션되지 않습니다.
- `DueDateIndex` - 파티션 키는 `DueDate`이고 정렬 키는 없습니다. 모든 테이블 속성이 인덱스로 프로젝션되지 않습니다.

`Issues` 테이블이 생성되면 프로그램에서 소프트웨어 버그 보고서를 나타내는 데이터와 함께 테이블을 로드합니다. 그런 다음 글로벌 보조 인덱스를 사용하여 데이터를 쿼리합니다. 마지막으로 프로그램에서 `Issues` 테이블을 삭제합니다.

다음 샘플을 테스트하기 위한 단계별 지침은 .NET 코드 예시 섹션을 참조하세요.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelGlobalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        public static String tableName = "Issues";

        public static void Main(string[] args)
        {
```

```
            CreateTable();
            LoadData();

            QueryIndex("CreateDateIndex");
            QueryIndex("TitleIndex");
            QueryIndex("DueDateIndex");

            DeleteTable(tableName);

            Console.WriteLine("To continue, press enter");
            Console.Read();
        }

        private static void CreateTable()
        {
            // Attribute definitions
            var attributeDefinitions = new List<AttributeDefinition>()
        {
            {new AttributeDefinition {
                 AttributeName = "IssueId", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "Title", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "CreateDate", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "DueDate", AttributeType = "S"
             }}
        };

            // Key schema for table
            var tableKeySchema = new List<KeySchemaElement>() {
            {
                new KeySchemaElement {
                    AttributeName= "IssueId",
                    KeyType = "HASH" //Partition key
                }
            },
            {
                new KeySchemaElement {
                    AttributeName = "Title",
                    KeyType = "RANGE" //Sort key
```

```
                }
            }
        };

            // Initial provisioned throughput settings for the indexes
            var ptIndex = new ProvisionedThroughput
            {
                ReadCapacityUnits = 1L,
                WriteCapacityUnits = 1L
            };

            // CreateDateIndex
            var createDateIndex = new GlobalSecondaryIndex()
            {
                IndexName = "CreateDateIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "CreateDate", KeyType = "HASH" //Partition key
                },
                new KeySchemaElement {
                    AttributeName = "IssueId", KeyType = "RANGE" //Sort key
                }
            },
                Projection = new Projection
                {
                    ProjectionType = "INCLUDE",
                    NonKeyAttributes = {
                    "Description", "Status"
                }
                }
            };

            // TitleIndex
            var titleIndex = new GlobalSecondaryIndex()
            {
                IndexName = "TitleIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "Title", KeyType = "HASH" //Partition key
                },
                new KeySchemaElement {
                    AttributeName = "IssueId", KeyType = "RANGE" //Sort key
```

```
        }
    },
        Projection = new Projection
        {
            ProjectionType = "KEYS_ONLY"
        }
    };

    // DueDateIndex
    var dueDateIndex = new GlobalSecondaryIndex()
    {
        IndexName = "DueDateIndex",
        ProvisionedThroughput = ptIndex,
        KeySchema = {
        new KeySchemaElement {
            AttributeName = "DueDate",
            KeyType = "HASH" //Partition key
        }
    },
        Projection = new Projection
        {
            ProjectionType = "ALL"
        }
    };



    var createTableRequest = new CreateTableRequest
    {
        TableName = tableName,
        ProvisionedThroughput = new ProvisionedThroughput
        {
            ReadCapacityUnits = (long)1,
            WriteCapacityUnits = (long)1
        },
        AttributeDefinitions = attributeDefinitions,
        KeySchema = tableKeySchema,
        GlobalSecondaryIndexes = {
        createDateIndex, titleIndex, dueDateIndex
    }
    };

    Console.WriteLine("Creating table " + tableName + "...");
    client.CreateTable(createTableRequest);
```

```
            WaitUntilTableReady(tableName);
        }

        private static void LoadData()
        {
            Console.WriteLine("Loading data into table " + tableName + "...");

            // IssueId, Title,
            // Description,
            // CreateDate, LastUpdateDate, DueDate,
            // Priority, Status

            putItem("A-101", "Compilation error",
                "Can't compile Project X - bad version number. What does this mean?",
                "2013-11-01", "2013-11-02", "2013-11-10",
                1, "Assigned");

            putItem("A-102", "Can't read data file",
                "The main data file is missing, or the permissions are incorrect",
                "2013-11-01", "2013-11-04", "2013-11-30",
                2, "In progress");

            putItem("A-103", "Test failure",
                "Functional test of Project X produces errors",
                "2013-11-01", "2013-11-02", "2013-11-10",
                1, "In progress");

            putItem("A-104", "Compilation error",
                "Variable 'messageCount' was not initialized.",
                "2013-11-15", "2013-11-16", "2013-11-30",
                3, "Assigned");

            putItem("A-105", "Network issue",
                "Can't ping IP address 127.0.0.1. Please fix this.",
                "2013-11-15", "2013-11-16", "2013-11-19",
                5, "Assigned");
        }

        private static void putItem(
            String issueId, String title,
            String description,
            String createDate, String lastUpdateDate, String dueDate,
            Int32 priority, String status)
```

```
        {
            Dictionary<String, AttributeValue> item = new Dictionary<string,
AttributeValue>();

            item.Add("IssueId", new AttributeValue
            {
                S = issueId
            });
            item.Add("Title", new AttributeValue
            {
                S = title
            });
            item.Add("Description", new AttributeValue
            {
                S = description
            });
            item.Add("CreateDate", new AttributeValue
            {
                S = createDate
            });
            item.Add("LastUpdateDate", new AttributeValue
            {
                S = lastUpdateDate
            });
            item.Add("DueDate", new AttributeValue
            {
                S = dueDate
            });
            item.Add("Priority", new AttributeValue
            {
                N = priority.ToString()
            });
            item.Add("Status", new AttributeValue
            {
                S = status
            });

            try
            {
                client.PutItem(new PutItemRequest
                {
                    TableName = tableName,
                    Item = item
                });
```

```
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }

        private static void QueryIndex(string indexName)
        {
            Console.WriteLine
                ("\n*****************************************************\n");
            Console.WriteLine("Querying index " + indexName + "...");

            QueryRequest queryRequest = new QueryRequest
            {
                TableName = tableName,
                IndexName = indexName,
                ScanIndexForward = true
            };


            String keyConditionExpression;
            Dictionary<string, AttributeValue> expressionAttributeValues = new
Dictionary<string, AttributeValue>();

            if (indexName == "CreateDateIndex")
            {
                Console.WriteLine("Issues filed on 2013-11-01\n");

                keyConditionExpression = "CreateDate = :v_date and
begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_date", new AttributeValue
                {
                    S = "2013-11-01"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });
            }
            else if (indexName == "TitleIndex")
            {
                Console.WriteLine("Compilation errors\n");
```

```
                keyConditionExpression = "Title = :v_title and
begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_title", new AttributeValue
                {
                    S = "Compilation error"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else if (indexName == "DueDateIndex")
            {
                Console.WriteLine("Items that are due on 2013-11-30\n");

                keyConditionExpression = "DueDate = :v_date";
                expressionAttributeValues.Add(":v_date", new AttributeValue
                {
                    S = "2013-11-30"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else
            {
                Console.WriteLine("\nNo valid index name provided");
                return;
            }

            queryRequest.KeyConditionExpression = keyConditionExpression;
            queryRequest.ExpressionAttributeValues = expressionAttributeValues;

            var result = client.Query(queryRequest);
            var items = result.Items;
            foreach (var currentItem in items)
            {
                foreach (string attr in currentItem.Keys)
                {
                    if (attr == "Priority")
                    {
```

```
                    Console.WriteLine(attr + "---> " + currentItem[attr].N);
                }
                else
                {
                    Console.WriteLine(attr + "---> " + currentItem[attr].S);
                }
            }
            Console.WriteLine();
        }
    }

    private static void DeleteTable(string tableName)
    {
        Console.WriteLine("Deleting table " + tableName + "...");
        client.DeleteTable(new DeleteTableRequest
        {
            TableName = tableName
        });
        WaitForTableToBeDeleted(tableName);
    }

    private static void WaitUntilTableReady(string tableName)
    {
        string status = null;
        // Let us wait until table is created. Call DescribeTable.
        do
        {
            System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
            try
            {
                var res = client.DescribeTable(new DescribeTableRequest
                {
                    TableName = tableName
                });

                Console.WriteLine("Table name: {0}, status: {1}",
                          res.Table.TableName,
                          res.Table.TableStatus);
                status = res.Table.TableStatus;
            }
            catch (ResourceNotFoundException)
            {
                // DescribeTable is eventually consistent. So you might
                // get resource not found. So we handle the potential exception.
```

```
                }
            } while (status != "ACTIVE");
        }

        private static void WaitForTableToBeDeleted(string tableName)
        {
            bool tablePresent = true;

            while (tablePresent)
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                }
                catch (ResourceNotFoundException)
                {
                    tablePresent = false;
                }
            }
        }
    }
}
```

## AWS CLI를 사용하여 DynamoDB에서 글로벌 보조 인덱스 작업

AWS CLI를 사용하여 하나 이상의 글로벌 보조 인덱스가 포함된 Amazon DynamoDB 테이블을 만들고, 테이블의 인덱스를 설명하고, 인덱스를 사용하여 쿼리를 수행할 수 있습니다.

주제

- 글로벌 보조 인덱스가 있는 테이블 생성
- 기존 테이블에 글로벌 보조 인덱스 추가
- 글로벌 보조 인덱스가 있는 테이블 설명

- 글로벌 보조 인덱스 쿼리

### 글로벌 보조 인덱스가 있는 테이블 생성

글로벌 보조 인덱스는 테이블을 생성할 때 동시에 생성할 수 있습니다. 이렇게 하려면 `create-table` 파라미터를 사용하여 하나 이상의 글로벌 보조 인덱스 사양을 입력합니다. 다음 예제에서는 `GameTitleIndex`라는 글로벌 보조 인덱스가 있는 `GameScores`라는 테이블을 생성합니다. 기본 테이블은 파티션 키가 `UserId`이고 정렬 키가 `GameTitle`이므로 특정 게임의 개별 사용자 최고 점수를 효율적으로 찾을 수 있는 반면 GSI는 파티션 키가 `GameTitle`이고 정렬 키가 `TopScore`이므로 특정 게임의 전체 최고 점수를 빠르게 찾을 수 있습니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S \
                            AttributeName=GameTitle,AttributeType=S \
                            AttributeName=TopScore,AttributeType=N  \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                                {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                \"Projection\":{
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

DynamoDB에서 테이블을 만들고 테이블 상태가 `ACTIVE`로 설정될 때까지 기다려야 합니다. 그런 다음 테이블에 데이터 항목을 입력할 수 있습니다. describe table을 사용하여 테이블 생성 상태를 결정할 수 있습니다.

기존 테이블에 글로벌 보조 인덱스 추가

또한 글로벌 보조 인덱스를 테이블 생성 후 추가하거나 수정할 수도 있습니다. 이렇게 하려면 update-table 파라미터를 사용하여 하나 이상의 글로벌 보조 인덱스 사양을 입력합니다. 다음 예제에서는 앞의 예제와 같은 스키마를 사용하지만 테이블이 이미 생성되었고 나중에 GSI를 추가한다고 가정합니다.

```
aws dynamodb update-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=TopScore,AttributeType=N  \
    --global-secondary-index-updates \
        "[
            {
                \"Create\": {
                    \"IndexName\": \"GameTitleIndex\",
                    \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH
\"},
                                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                    \"Projection\":{
                        \"ProjectionType\":\"INCLUDE\",
                        \"NonKeyAttributes\":[\"UserId\"]
                    }
                }
            }
        ]"
```

글로벌 보조 인덱스가 있는 테이블 설명

테이블의 글로벌 보조 인덱스에 관한 자세한 내용은 describe-table 파라미터를 참조하세요. 각 인덱스에 대해 인덱스의 이름, 키 스키마 및 프로젝션된 속성에 액세스할 수 있습니다.

```
aws dynamodb describe-table --table-name GameScores
```

글로벌 보조 인덱스 쿼리

테이블을 query할 때와 거의 동일한 방식으로 글로벌 보조 인덱스에서 query 작업을 사용할 수 있습니다. 인덱스 이름, 인덱스 정렬 키의 쿼리 기준, 반환하려는 속성을 지정해야 합니다. 이 예제에서 인덱스는 GameTitleIndex이고 인덱스 정렬 키는 GameTitle입니다.

인덱스로 프로젝션된 속성만 반환됩니다. 키가 아닌 속성을 선택하도록 이 쿼리를 수정할 수도 있지만, 그렇게 하려면 비교적 많은 비용이 드는 테이블 가져오기 작업이 필요합니다. 테이블 가져오기에 대한 자세한 내용은 속성 프로젝션 단원을 참조하세요.

```
aws dynamodb query --table-name GameScores\
    --index-name GameTitleIndex \
    --key-condition-expression "GameTitle = :v_game" \
    --expression-attribute-values '{":v_game":{"S":"Alien Adventure"} }'
```

## DynamoDB의 로컬 보조 인덱스

일부 애플리케이션은 기본 테이블의 기본 키만 사용하여 데이터를 쿼리하지만, 대체 정렬 키가 유용한 상황이 있습니다. 애플리케이션에서 정렬 키를 선택할 수 있도록 Amazon DynamoDB 테이블에 하나 이상의 로컬 보조 인덱스를 생성하고 이러한 인덱스에 대해 `Query` 또는 `Scan` 요청을 실행할 수 있습니다.

주제

- 시나리오: 로컬 보조 인덱스 사용
- 속성 프로젝션
- 로컬 보조 인덱스 생성
- 로컬 보조 인덱스에서 데이터 읽기
- 항목 읽기 및 로컬 보조 인덱스
- 로컬 보조 인덱스에 대해 프로비저닝된 처리량 고려 사항
- 로컬 보조 인덱스에 대한 스토리지 고려 사항
- 로컬 보조 인덱스의 항목 컬렉션
- 로컬 보조 인덱스로 작업: Java
- 로컬 보조 인덱스로 작업: .NET
- DynamoDB AWS CLI의 로컬 보조 인덱스로 작업

### 시나리오: 로컬 보조 인덱스 사용

예를 들어 `Thread` 테이블을 고려할 수 있습니다. 이 테이블은 AWS 토론 포럼과 같은 애플리케이션에 유용합니다. 다음 그림은 테이블에서 항목을 구성하는 방식을 보여 줍니다. (일부 속성만 표시)

Thread

| ForumName | Subject | LastPostDateTime | Replies | |
|---|---|---|---|---|
| "S3" | "aaa" | "2015-03-15:17:24:31" | 12 | ... |
| "S3" | "bbb" | "2015-01-22:23:18:01" | 3 | ... |
| "S3" | "ccc" | "2015-02-31:13:14:21" | 4 | ... |
| "S3" | "ddd" | "2015-01-03:09:21:11" | 9 | ... |
| "EC2" | "yyy" | "2015-02-12:11:07:56" | 18 | ... |
| "EC2" | "zzz" | "2015-01-18:07:33:42" | 0 | ... |
| "RDS" | "rrr" | "2015-01-19:01:13:24" | 3 | ... |
| "RDS" | "sss" | "2015-03-11:06:53:00" | 11 | ... |
| "RDS" | "ttt" | "2015-10-22:12:19:44" | 5 | ... |
| ... | ... | ... | ... | |

DynamoDB는 파티션 키 값이 동일한 모든 항목을 연속적으로 저장합니다. 이 예에서는 특정 `ForumName`에 의해 `Query` 작업으로 해당 포럼의 모든 스레드를 즉시 찾을 수 있습니다. 동일한 파티션 키 값을 가진 항목 그룹 내에서는 항목이 정렬 키 값을 기준으로 정렬됩니다. 정렬 키(`Subject`)가 쿼리에도 제공된 경우 DynamoDB는 반환된 결과를 좁힐 수 있습니다. 예를 들어, "S3" 포럼에서 문자 "a"로 시작하는 `Subject`를 포함한 모든 스레드가 반환됩니다.

일부 요청은 더 복잡한 데이터 액세스 패턴이 필요할 수 있습니다. 예:

- 보기 및 회신 횟수가 가장 많은 포럼 스레드
- 특정 포럼에서 메시지 수가 가장 많은 스레드
- 특정 기간 동안 특정 포럼에 게시된 스레드 수

이러한 질문에 답변하려면 `Query` 작업으로 충분하지 않습니다. 대신, 전체 테이블을 `Scan`해야 합니다. 항목이 수백 개인 테이블은 많은 양의 프로비저닝 읽기 처리량을 소비하며 완료되는 데 오랜 시간이 소요됩니다.

하지만, `Replies` 또는 `LastPostDateTime` 같이 키가 아닌 속성에 하나 이상의 를 지정할 수 있습니다.

로컬 보조 인덱스는 지정된 파티션 키 값에 대해 대체 정렬 키를 유지합니다. 또한 로컬 보조 인덱스에는 기본 테이블에서 나온 일부 또는 모든 속성의 복사본이 포함되어 있습니다. 테이블을 생성할 때 로컬 보조 인덱스로 프로젝션되는 속성을 지정할 수 있습니다. 로컬 보조 인덱스의 데이터는 기본 테이블과 동일한 파티션 키를 기준으로 구성되지만, 다른 정렬 키를 사용합니다. 따라서 이와 같이 다른 차원

에서 데이터 항목을 효율적으로 액세스할 수 있습니다. 쿼리 또는 스캔 유연성을 높이려면 테이블당 최대 5개의 로컬 보조 인덱스를 만들 수 있습니다.

애플리케이션이 특정 포럼에서 최근 3개월간 게시된 모든 스레드를 찾아야 하는 경우를 가정하겠습니다. 로컬 보조 인덱스가 없을 경우 애플리케이션은 전체 `Thread` 테이블을 `Scan`하고 지정된 기간에 속하지 않는 게시물을 무시해야 합니다. 로컬 보조 인덱스가 있으면 `Query` 작업에서 정렬 키로 `LastPostDateTime`을 사용하여 데이터를 빠르게 찾을 수 있습니다.

다음 다이어그램은 `LastPostIndex`라는 로컬 보조 인덱스를 보여 줍니다.. 파티션 키는 `Thread` 테이블과 동일하지만 정렬 키는 `LastPostDateTime`입니다.

*LastPostIndex*

| *ForumName* | *LastPostDateTime* | *Subject* |
|---|---|---|
| *"S3"* | *"2015-01-03:09:21:11"* | *"ddd"* |
| *"S3"* | *"2015-01-22:23:18:01"* | *"bbb"* |
| *"S3"* | *"2015-02-31:13:14:21"* | *"ccc"* |
| *"S3"* | *"2015-03-15:17:24:31"* | *"aaa"* |
| *"EC2"* | *"2015-01-18:07:33:42"* | *"zzz"* |
| *"EC2"* | *"2015-02-12:11:07:56"* | *"yyy"* |
| *"RDS"* | *"2015-01-19:01:13:24"* | *"rrr"* |
| *"RDS"* | *"2015-02-22:12:19:44"* | *"ttt"* |
| *"RDS"* | *"2015-03-11:06:53:00"* | *"sss"* |
| ... | ... | ... |

모든 로컬 보조 인덱스는 다음 조건을 충족해야 합니다.

- 파티션 키는 기본 테이블의 키와 동일합니다.
- 정렬 키는 정확히 하나의 스칼라 속성으로 구성됩니다.
- 기본 테이블의 정렬 키가 인덱스로 프로젝션되며, 이 인덱스는 키가 아닌 속성으로 작동합니다.

이 예제에서는 파티션 키가 ForumName이고 로컬 보조 인덱스의 정렬 키는 LastPostDateTime입니다. 또한 기본 테이블의 정렬 키 값(이 예제의 경우 Subject)이 인덱스로 프로젝션되지만 인덱스 키의 부분이 아닙니다. 애플리케이션에 ForumName 및 LastPostDateTime을 기준으로 하는 목록이 필요할 경우 LastPostIndex에 Query 요청을 실행합니다. 쿼리 결과가 LastPostDateTime 기준으로 정렬되고 오름차순 또는 내림차순으로 반환할 수 있습니다. 또한 쿼리는 특정 기간 내 LastPostDateTime이 있는 항목만 반환하는 등의 키 조건을 적용할 수 있습니다.

모든 로컬 보조 인덱스에는 기본 테이블의 파티션 및 정렬 키가 자동적으로 포함되며, 키가 아닌 속성을 인덱스로 프로젝션할 수 있습니다(선택 사항). 인덱스를 쿼리할 경우 DynamoDB에서 이와 같이 프로젝션된 속성을 효율적으로 가져올 수 있습니다. 로컬 보조 인덱스를 쿼리하는 경우 쿼리에서는 인덱스로 프로젝션되지 않은 속성도 검색할 수 있습니다. DynamoDB는 기본 테이블에서 이러한 속성을 자동으로 가져오지만, 지연 시간이 길어지고 프로비저닝된 처리량 비용이 높아집니다.

모든 로컬 보조 인덱스에서 개별 파티션 키 값마다 최대 10GB의 데이터를 저장할 수 있습니다. 이 수치에는 기본 테이블과 인덱스에서 동일한 파티션 키 값을 가진 모든 항목이 포함됩니다. 자세한 내용은 로컬 보조 인덱스의 항목 컬렉션 단원을 참조하십시오.

## 속성 프로젝션

LastPostIndex에서 애플리케이션은 ForumName와 LastPostDateTime을 쿼리 기준으로 사용할 수 있었습니다. 그러나 추가 속성을 검색하려면, DynamoDB는 Thread 테이블에 대해 추가 읽기 작업을 수행해야 합니다. 이러한 추가 읽기를 가져오기(fetch)라고 하며 쿼리에 필요한 총 할당 처리량이 증가할 수 있습니다.

웹 페이지를 "S3"의 모든 스레드와 각 스레드에 대한 댓글 수 목록으로 채우고, 최근 댓글부터 시작하여 마지막 댓글 날짜/시간을 기준으로 정렬하려는 경우를 가정해 보세요. 이 목록을 채우려면 다음과 같은 속성이 필요합니다.

- Subject
- Replies
- LastPostDateTime

이 데이터를 쿼리하고 가져오기 작업을 회피하는 가장 효율적인 방법은 아래 그림과 같이 Replies 속성을 테이블에서 로컬 보조 인덱스로 프로젝션하는 것입니다.

*LastPostIndex*

| ForumName | LastPostDateTime | Subject | Replies |
|---|---|---|---|
| "S3" | "2015-01-03:09:21:11" | "ddd" | 9 |
| "S3" | "2015-01-22:23:18:01" | "bbb" | 3 |
| "S3" | "2015-02-31:13:14:21" | "ccc" | 4 |
| "S3" | "2015-03-15:17:24:31" | "aaa" | 12 |
| "EC2" | "2015-01-18:07:33:42" | "zzz" | 0 |
| "EC2" | "2015-02-12:11:07:56" | "yyy" | 18 |
| "RDS" | "2015-01-19:01:13:24" | "rrr" | 3 |
| "RDS" | "2015-02-22:12:19:44" | "ttt" | 5 |
| "RDS" | "2015-03-11:06:53:00" | "sss" | 11 |
| ... | ... | ... | ... |

프로젝션은 테이블에서 보조 인덱스로 복사되는 속성 집합입니다. 테이블의 파티션 키와 정렬 키는 항상 인덱스로 프로젝션되지만, 다른 속성을 프로젝션하여 애플리케이션의 쿼리 요건을 지원하는 것도 가능합니다. 따라서 인덱스에 쿼리를 실행할 때는 마치 속성이 자체 테이블에 저장되어 있는 것처럼 Amazon DynamoDB가 프로젝션의 모든 속성에 액세스할 수 있습니다.

보조 인덱스를 생성할 때 인덱스에 프로젝션될 속성을 지정해야 합니다. DynamoDB는 이를 위해 다음과 같은 세 가지 옵션을 제공합니다.

- KEYS_ONLY - 인덱스의 각 항목은 테이블 파티션 키 및 정렬 키 값, 그리고 인덱스 키 값으로만 구성됩니다. KEYS_ONLY 옵션은 보조 인덱스의 크기를 최소화합니다.
- INCLUDE – KEYS_ONLY에서 설명한 속성 외에도 지정하는 키가 아닌 다른 속성이 보조 인덱스에 포함됩니다.
- ALL – 보조 인덱스에 소스 테이블의 모든 속성이 포함됩니다. 모든 테이블 데이터가 인덱스에 복제되므로 ALL 프로젝션은 보조 인덱스의 크기를 최대화합니다.

이전 다이어그램에서는 키가 아닌 Replies 속성이 LastPostIndex로 프로젝션됩니다. 애플리케이션은 전체 Thread 테이블 대신에 LastPostIndex을 쿼리하여 Subject, Replies 및 LastPostDateTime로 웹 페이지를 채울 수 있습니다. 키가 아닌 다른 속성을 요청할 경우 DynamoDB는 Thread 테이블에서 이러한 속성을 가져와야 합니다.

애플리케이션의 관점에서, 기본 테이블에서 추가 속성을 가져오는 작업은 자동으로 수행되므로 애플리케이션 로직을 다시 쓸 필요는 없습니다. 하지만, 이와 같은 가져오기 작업을 실행할 경우 로컬 보조 인덱스를 이용하는 성능 상의 이점이 크게 감소할 수 있습니다.

속성을 로컬 보조 인덱스로 프로젝션할 경우에는 프로비저닝된 처리량 비용과 스토리지 비용 간 균형을 고려해야 합니다.

- 대기 시간이 가장 낮은 몇 개의 속성만 액세스해야 할 경우 해당 속성만 로컬 보조 인덱스로 프로젝션하는 방법을 고려해 볼 수 있습니다. 인덱스가 작을수록 스토리지 비용과 쓰기 비용이 절감됩니다. 간혹 가져와야 하는 속성이 있을 경우 할당 처리량의 비용이 이러한 속성의 장기 저장 비용보다 크게 증가할 수 있습니다.
- 애플리케이션이 키가 아닌 일부 속성에 빈번하게 액세스할 경우 해당 속성을 로컬 보조 인덱스로 프로젝션하는 방법을 고려해야 합니다. 로컬 보조 인덱스의 추가 스토리지 비용이 빈번한 테이블 스캔 수행으로 발생하는 비용보다 경제적입니다.
- 키가 아닌 속성 대부분에 자주 액세스해야 하는 경우 해당 속성이나 심지어 전체 기본 테이블을 글로벌 보조 인덱스로 프로젝션할 수 있습니다. 그러면 가져오기가 필요하지 않으므로 유연성이 극대화되고 프로비저닝된 처리량 소비가 최소화됩니다. 하지만, 모든 속성을 프로젝션하면 스토리지 비용이 최대 2배까지 증가할 수 있습니다.
- 애플리케이션이 테이블에 빈번하게 쿼리하지 않지만 테이블 데이터에 대해 많은 쓰기 또는 업데이트 작업을 수행해야 할 경우 KEYS_ONLY를 프로젝션할 수 있습니다. 이 경우 로컬 보조 인덱스는 크기는 최소화되지만 쿼리 작업에 필요할 경우 계속 사용할 수 있습니다.

## 로컬 보조 인덱스 생성

테이블에서 하나 이상의 로컬 보조 인덱스를 생성하려면 CreateTable 작업의 LocalSecondaryIndexes 파라미터를 사용합니다. 테이블이 생성되면 테이블에 로컬 보조 인덱스가 생성됩니다. 테이블을 삭제할 경우 해당 테이블의 로컬 보조 인덱스도 삭제됩니다.

키가 아닌 하나의 속성을 로컬 보조 인덱스의 정렬 키로 지정해야 합니다. 선택한 속성은 스칼라 String, Number 또는 Binary여야 합니다. 다른 스칼라 유형, 문서 유형 및 집합 유형은 허용되지 않습니다. 데이터 형식에 대한 전체 목록은 데이터 타입 단원을 참조하세요.

> **Important**
>
> 로컬 보조 인덱스가 있는 테이블의 경우 파티션 키 값당 10GB 크기 한도가 적용됩니다. 하나의 파티션 키 값의 총 크기가 10GB를 초과하지 않는 이상 로컬 보조 인덱스가 있는 테이블은 모든 수의 항목을 저장할 수 있습니다. 자세한 내용은 항목 컬렉션 크기 제한 단원을 참조하십시오.

모든 데이터 형식의 속성을 로컬 보조 인덱스로 프로젝션할 수 있습니다. 여기에는 스칼라, 문서, 집합이 포함됩니다. 데이터 형식에 대한 전체 목록은 데이터 타입 단원을 참조하세요.

## 로컬 보조 인덱스에서 데이터 읽기

`Query` 및 `Scan` 작업을 사용하여 로컬 보조 인덱스에서 항목을 가져올 수 있습니다. `GetItem` 및 `BatchGetItem` 작업은 로컬 보조 인덱스에 사용할 수 없습니다.

### 로컬 보조 인덱스 쿼리

DynamoDB 테이블에서 각 항목에 대한 파티션 키 값과 정렬 키 값의 조합은 고유해야 합니다. 하지만, 로컬 보조 인덱스에서 정렬 키 값은 특정 파티션 키 값에 대해 고유하지 않아도 됩니다. 로컬 보조 인덱스에 정렬 키 값이 같은 여러 항목이 있을 경우 `Query` 작업은 파티션 키 값이 같은 항목을 모두 반환합니다. 응답 시 일치하는 항목이 특정 순서로 반환되지 않습니다.

최종적으로 일관된 읽기 또는 강력히 일관된 읽기를 사용하여 로컬 보조 인덱스를 쿼리할 수 있습니다. 원하는 유형의 일관성을 지정하려면 `Query` 작업의 `ConsistentRead` 파라미터를 사용합니다. 로컬 보조 인덱스에서 강력히 일관된 읽기는 항상 업데이트된 최신 값을 반환합니다. 쿼리가 기본 테이블에서 추가 속성을 가져와야 하는 경우 해당 속성은 인덱스와 일관성을 유지합니다.

#### Example

특정 포럼의 토론 스레드에서 데이터를 요청하는 `Query`에서 다음과 같은 데이터가 반환된 경우를 가정해 보세요.

```
{
    "TableName": "Thread",
    "IndexName": "LastPostIndex",
    "ConsistentRead": false,
    "ProjectionExpression": "Subject, LastPostDateTime, Replies, Tags",
    "KeyConditionExpression":
        "ForumName = :v_forum and LastPostDateTime between :v_start and :v_end",
    "ExpressionAttributeValues": {
        ":v_start": {"S": "2015-08-31T00:00:00.000Z"},
```

```
        ":v_end": {"S": "2015-11-31T00:00:00.000Z"},
        ":v_forum": {"S": "EC2"}
    }
}
```

이 쿼리에서

- DynamoDB는 `LastPostIndex`에 액세스하고 `ForumName` 파티션 키를 사용하여 "EC2"의 인덱스 항목을 찾습니다. 이 키가 있는 모든 인덱스 항목은 빠른 검색을 위해 인접한 상태로 저장됩니다.
- 이 포럼 내에서 DynamoDB는 인덱스를 사용하여 지정된 `LastPostDateTime` 조건과 일치하는 키를 조회합니다.
- `Replies` 속성은 인덱스로 프로젝션되므로 DynamoDB는 프로비저닝된 처리량을 추가로 소비하지 않고 이 속성을 가져올 수 있습니다.
- `Tags` 속성은 인덱스로 프로젝션되지 않으므로 DynamoDB는 `Thread` 테이블에 액세스하여 이 속성을 가져와야 합니다.
- 결과가 반환되고, `LastPostDateTime`을 기준으로 정렬됩니다. 인덱스 항목이 파티션 키 값을 기준으로 정렬된 다음 정렬 키를 기준으로 정렬되고, `Query`는 저장된 순서로 인덱스 항목을 반환합니다. `ScanIndexForward` 파라미터를 사용하여 결과를 내림차순으로 정렬할 수 있습니다.

`Tags` 속성은 로컬 보조 인덱스로 프로젝션되지 않으므로 DynamoDB는 추가 읽기 용량 단위를 사용하여 이 속성을 기본 테이블에서 가져와야 합니다. 이 쿼리를 자주 실행해야 하는 경우에는 기본 테이블에서 가져오기가 되지 않도록 `Tags`를 `LastPostIndex`로 프로젝션해야 합니다. 그러나 가끔씩만 `Tags`에 액세스해야 하는 경우에는 인덱스에 `Tags`을 프로젝션하기 위한 추가 스토리지 비용을 지불할 가치가 없을 것입니다.

로컬 보조 인덱스 스캔

`Scan`을 사용하여 로컬 보조 인덱스에서 모든 데이터를 검색할 수 있습니다. 요청에 기본 테이블 이름과 인덱스 이름을 제공해야 합니다. DynamoDB는 `Scan`을 사용하여 인덱스에 있는 모든 데이터를 읽고 애플리케이션에 반환합니다. 또한 일부 데이터만 반환하고 나머지 데이터를 무시하도록 요청할 수 있습니다. 이와 같이 하려면 `Scan` API의 `FilterExpression` 파라미터를 사용합니다. 자세한 내용은 스캔에 대한 필터 표현식 단원을 참조하십시오.

## 항목 읽기 및 로컬 보조 인덱스

DynamoDB는 자동으로 모든 로컬 보조 인덱스를 관련 기본 테이블과 동기화 상태로 유지합니다. 애플리케이션이 인덱스에 직접 쓰기를 수행하는 경우는 없습니다. 하지만, DynamoDB가 이러한 인덱스를 유지하는 방식이 어떤 영향을 미치는지 이해하는 것이 중요합니다.

로컬 보조 인덱스를 생성할 때 인덱스의 정렬 키로 사용할 속성을 지정해야 합니다. 해당 속성의 데이터 형식을 지정합니다. 다시 말해, 기본 테이블에 항목을 쓸 때마다 항목이 인덱스 키 속성을 정의할 경우 해당 형식이 인덱스 키 스키마의 데이터 형식과 일치해야 함을 의미합니다. `LastPostIndex`의 경우, 인덱스의 `LastPostDateTime` 정렬 키가 `String` 데이터 유형으로 정의됩니다. `Thread` 테이블에 항목을 추가하고 `LastPostDateTime`(예: `Number`)에 다른 데이터 형식을 지정할 경우 DynamoDB에서 데이터 형식 불일치로 인해 `ValidationException`을 반환합니다.

기본 테이블의 항목과 로컬 보조 인덱스의 항목이 일대일 관계를 가질 필요는 없습니다. 하지만 실제로 이러한 동작은 많은 애플리케이션에서 유리할 수 있습니다.

로컬 보조 인덱스가 많은 테이블은 인덱스가 적은 테이블보다 쓰기 작업에 더 높은 비용이 발생합니다. 자세한 내용은 로컬 보조 인덱스에 대해 프로비저닝된 처리량 고려 사항 단원을 참조하십시오.

> **Important**
>
> 로컬 보조 인덱스가 있는 테이블의 경우 파티션 키 값당 10GB 크기 한도가 적용됩니다. 하나의 파티션 키 값의 총 크기가 10GB를 초과하지 않는 이상 로컬 보조 인텍스가 있는 테이블은 모든 수의 항목을 저장할 수 있습니다. 자세한 내용은 항목 컬렉션 크기 제한 단원을 참조하십시오.

## 로컬 보조 인덱스에 대해 프로비저닝된 처리량 고려 사항

DynamoDB에서 테이블을 생성할 때는 해당 테이블의 예상 워크로드를 위한 읽기 및 쓰기 용량 단위를 프로비저닝해야 합니다. 이러한 워크로드에는 테이블의 로컬 보조 인덱스에 대한 읽기 및 쓰기 작업이 포함됩니다.

프로비저닝된 처리 용량에 대한 현재 요율을 보려면 Amazon DynamoDB 요금을 참조하세요.

### 읽기 용량 단위

로컬 보조 인덱스를 쿼리할 경우 소비되는 읽기 용량 단위의 수는 데이터가 액세스되는 방식에 따라 달라집니다.

테이블 쿼리와 마찬가지로, 인덱스 쿼리는 `ConsistentRead`값에 따라 최종적으로 일관된 읽기 또는 강력한 일관된 읽기를 사용할 수 있습니다. 하나의 강력한 일관된 읽기(Strongly Consistent Read)는 하나의 읽기 용량 단위를 소비하고 하나의 최종적 일관된 읽기(Eventually Consistent Read)는 그 절반만 소비합니다. 따라서, 최종적 일관된 읽기(Eventually Consistent Read)를 선택할 경우 읽기 용량 단위 요금을 줄일 수 있습니다.

인덱스 키 및 프로젝션된 속성만 요구하는 인덱스 쿼리의 경우 DynamoDB는 테이블에 대한 쿼리와 같은 방식으로 프로비저닝된 읽기 작업을 계산합니다. 유일한 차이는 기본 테이블 항목의 크기가 아닌 인

덱스 항목의 크기를 기준으로 계산이 이루어진다는 점입니다. 읽기 용량 단위의 수는 반환된 모든 항목에서 프로젝션된 모든 속성 크기의 합계입니다. 그런 다음 결과가 다음 4KB 경계로 반올림됩니다. DynamoDB에서 프로비저닝된 처리량을 계산하는 방법에 대한 자세한 내용은 DynamoDB 프로비저닝된 용량 모드 단원을 참조하세요.

로컬 보조 인덱스로 프로젝션되지 않은 속성을 읽는 인덱스 쿼리의 경우 DynamoDB는 인덱스에서 프로젝션된 속성을 읽을 뿐만 아니라 기본 테이블에서 이러한 속성을 가져와야 합니다. 이와 같은 가져오기 작업은 `Query` 작업의 `Select` 또는 `ProjectionExpression` 파라미터에 프로젝션되지 않은 속성을 포함할 때 발생합니다. 가져오기를 수행하면 쿼리 응답이 더욱 지연되며 프로비저닝된 처리량 비용도 증가합니다. 위에서 설명한 로컬 보조 인덱스에서 읽기 이외에 가져온 모든 기본 테이블 항목의 읽기 용량 단위에도 요금이 청구됩니다. 이 요금은 요청된 속성뿐만 아니라 테이블에서 각각의 항목 전체를 읽는 데 부과되는 것입니다.

`Query` 작업에서 반환된 결과의 최대 크기는 1MB입니다. 여기에는 반환된 모든 항목의 속성 이름 및 값의 크기가 포함됩니다. 하지만 로컬 보조 인덱스에 대한 쿼리로 DynamoDB가 기본 테이블에서 항목 속성을 가져올 경우 결과에 있는 데이터의 최대 크기는 더 작을 수 있습니다. 이 경우 결과 크기는 다음의 합계입니다.

- 인덱스에서 일치하는 항목의 크기(다음 4KB로 반올림됨)
- 기본 테이블에서 일치하는 각 항목의 크기(각 항목이 개별적으로 다음 4KB로 반올림됨)

이 수식을 사용하면 쿼리 작업에서 반환한 결과의 최대 크기는 계속 1MB입니다.

예를 들어 각 항목의 크기가 300바이트인 테이블을 가정해 보겠습니다. 이 테이블에 로컬 보조 인덱스가 있지만 각 항목에서 200바이트만 인덱스로 프로젝션되었습니다. 이제 이 인덱스를 `Query`할 때 쿼리에서 각 항목의 테이블을 가져와야 하고 쿼리에서 4개 항목을 반환한다고 가정합니다. DynamoDB는 다음과 같이 합계를 계산합니다.

- 인덱스에서 일치하는 항목의 크기는 200바이트 × 4개 항목 = 800바이트이며, 이 값은 4KB로 반올림됩니다.
- 기본 테이블에서 일치하는 각 항목의 크기는 300바이트(4KB로 반올림됨) × 4개 항목 = 16KB입니다.

따라서 결과에 포함된 데이터의 총 크기는 20KB입니다.

### 쓰기 용량 단위

테이블에 항목을 추가, 업데이트 또는 삭제하고 로컬 보조 인덱스를 업데이트할 경우 테이블의 할당 쓰기 용량 단위가 소비됩니다. 쓰기에 프로비저닝된 총 처리량 비용은 테이블에 쓰기 작업으로 소비된 쓰기 용량 단위와 로컬 보조 인덱스를 업데이트하여 소비된 쓰기 용량 단위의 합계입니다.

로컬 보조 인덱스에 항목을 쓰는 비용은 몇 가지 요인에 따라 달라집니다.

- 인덱싱된 속성을 정의하는 테이블에 새 항목을 쓰거나 이전에 정의되지 않은 인덱싱된 속성을 정의하도록 기존 항목을 업데이트하는 경우 항목을 인덱스에 넣으려면 1번의 쓰기 작업이 필요합니다.
- 테이블을 업데이트하여 인덱스 키 속성 값이 A에서 B로 변경될 경우 두 번의 쓰기, 즉, 인덱스에서 이전 항목을 삭제하는 쓰기와 새 항목을 인덱스에 추가하는 쓰기가 필요합니다.
- 항목이 인덱스에 있지만 테이블 쓰기로 인해 인덱스 속성이 삭제된 경우 인덱스에서 기존 항목 프로젝션을 삭제하는 한 번의 쓰기가 필요합니다.
- 항목이 업데이트되기 전후에 인덱스에 항목이 없을 경우 인덱스에 추가 쓰기 비용이 발생하지 않습니다.

이러한 모든 요인은 인덱스에 있는 각 항목의 크기가 쓰기 용량 단위를 계산하기 위한 1KB 항목 크기보다 작거나 같은 경우를 가정합니다. 이보다 큰 인덱스 항목에서는 추가 쓰기 용량 단위가 필요합니다. 쿼리에서 어떤 속성을 반환해야 하는지 고려하고 해당 속성만 인덱스에 프로젝션함으로써 쓰기 비용을 최소화할 수 있습니다.

## 로컬 보조 인덱스에 대한 스토리지 고려 사항

애플리케이션에서 테이블에 항목을 쓰는 경우 DynamoDB는 해당 속성이 표시되어야 하는 모든 로컬 보조 인덱스에 올바른 속성 하위 집합을 자동으로 복사합니다. AWS 계정에는 기본 테이블의 항목 스토리지 비용 및 해당 테이블에 있는 로컬 보조 인덱스의 속성 스토리지 비용이 청구됩니다.

인덱스 항목에서 사용하는 공간의 양은 다음의 합계입니다.

- 기본 테이블 기본 키(파티션 및 정렬 키)의 크기(바이트)
- 인덱스 키 속성의 크기(바이트)
- 프로젝션된 속성(있는 경우)의 크기(바이트)
- 인덱스 항목당 오버헤드의 100바이트

로컬 보조 인덱스의 스토리지 요구 사항을 추정하려면 인덱스의 평균 항목 크기를 추정한 다음 인덱스에 있는 항목의 수를 곱합니다.

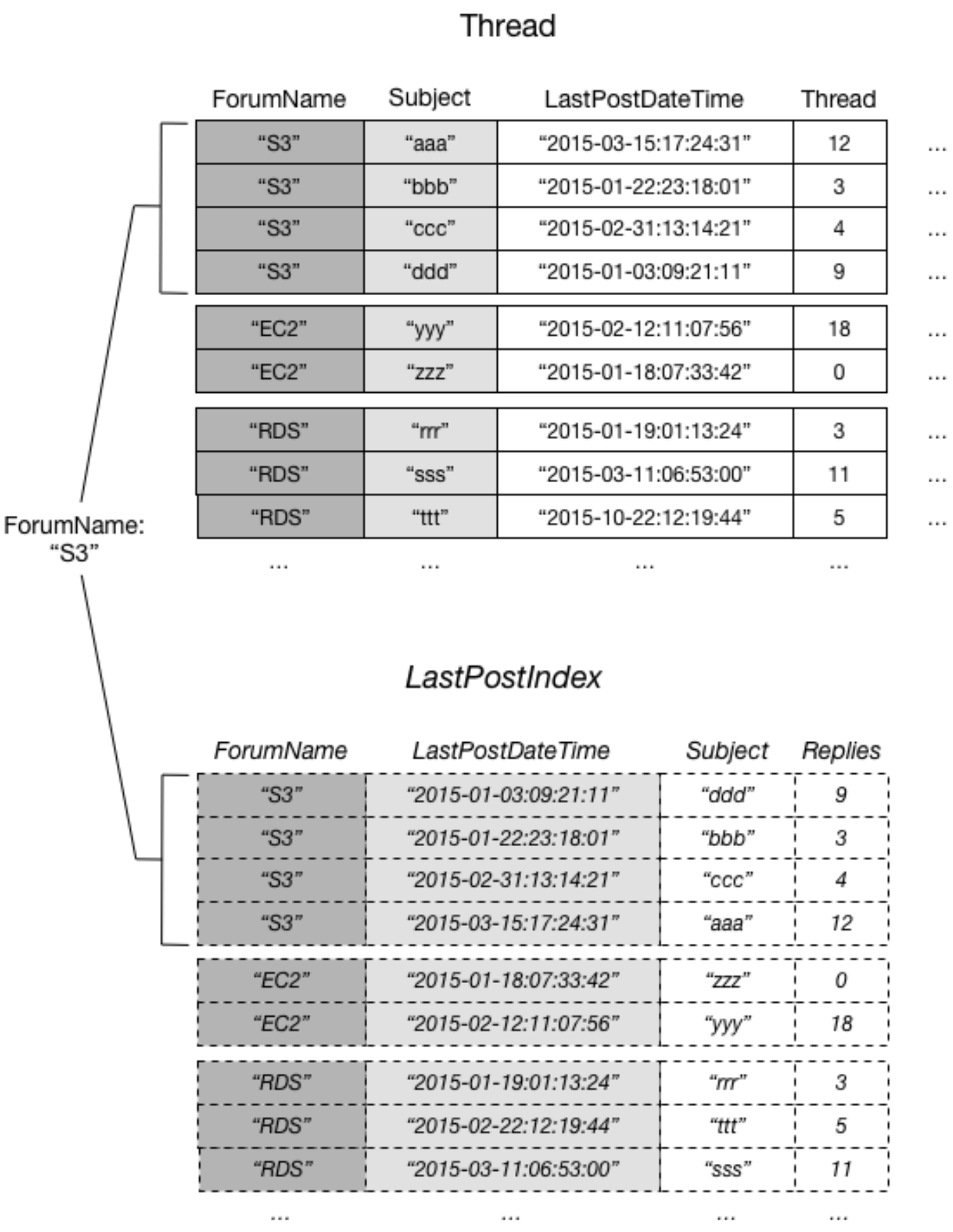
테이블에 특정 속성이 정의되지 않은 항목이 포함되어 있지만 해당 속성이 인덱스 정렬 키로 정의된 경우 DynamoDB에서 해당 항목의 데이터를 인덱스에 쓰지 않습니다.

## 로컬 보조 인덱스의 항목 컬렉션

> **Note**
>
> 다음 단원은 로컬 보조 인덱스가 있는 테이블에만 적용됩니다.

DynamoDB에서 항목 컬렉션은 테이블에서 동일한 파티션 키 값과 해당하는 모든 로컬 보조 인덱스가 있는 항목의 그룹입니다. 이 단원 전반에 사용된 예제에서 `Thread` 테이블의 파티션 키는 `ForumName`이며 `LastPostIndex`의 파티션 키도 `ForumName`입니다. 동일한 `ForumName`을 가진 모든 테이블과 인덱스 항목은 동일한 항목 컬렉션에 포함됩니다. 예를 들어 `Thread` 테이블과 `LastPostIndex` 로컬 보조 인덱스에는 포럼 `EC2`에 대한 한 가지 항목 컬렉션과 포럼 `RDS`에 대한 다른 항목 컬렉션이 있습니다.

다음 그림은 포럼 `S3`의 항목 컬렉션을 보여 줍니다.


Thread
ForumName
Subject
LastPostDateTime
Thread
"S3"
"aaa"
"2015-03-15:17:24:31"
12
"S3"
"bbb"
"2015-01-22:23:18:01"
3
"S3"
"ccc"
"2015-02-31:13:14:21"
4
"S3"
"ddd"
"2015-01-03:09:21:11"
9
"EC2"
"yyy"
"2015-02-12:11:07:56"
18
"EC2"
"zzz"
"2015-01-18:07:33:42"
0
"RDS"
"rrr"
"2015-01-19:01:13:24"
3
"RDS"
"sss"
"2015-03-11:06:53:00"
11
"RDS"
"ttt"
"2015-10-22:12:19:44"
5
ForumName:
"S3"
LastPostIndex
ForumName
LastPostDateTime
Subject
Replies
"S3"
"2015-01-03:09:21:11"
"ddd"
9
"S3"
"2015-01-22:23:18:01"
"bbb"
3
"S3"
"2015-02-31:13:14:21"
"ccc"
4
"S3"
"2015-03-15:17:24:31"
"aaa"
12
"EC2"
"2015-01-18:07:33:42"
"zzz"
0
"EC2"
"2015-02-12:11:07:56"
"yyy"
18
"RDS"
"2015-01-19:01:13:24"
"rrr"
3
"RDS"
"2015-02-22:12:19:44"
"ttt"
5
"RDS"
"2015-03-11:06:53:00"
"sss"
11

이 그림에서 항목 컬렉션은 Thread 및 LastPostIndex의 모든 항목으로 구성되어 있으며, 여기에서 ForumName 파티션 키 값은 "S3"입니다. 테이블에 다른 로컬 보조 인덱스가 있을 경우 해당 인덱스에서 ForumName이 "S3"인 모든 항목도 항목 컬렉션에 포함됩니다.

DynamoDB에서 다음 작업 중 하나를 사용하여 항목 컬렉션에 대한 정보를 반환할 수 있습니다.

- BatchWriteItem
- DeleteItem
- PutItem
- UpdateItem
- TransactWriteItems

각 작업은 ReturnItemCollectionMetrics 파라미터를 지원합니다. 이 파라미터를 SIZE로 설정할 경우 인덱스에 있는 각 항목 컬렉션의 크기에 대한 정보를 볼 수 있습니다.

Example

다음은ReturnItemCollectionMetrics를 SIZE로 설정한 상태에서 Thread 테이블에서 UpdateItem 작업을 실행하는 출력의 예제입니다. 업데이트된 항목에 ForumName 값 "EC2"가 있으므로 출력에 항목 컬렉션에 대한 정보가 포함됩니다.

```
{
    ItemCollectionMetrics: {
        ItemCollectionKey: {
            ForumName: "EC2"
        },
        SizeEstimateRangeGB: [0.0, 1.0]
    }
}
```

SizeEstimateRangeGB 객체는 이 항목 컬렉션의 크기가 0~1GB임을 나타냅니다. DynamoDB는 이 크기 추정 값을 주기적으로 업데이트하므로 다음에 항목이 수정되면 숫자가 달라질 수 있습니다.

항목 컬렉션 크기 제한

로컬 보조 인덱스가 하나 이상 포함된 테이블의 항목 컬렉션의 최대 크기는 10GB입니다. 이는 로컬 보조 인덱스가 없는 테이블의 항목 컬렉션에는 적용되지 않으며, 글로벌 보조 인덱스의 항목 컬렉션에도 적용되지 않습니다. 로컬 보조 인덱스가 하나 이상 있는 테이블만 영향을 받습니다.

항목 컬렉션이 10GB 한도를 초과할 경우 DynamoDB에서 `ItemCollectionSizeLimitExceededException`을 반환하며 항목 컬렉션에 더 이상 항목을 추가하거나 항목 컬렉션에 있는 항목의 크기를 늘릴 수 없습니다. (항목 컬렉션의 크기를 줄이는 읽기 및 쓰기 작업은 계속 할 수 있습니다.) 다른 항목 컬렉션에는 계속 항목을 추가할 수 있습니다.

항목 컬렉션의 크기를 줄이려면 다음 중 하나를 수행합니다.

- 해당 파티션 키 값이 있는 불필요한 항목을 삭제합니다. 기본 테이블에서 이러한 항목을 삭제하면 DynamoDB도 동일한 파티션 키 값을 가진 인덱스 항목을 제거합니다.
- 속성을 제거하거나 속성 크기를 줄여 항목을 업데이트합니다. 이러한 속성이 로컬 보조 인덱스로 프로젝션될 경우 DynamoDB에서도 해당 인덱스 항목의 크기를 줄입니다.
- 동일한 파티션 키와 정렬 키로 새 테이블을 만든 다음 기존 테이블의 항목을 새 테이블로 이동합니다. 이 방법은 테이블에 자주 액세스하지 않는 과거 데이터가 있을 경우 효과적입니다. 이 과거 데이터를 Amazon Simple Storage Service(Amazon S3)에 보관하는 방법도 고려할 수 있습니다.

항목 컬렉션의 총 크기가 10GB 미만으로 감소하면 동일한 파티션 키 값을 가진 항목을 다시 추가할 수 있습니다.

항목 컬렉션의 크기를 모니터링하기 위해 애플리케이션을 활용하는 것이 가장 좋습니다. `BatchWriteItem`, `DeleteItem`, `PutItem` 또는 `UpdateItem`을 사용할 때마다 `ReturnItemCollectionMetrics` 파라미터를 `SIZE`로 설정하는 것이 그중 한 가지 방법입니다. 애플리케이션은 출력에 있는 `ReturnItemCollectionMetrics` 객체를 조사하고 항목 컬렉션이 사용자 정의 한도(예: 8GB)를 초과할 때마다 오류 메시지를 기록해야 합니다. 한도를 10GB보다 작게 설정할 경우 이른 단계에서 경고를 받을 수 있으므로 항목 컬렉션이 한도에 가까워지면 미리 조치를 수행할 수 있습니다.

항목 컬렉션 및 파티션

로컬 보조 인덱스가 하나 이상 포함된 테이블에서 각 항목 컬렉션은 한 파티션에 저장됩니다. 이러한 항목 컬렉션의 전체 크기는 해당 파티션의 용량(10GB)으로 제한됩니다. 데이터 모델에 크기 제한이 없는 항목 컬렉션이 포함되어 있거나 향후 일부 항목 컬렉션이 10GB를 초과할 것으로 합리적으로 예상되는 애플리케이션의 경우 대신 글로벌 보조 인덱스를 사용하는 것이 좋습니다.

테이블 데이터가 각각의 파티션 키 값에서 고르게 분산되도록 애플리케이션을 설계해야 합니다. 로컬 보조 인덱스가 있는 테이블의 경우 애플리케이션이 단일 파티션의 단일 항목 컬렉션 내에 읽기 및 쓰기 작업의 "핫스폿"을 만들지 않아야 합니다.

# 로컬 보조 인덱스로 작업: Java

AWS SDK for Java Document API를 사용하여 하나 이상의 로컬 보조 인덱스가 포함된 Amazon DynamoDB 테이블을 만들고, 테이블의 인덱스를 설명하고, 인덱스를 사용하여 쿼리를 수행할 수 있습니다.

다음은 AWS SDK for Java Document API를 사용하여 테이블 작업을 할 때 따라야 할 공통 단계입니다.

1. DynamoDB 클래스의 인스턴스를 만듭니다.
2. 해당하는 요청 객체를 만들어 작업의 필수 및 선택적 파라미터를 제공합니다.
3. 이전 단계에서 만든 클라이언트가 제공한 적절한 메서드를 호출합니다.

주제

- 로컬 보조 인덱스가 있는 테이블 생성
- 로컬 보조 인덱스가 있는 테이블 설명
- 로컬 보조 인덱스 쿼리
- 예: Java 문서 API를 사용하는 로컬 보조 인덱스

## 로컬 보조 인덱스가 있는 테이블 생성

로컬 보조 인덱스는 테이블을 만들 때 동시에 만들어야 합니다. 이렇게 하려면 `createTable` 메서드를 사용하여 하나 이상의 로컬 보조 인덱스 사양을 입력합니다. 다음 Java 코드 예제는 보유한 음악 파일에 있는 곡의 정보를 담은 테이블을 만듭니다. 파티션 키는 `Artist`이고 정렬 키는 `SongTitle`입니다. 보조 인덱스인 `AlbumTitleIndex`는 앨범 제목을 사용해 쿼리를 쉽게 수행하는 데 사용합니다.

다음은 DynamoDB Document API를 사용하여 로컬 보조 인덱스가 있는 테이블을 생성하는 단계입니다.

1. DynamoDB 클래스의 인스턴스를 만듭니다.
2. `CreateTableRequest` 클래스 인스턴스를 만들어 요청 정보를 입력합니다.

   이때 입력해야 하는 정보는 테이블 이름, 기본 키, 그리고 프로비저닝된 처리량 값입니다. 로컬 보조 인덱스의 경우 인덱스 이름, 인덱스 정렬 키의 이름 및 데이터 형식, 인덱스의 키 스키마, 속성 프로젝션을 입력해야 합니다.
3. 요청 객체를 파라미터로 입력하여 `createTable` 메서드를 호출합니다.

다음 Java 코드 예는 앞의 단계를 보여줍니다. 이 코드는 AlbumTitle 속성에 보조 인덱스가 있는 테이블(Music)을 생성합니다. 인덱스에 프로젝션되는 속성은 테이블 파티션 키 및 정렬 키와 인덱스 정렬 키뿐입니다.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName);

//ProvisionedThroughput
createTableRequest.setProvisionedThroughput(new
 ProvisionedThroughput().withReadCapacityUnits((long)5).withWriteCapacityUnits((long)5));

//AttributeDefinitions
ArrayList<AttributeDefinition> attributeDefinitions= new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Artist").withAttributeType("S"));
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("SongTitle").withAttributeType("S"));
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("AlbumTitle").withAttributeType("S"));

createTableRequest.setAttributeDefinitions(attributeDefinitions);

//KeySchema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("SongTitle").withKeyType(KeyType.RANGE));  //Sort
 key

createTableRequest.setKeySchema(tableKeySchema);

ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
```

```
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("AlbumTitle").withKeyType(KeyType.RANGE));  //
Sort key

Projection projection = new Projection().withProjectionType(ProjectionType.INCLUDE);
ArrayList<String> nonKeyAttributes = new ArrayList<String>();
nonKeyAttributes.add("Genre");
nonKeyAttributes.add("Year");
projection.setNonKeyAttributes(nonKeyAttributes);

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()

 .withIndexName("AlbumTitleIndex").withKeySchema(indexKeySchema).withProjection(projection);

ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
 ArrayList<LocalSecondaryIndex>();
localSecondaryIndexes.add(localSecondaryIndex);
createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);

Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

DynamoDB에서 테이블을 만들고 테이블 상태가 ACTIVE로 설정될 때까지 기다려야 합니다. 그런 다음 테이블에 데이터 항목을 입력할 수 있습니다.

로컬 보조 인덱스가 있는 테이블 설명

테이블의 로컬 보조 인덱스에 관한 자세한 내용은 `describeTable` 메서드를 참조하세요. 각 인덱스에 대해 인덱스의 이름, 키 스키마 및 프로젝션된 속성에 액세스할 수 있습니다.

다음은 AWS SDK for Java Document API를 사용하여 테이블의 로컬 보조 인덱스 정보에 액세스하는 단계입니다.

1. `DynamoDB` 클래스의 인스턴스를 만듭니다.
2. `Table` 클래스의 인스턴스를 만듭니다. 테이블 이름을 입력해야 합니다.
3. `Table` 객체의 `describeTable` 메서드를 호출합니다.

다음 Java 코드 예는 앞의 단계를 보여줍니다.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
```

```
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

Table table = dynamoDB.getTable(tableName);

TableDescription tableDescription = table.describe();

List<LocalSecondaryIndexDescription> localSecondaryIndexes
    = tableDescription.getLocalSecondaryIndexes();

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.

Iterator<LocalSecondaryIndexDescription> lsiIter = localSecondaryIndexes.iterator();
while (lsiIter.hasNext()) {

    LocalSecondaryIndexDescription lsiDescription = lsiIter.next();
    System.out.println("Info for index " + lsiDescription.getIndexName() + ":");
    Iterator<KeySchemaElement> kseIter = lsiDescription.getKeySchema().iterator();
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = lsiDescription.getProjection();
    System.out.println("\tThe projection type is: " + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: " +
 projection.getNonKeyAttributes());
    }
}
```

로컬 보조 인덱스 쿼리

테이블을 Query할 때와 거의 동일한 방식으로 로컬 보조 인덱스에서 Query 작업을 사용할 수 있습니다. 인덱스 이름, 인덱스 정렬 키의 쿼리 기준, 반환하려는 속성을 지정해야 합니다. 이 예제에서 인덱스는 AlbumTitleIndex이고 인덱스 정렬 키는 AlbumTitle입니다.

인덱스로 프로젝션된 속성만 반환됩니다. 키가 아닌 속성을 선택하도록 이 쿼리를 수정할 수도 있지만, 그렇게 하려면 비교적 많은 비용이 드는 테이블 가져오기 작업이 필요합니다. 테이블 가져오기에 대한 자세한 내용은 속성 프로젝션 단원을 참조하세요.

다음은 AWS SDK for Java Document API를 사용하여 로컬 보조 인덱스를 쿼리하는 단계입니다.

1. DynamoDB 클래스의 인스턴스를 만듭니다.
2. Table 클래스의 인스턴스를 만듭니다. 테이블 이름을 입력해야 합니다.
3. Index 클래스의 인스턴스를 만듭니다. 인덱스 이름을 입력해야 합니다.
4. Index 클래스의 query 메서드를 호출합니다.

다음 Java 코드 예는 앞의 단계를 보여줍니다.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

Table table = dynamoDB.getTable(tableName);
Index index = table.getIndex("AlbumTitleIndex");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Artist = :v_artist and AlbumTitle = :v_title")
    .withValueMap(new ValueMap()
        .withString(":v_artist", "Acme Band")
        .withString(":v_title", "Songs About Life"));

ItemCollection<QueryOutcome> items = index.query(spec);

Iterator<Item> itemsIter = items.iterator();

while (itemsIter.hasNext()) {
    Item item = itemsIter.next();
    System.out.println(item.toJSONPretty());
}
```

예: Java 문서 API를 사용하는 로컬 보조 인덱스

다음 Java 코드 예제는 Amazon DynamoDB에서 로컬 보조 인덱스로 작업하는 방법을 보여 줍니다. 예를 들어 파티션 키가 CustomerId이고 정렬 키가 OrderId인 CustomerOrders라는 테이블을 만들 수 있습니다. 이 테이블에는 두 개의 로컬 보조 인덱스가 있습니다.

- OrderCreationDateIndex - 정렬 키는 OrderCreationDate이며 다음 속성이 인덱스로 프로젝션됩니다.

  - ProductCategory
  - ProductName
  - OrderStatus
  - ShipmentTrackingId
- IsOpenIndex - 정렬 키는 IsOpen이며 모든 테이블 속성이 인덱스로 프로젝션됩니다.

CustomerOrders 테이블이 생성되면 프로그램에서 고객 주문을 나타내는 데이터와 함께 테이블을 로드합니다. 그런 다음 로컬 보조 인덱스를 사용하여 데이터를 쿼리합니다. 마지막으로 프로그램에서 CustomerOrders 테이블을 삭제합니다.

다음 샘플을 테스트하기 위한 단계별 지침은 Java 코드 예 섹션을 참조하세요.

Example

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.PutItemOutcome;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.LocalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ReturnConsumedCapacity;
import com.amazonaws.services.dynamodbv2.model.Select;
```

```
public class DocumentAPILocalSecondaryIndexExample {

        static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        static DynamoDB dynamoDB = new DynamoDB(client);

        public static String tableName = "CustomerOrders";

        public static void main(String[] args) throws Exception {

                createTable();
                loadData();

                query(null);
                query("IsOpenIndex");
                query("OrderCreationDateIndex");

                deleteTable(tableName);

        }

        public static void createTable() {

                CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName)
                                .withProvisionedThroughput(
                                                new
 ProvisionedThroughput().withReadCapacityUnits((long) 1)

 .withWriteCapacityUnits((long) 1));

                // Attribute definitions for table partition and sort keys
                ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();
                attributeDefinitions
                                .add(new
 AttributeDefinition().withAttributeName("CustomerId").withAttributeType("S"));
                attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("OrderId").withAttributeType("N"));

                // Attribute definition for index primary key attributes
                attributeDefinitions
                                .add(new
 AttributeDefinition().withAttributeName("OrderCreationDate")
```

```
                                                .withAttributeType("N"));
                attributeDefinitions.add(new
AttributeDefinition().withAttributeName("IsOpen").withAttributeType("N"));

                createTableRequest.setAttributeDefinitions(attributeDefinitions);

                // Key schema for table
                ArrayList<KeySchemaElement> tableKeySchema = new
ArrayList<KeySchemaElement>();
                tableKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                                // key
                tableKeySchema.add(new
KeySchemaElement().withAttributeName("OrderId").withKeyType(KeyType.RANGE)); // Sort

                               // key

                createTableRequest.setKeySchema(tableKeySchema);

                ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();

                // OrderCreationDateIndex
                LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
                                .withIndexName("OrderCreationDateIndex");

                // Key schema for OrderCreationDateIndex
                ArrayList<KeySchemaElement> indexKeySchema = new
ArrayList<KeySchemaElement>();
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                               // key
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("OrderCreationDate")
                                .withKeyType(KeyType.RANGE)); // Sort
                                                              // key

                orderCreationDateIndex.setKeySchema(indexKeySchema);

                // Projection (with list of projected attributes) for
```

```
                // OrderCreationDateIndex
                Projection projection = new
Projection().withProjectionType(ProjectionType.INCLUDE);
                ArrayList<String> nonKeyAttributes = new ArrayList<String>();
                nonKeyAttributes.add("ProductCategory");
                nonKeyAttributes.add("ProductName");
                projection.setNonKeyAttributes(nonKeyAttributes);

                orderCreationDateIndex.setProjection(projection);

                localSecondaryIndexes.add(orderCreationDateIndex);

                // IsOpenIndex
                LocalSecondaryIndex isOpenIndex = new
LocalSecondaryIndex().withIndexName("IsOpenIndex");

                // Key schema for IsOpenIndex
                indexKeySchema = new ArrayList<KeySchemaElement>();
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                                // key
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("IsOpen").withKeyType(KeyType.RANGE)); // Sort

                            // key

                // Projection (all attributes) for IsOpenIndex
                projection = new Projection().withProjectionType(ProjectionType.ALL);

                isOpenIndex.setKeySchema(indexKeySchema);
                isOpenIndex.setProjection(projection);

                localSecondaryIndexes.add(isOpenIndex);

                // Add index definitions to CreateTable request
                createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);

                System.out.println("Creating table " + tableName + "...");
                System.out.println(dynamoDB.createTable(createTableRequest));

                // Wait for table to become active
```

```
                System.out.println("Waiting for " + tableName + " to become
ACTIVE...");
                try {
                        Table table = dynamoDB.getTable(tableName);
                        table.waitForActive();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        public static void query(String indexName) {

                Table table = dynamoDB.getTable(tableName);


System.out.println("\n***********************************************************\n");
                System.out.println("Querying table " + tableName + "...");

                QuerySpec querySpec = new
QuerySpec().withConsistentRead(true).withScanIndexForward(true)

.withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

                if (indexName == "IsOpenIndex") {

                        System.out.println("\nUsing index: '" + indexName + "': Bob's
orders that are open.");
                        System.out.println("Only a user-specified list of attributes
are returned\n");
                        Index index = table.getIndex(indexName);

                        querySpec.withKeyConditionExpression("CustomerId = :v_custid
and IsOpen = :v_isopen")
                                        .withValueMap(new
ValueMap().withString(":v_custid", "bob@example.com")
                                                        .withNumber(":v_isopen", 1));

                        querySpec.withProjectionExpression(
                                        "OrderCreationDate, ProductCategory,
ProductName, OrderStatus");

                        ItemCollection<QueryOutcome> items = index.query(querySpec);
                        Iterator<Item> iterator = items.iterator();
```

```
                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                } else if (indexName == "OrderCreationDateIndex") {
                        System.out.println("\nUsing index: '" + indexName
                                        + "': Bob's orders that were placed after
 01/31/2015.");
                        System.out.println("Only the projected attributes are returned
\n");
                        Index index = table.getIndex(indexName);

                        querySpec.withKeyConditionExpression(
                                        "CustomerId = :v_custid and OrderCreationDate
 >= :v_orddate")
                                        .withValueMap(
                                                        new
 ValueMap().withString(":v_custid", "bob@example.com")

 .withNumber(":v_orddate",

 20150131));

                        querySpec.withSelect(Select.ALL_PROJECTED_ATTRIBUTES);

                        ItemCollection<QueryOutcome> items = index.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                } else {
                        System.out.println("\nNo index: All of Bob's orders, by
 OrderId:\n");

                        querySpec.withKeyConditionExpression("CustomerId = :v_custid")
                                        .withValueMap(new
 ValueMap().withString(":v_custid", "bob@example.com"));
```

```
                        ItemCollection<QueryOutcome> items = table.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                }

        }

        public static void deleteTable(String tableName) {

                Table table = dynamoDB.getTable(tableName);
                System.out.println("Deleting table " + tableName + "...");
                table.delete();

                // Wait for table to be deleted
                System.out.println("Waiting for " + tableName + " to be deleted...");
                try {
                        table.waitForDelete();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        public static void loadData() {

                Table table = dynamoDB.getTable(tableName);

                System.out.println("Loading data into table " + tableName + "...");

                Item item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 1)
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150101)
                                .withString("ProductCategory", "Book")
                                .withString("ProductName", "The Great Outdoors")
                                .withString("OrderStatus", "PACKING ITEMS");
                // no ShipmentTrackingId attribute

                PutItemOutcome putItemOutcome = table.putItem(item);
```

```
                item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 2)
                                 .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150221)
                                 .withString("ProductCategory", "Bike")
                                 .withString("ProductName", "Super Mountain")
                                 .withString("OrderStatus", "ORDER RECEIVED");
                // no ShipmentTrackingId attribute

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 3)
                                 // no IsOpen attribute
                                 .withNumber("OrderCreationDate",
20150304).withString("ProductCategory", "Music")
                                 .withString("ProductName", "A Quiet
Interlude").withString("OrderStatus", "IN TRANSIT")
                                 .withString("ShipmentTrackingId", "176493");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 1)
                                 // no IsOpen attribute
                                 .withNumber("OrderCreationDate",
20150111).withString("ProductCategory", "Movie")
                                 .withString("ProductName", "Calm Before The Storm")
                                 .withString("OrderStatus", "SHIPPING DELAY")
                                 .withString("ShipmentTrackingId", "859323");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 2)
                                 // no IsOpen attribute
                                 .withNumber("OrderCreationDate",
20150124).withString("ProductCategory", "Music")
                                 .withString("ProductName", "E-Z
Listening").withString("OrderStatus", "DELIVERED")
                                 .withString("ShipmentTrackingId", "756943");

                putItemOutcome = table.putItem(item);
```

```
                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 3)
                                 // no IsOpen attribute
                                 .withNumber("OrderCreationDate",
20150221).withString("ProductCategory", "Music")
                                 .withString("ProductName", "Symphony
9").withString("OrderStatus", "DELIVERED")
                                 .withString("ShipmentTrackingId", "645193");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 4)
                                 .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150222)
                                 .withString("ProductCategory", "Hardware")
                                 .withString("ProductName", "Extra Heavy Hammer")
                                 .withString("OrderStatus", "PACKING ITEMS");
                // no ShipmentTrackingId attribute

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 5)
                                 /* no IsOpen attribute */
                                 .withNumber("OrderCreationDate",
20150309).withString("ProductCategory", "Book")
                                 .withString("ProductName", "How To
Cook").withString("OrderStatus", "IN TRANSIT")
                                 .withString("ShipmentTrackingId", "440185");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 6)
                                 // no IsOpen attribute
                                 .withNumber("OrderCreationDate",
20150318).withString("ProductCategory", "Luggage")
                                 .withString("ProductName", "Really Big
Suitcase").withString("OrderStatus", "DELIVERED")
                                 .withString("ShipmentTrackingId", "893927");

                putItemOutcome = table.putItem(item);
```

```
                item = new Item().withPrimaryKey("CustomerId",
 "bob@example.com").withNumber("OrderId", 7)
                                 /* no IsOpen attribute */
                                 .withNumber("OrderCreationDate",
 20150324).withString("ProductCategory", "Golf")
                                 .withString("ProductName", "PGA Pro
 II").withString("OrderStatus", "OUT FOR DELIVERY")
                                 .withString("ShipmentTrackingId", "383283");

                putItemOutcome = table.putItem(item);
                assert putItemOutcome != null;
        }

}
```

## 로컬 보조 인덱스로 작업: .NET

주제

- 로컬 보조 인덱스가 있는 테이블 생성
- 로컬 보조 인덱스가 있는 테이블 설명
- 로컬 보조 인덱스 쿼리
- 예: AWS SDK for .NET 하위 수준 API를 사용하는 로컬 보조 인덱스

AWS SDK for .NET 하위 수준 API를 사용하여 하나 이상의 로컬 보조 인덱스가 포함된 Amazon DynamoDB 테이블을 만들고, 테이블의 인덱스를 설명하고, 인덱스를 사용하여 쿼리를 수행할 수 있습니다. 이들 작업은 해당되는 하위 수준 DynamoDB API 작업으로 매핑됩니다. 자세한 내용은 .NET 코드 예시 단원을 참조하십시오.

다음은 .NET 하위 수준 API를 사용하여 테이블 작업을 할 때 따라야 할 공통 단계입니다.

1. `AmazonDynamoDBClient` 클래스의 인스턴스를 만듭니다.
2. 해당하는 요청 객체를 만들어 작업의 필수 및 선택적 파라미터를 제공합니다.

   예를 들어 `CreateTableRequest` 객체를 만들어 테이블을 생성하거나 `QueryRequest` 객체를 만들어 테이블 또는 인덱스를 쿼리합니다.
3. 이전 단계에서 만든 클라이언트가 제공한 적절한 메서드를 실행합니다.

### 로컬 보조 인덱스가 있는 테이블 생성

로컬 보조 인덱스는 테이블을 만들 때 동시에 만들 수 있습니다. 이렇게 하려면 `CreateTable`을 사용하고 하나 이상의 로컬 보조 인덱스에 대한 사양을 제공합니다. 다음 C# 코드 예제는 보유한 음악 파일에 있는 곡의 정보를 담은 테이블을 만듭니다. 파티션 키는 `Artist`이고 정렬 키는 `SongTitle`입니다. 보조 인덱스인 `AlbumTitleIndex`는 앨범 제목을 사용해 쿼리를 쉽게 수행하는 데 사용합니다.

다음은 .NET 하위 수준 API를 사용하여 로컬 보조 인덱스가 포함된 테이블을 생성하는 단계입니다.

1. `AmazonDynamoDBClient` 클래스의 인스턴스를 만듭니다.
2. `CreateTableRequest` 클래스 인스턴스를 만들어 요청 정보를 입력합니다.

   이때 입력해야 하는 정보는 테이블 이름, 기본 키, 그리고 프로비저닝된 처리량 값입니다. 로컬 보조 인덱스의 경우 인덱스 이름, 인덱스 정렬 키의 이름 및 데이터 형식, 인덱스의 키 스키마, 속성 프로젝션을 입력해야 합니다.
3. 요청 객체를 파라미터로 입력하여 `CreateTable` 메서드를 실행합니다.

다음 C# 코드 예제에서는 이전 단계를 설명합니다. 이 코드는 `AlbumTitle` 속성에 보조 인덱스가 있는 테이블(`Music`)을 생성합니다. 인덱스에 프로젝션되는 속성은 테이블 파티션 키 및 정렬 키와 인덱스 정렬 키뿐입니다.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

CreateTableRequest createTableRequest = new CreateTableRequest()
{
    TableName = tableName
};

//ProvisionedThroughput
createTableRequest.ProvisionedThroughput = new ProvisionedThroughput()
{
    ReadCapacityUnits = (long)5,
    WriteCapacityUnits = (long)5
};

//AttributeDefinitions
List<AttributeDefinition> attributeDefinitions = new List<AttributeDefinition>();

attributeDefinitions.Add(new AttributeDefinition()
{
```

```
    AttributeName = "Artist",
    AttributeType = "S"
});

attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "SongTitle",
     AttributeType = "S"
 });

attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "AlbumTitle",
     AttributeType = "S"
 });

createTableRequest.AttributeDefinitions = attributeDefinitions;

//KeySchema
List<KeySchemaElement> tableKeySchema = new List<KeySchemaElement>();

tableKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
tableKeySchema.Add(new KeySchemaElement() { AttributeName = "SongTitle", KeyType =
 "RANGE" });  //Sort key

createTableRequest.KeySchema = tableKeySchema;

List<KeySchemaElement> indexKeySchema = new List<KeySchemaElement>();
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "AlbumTitle", KeyType =
 "RANGE" });  //Sort key

Projection projection = new Projection() { ProjectionType = "INCLUDE" };

List<string> nonKeyAttributes = new List<string>();
nonKeyAttributes.Add("Genre");
nonKeyAttributes.Add("Year");
projection.NonKeyAttributes = nonKeyAttributes;

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()
{
    IndexName = "AlbumTitleIndex",
```

```
    KeySchema = indexKeySchema,
    Projection = projection
};

List<LocalSecondaryIndex> localSecondaryIndexes = new List<LocalSecondaryIndex>();
localSecondaryIndexes.Add(localSecondaryIndex);
createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;

CreateTableResponse result = client.CreateTable(createTableRequest);
Console.WriteLine(result.CreateTableResult.TableDescription.TableName);
Console.WriteLine(result.CreateTableResult.TableDescription.TableStatus);
```

DynamoDB에서 테이블을 만들고 테이블 상태가 ACTIVE로 설정될 때까지 기다려야 합니다. 그런 다음 테이블에 데이터 항목을 입력할 수 있습니다.

로컬 보조 인덱스가 있는 테이블 설명

테이블의 로컬 보조 인덱스에 관한 자세한 내용은 DescribeTable API를 참조하세요. 각 인덱스에 대해 인덱스의 이름, 키 스키마 및 프로젝션된 속성에 액세스할 수 있습니다.

다음은 .NET 하위 수준 API를 사용하여 테이블의 로컬 보조 인덱스 정보에 액세스하는 단계입니다.

1. AmazonDynamoDBClient 클래스의 인스턴스를 만듭니다.
2. DescribeTableRequest 클래스 인스턴스를 만들어 요청 정보를 입력합니다. 테이블 이름을 입력해야 합니다.
3. 요청 객체를 파라미터로 입력하여 describeTable 메서드를 실행합니다.
4. 

다음 C# 코드 예제에서는 이전 단계를 설명합니다.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest()
 { TableName = tableName });
List<LocalSecondaryIndexDescription> localSecondaryIndexes =
    response.DescribeTableResult.Table.LocalSecondaryIndexes;
```

```
// This code snippet will work for multiple indexes, even though
// there is only one index in this example.
foreach (LocalSecondaryIndexDescription lsiDescription in localSecondaryIndexes)
{
    Console.WriteLine("Info for index " + lsiDescription.IndexName + ":");

    foreach (KeySchemaElement kse in lsiDescription.KeySchema)
    {
        Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
    }

    Projection projection = lsiDescription.Projection;

    Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

    if (projection.ProjectionType.ToString().Equals("INCLUDE"))
    {
        Console.WriteLine("\t\tThe non-key projected attributes are:");

        foreach (String s in projection.NonKeyAttributes)
        {
            Console.WriteLine("\t\t" + s);
        }

    }
}
```

로컬 보조 인덱스 쿼리

테이블을 Query할 때와 거의 동일한 방식으로 로컬 보조 인덱스에서 Query를 사용할 수 있습니다. 인덱스 이름, 인덱스 정렬 키의 쿼리 기준, 반환하려는 속성을 지정해야 합니다. 이 예제에서 인덱스는 AlbumTitleIndex이고 인덱스 정렬 키는 AlbumTitle입니다.

인덱스로 프로젝션된 속성만 반환됩니다. 키가 아닌 속성을 선택하도록 이 쿼리를 수정할 수도 있지만, 그렇게 하려면 비교적 많은 비용이 드는 테이블 가져오기 작업이 필요합니다. 테이블 가져오기에 대한 자세한 내용은 속성 프로젝션 단원을 참조하세요.

다음은 .NET 하위 수준 API를 사용하여 로컬 보조 인덱스를 쿼리하는 단계입니다.

1. AmazonDynamoDBClient 클래스의 인스턴스를 만듭니다.
2. QueryRequest 클래스 인스턴스를 만들어 요청 정보를 입력합니다.
3. 요청 객체를 파라미터로 입력하여 query 메서드를 실행합니다.

다음 C# 코드 예제에서는 이전 단계를 설명합니다.

Example

```
QueryRequest queryRequest = new QueryRequest
{
    TableName = "Music",
    IndexName = "AlbumTitleIndex",
    Select = "ALL_ATTRIBUTES",
    ScanIndexForward = true,
    KeyConditionExpression = "Artist = :v_artist and AlbumTitle = :v_title",
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":v_artist",new AttributeValue {S = "Acme Band"}},
        {":v_title",new AttributeValue {S = "Songs About Life"}}
    },
};

QueryResponse response = client.Query(queryRequest);

foreach (var attribs in response.Items)
{
    foreach (var attrib in attribs)
    {
        Console.WriteLine(attrib.Key + " ---> " + attrib.Value.S);
    }
    Console.WriteLine();
}
```

예: AWS SDK for .NET 하위 수준 API를 사용하는 로컬 보조 인덱스

다음 C# 코드 예제는 Amazon DynamoDB에서 로컬 보조 인덱스로 작업하는 방법을 보여 줍니다. 예를 들어 파티션 키가 `CustomerId`이고 정렬 키가 `OrderId`인 `CustomerOrders`라는 테이블을 만들 수 있습니다. 이 테이블에는 두 개의 로컬 보조 인덱스가 있습니다.

- `OrderCreationDateIndex` - 정렬 키는 `OrderCreationDate`이며 다음 속성이 인덱스로 프로젝션됩니다.
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`

  - ShipmentTrackingId
- IsOpenIndex - 정렬 키는 IsOpen이며 모든 테이블 속성이 인덱스로 프로젝션됩니다.

CustomerOrders 테이블이 생성되면 프로그램에서 고객 주문을 나타내는 데이터와 함께 테이블을 로드합니다. 그런 다음 로컬 보조 인덱스를 사용하여 데이터를 쿼리합니다. 마지막으로 프로그램에서 CustomerOrders 테이블을 삭제합니다.

다음 예제를 테스트하기 위한 단계별 지침은 .NET 코드 예시 단원을 참조하세요.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelLocalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        private static string tableName = "CustomerOrders";

        static void Main(string[] args)
        {
            try
            {
                CreateTable();
                LoadData();

                Query(null);
                Query("IsOpenIndex");
                Query("OrderCreationDateIndex");

                DeleteTable(tableName);

                Console.WriteLine("To continue, press Enter");
```

```
            Console.ReadLine();
        }
        catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
        catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
        catch (Exception e) { Console.WriteLine(e.Message); }
    }

    private static void CreateTable()
    {
        var createTableRequest =
            new CreateTableRequest()
            {
                TableName = tableName,
                ProvisionedThroughput =
                new ProvisionedThroughput()
                {
                    ReadCapacityUnits = (long)1,
                    WriteCapacityUnits = (long)1
                }
            };

        var attributeDefinitions = new List<AttributeDefinition>()
    {
        // Attribute definitions for table primary key
        { new AttributeDefinition() {
              AttributeName = "CustomerId", AttributeType = "S"
          } },
        { new AttributeDefinition() {
              AttributeName = "OrderId", AttributeType = "N"
          } },
        // Attribute definitions for index primary key
        { new AttributeDefinition() {
              AttributeName = "OrderCreationDate", AttributeType = "N"
          } },
        { new AttributeDefinition() {
              AttributeName = "IsOpen", AttributeType = "N"
          }}
    };

        createTableRequest.AttributeDefinitions = attributeDefinitions;

        // Key schema for table
        var tableKeySchema = new List<KeySchemaElement>()
    {
```

```
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              } },                                                    //Partition key
            { new KeySchemaElement() {
                  AttributeName = "OrderId", KeyType = "RANGE"
              } }                                                   //Sort key
        };

            createTableRequest.KeySchema = tableKeySchema;

            var localSecondaryIndexes = new List<LocalSecondaryIndex>();

            // OrderCreationDateIndex
            LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
            {
                IndexName = "OrderCreationDateIndex"
            };

            // Key schema for OrderCreationDateIndex
            var indexKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              } },                                                      //Partition key
            { new KeySchemaElement() {
                  AttributeName = "OrderCreationDate", KeyType = "RANGE"
              } }                                                                 //Sort key
        };

            orderCreationDateIndex.KeySchema = indexKeySchema;

            // Projection (with list of projected attributes) for
            // OrderCreationDateIndex
            var projection = new Projection()
            {
                ProjectionType = "INCLUDE"
            };

            var nonKeyAttributes = new List<string>()
        {
            "ProductCategory",
            "ProductName"
        };
            projection.NonKeyAttributes = nonKeyAttributes;
```

```
            orderCreationDateIndex.Projection = projection;

            localSecondaryIndexes.Add(orderCreationDateIndex);

            // IsOpenIndex
            LocalSecondaryIndex isOpenIndex
                = new LocalSecondaryIndex()
                {
                    IndexName = "IsOpenIndex"
                };

            // Key schema for IsOpenIndex
            indexKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              }},                                                    //Partition key
            { new KeySchemaElement() {
                  AttributeName = "IsOpen", KeyType = "RANGE"
              }}                                                  //Sort key
        };

            // Projection (all attributes) for IsOpenIndex
            projection = new Projection()
            {
                ProjectionType = "ALL"
            };

            isOpenIndex.KeySchema = indexKeySchema;
            isOpenIndex.Projection = projection;

            localSecondaryIndexes.Add(isOpenIndex);

            // Add index definitions to CreateTable request
            createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;

            Console.WriteLine("Creating table " + tableName + "...");
            client.CreateTable(createTableRequest);
            WaitUntilTableReady(tableName);
        }

        public static void Query(string indexName)
        {
```

```
Console.WriteLine("\n*************************************************************\n");
            Console.WriteLine("Querying table " + tableName + "...");

            QueryRequest queryRequest = new QueryRequest()
            {
                TableName = tableName,
                ConsistentRead = true,
                ScanIndexForward = true,
                ReturnConsumedCapacity = "TOTAL"
            };


            String keyConditionExpression = "CustomerId = :v_customerId";
            Dictionary<string, AttributeValue> expressionAttributeValues = new
Dictionary<string, AttributeValue> {
            {":v_customerId", new AttributeValue {
                 S = "bob@example.com"
             }}
        };


            if (indexName == "IsOpenIndex")
            {
                Console.WriteLine("\nUsing index: '" + indexName
                          + "': Bob's orders that are open.");
                Console.WriteLine("Only a user-specified list of attributes are
returned\n");
                queryRequest.IndexName = indexName;

                keyConditionExpression += " and IsOpen = :v_isOpen";
                expressionAttributeValues.Add(":v_isOpen", new AttributeValue
                {
                    N = "1"
                });

                // ProjectionExpression
                queryRequest.ProjectionExpression = "OrderCreationDate,
ProductCategory, ProductName, OrderStatus";
            }
            else if (indexName == "OrderCreationDateIndex")
            {
                Console.WriteLine("\nUsing index: '" + indexName
                          + "': Bob's orders that were placed after 01/31/2013.");
```

```
                Console.WriteLine("Only the projected attributes are returned\n");
                queryRequest.IndexName = indexName;

                keyConditionExpression += " and OrderCreationDate > :v_Date";
                expressionAttributeValues.Add(":v_Date", new AttributeValue
                {
                    N = "20130131"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else
            {
                Console.WriteLine("\nNo index: All of Bob's orders, by OrderId:\n");
            }
            queryRequest.KeyConditionExpression = keyConditionExpression;
            queryRequest.ExpressionAttributeValues = expressionAttributeValues;

            var result = client.Query(queryRequest);
            var items = result.Items;
            foreach (var currentItem in items)
            {
                foreach (string attr in currentItem.Keys)
                {
                    if (attr == "OrderId" || attr == "IsOpen"
                        || attr == "OrderCreationDate")
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].N);
                    }
                    else
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].S);
                    }
                }
                Console.WriteLine();
            }
            Console.WriteLine("\nConsumed capacity: " +
result.ConsumedCapacity.CapacityUnits + "\n");
        }

        private static void DeleteTable(string tableName)
        {
            Console.WriteLine("Deleting table " + tableName + "...");
```

```
            client.DeleteTable(new DeleteTableRequest()
            {
                TableName = tableName
            });
            WaitForTableToBeDeleted(tableName);
        }

        public static void LoadData()
        {
            Console.WriteLine("Loading data into table " + tableName + "...");

            Dictionary<string, AttributeValue> item = new Dictionary<string,
AttributeValue>();

            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "1"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130101"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Book"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "The Great Outdoors"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "PACKING ITEMS"
            };
            /* no ShipmentTrackingId attribute */
            PutItemRequest putItemRequest = new PutItemRequest
```

```
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "2"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130221"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Bike"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Super Mountain"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "ORDER RECEIVED"
            };
            /* no ShipmentTrackingId attribute */
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);
```

```
item = new Dictionary<string, AttributeValue>();
item["CustomerId"] = new AttributeValue
{
    S = "alice@example.com"
};
item["OrderId"] = new AttributeValue
{
    N = "3"
};
/* no IsOpen attribute */
item["OrderCreationDate"] = new AttributeValue
{
    N = "20130304"
};
item["ProductCategory"] = new AttributeValue
{
    S = "Music"
};
item["ProductName"] = new AttributeValue
{
    S = "A Quiet Interlude"
};
item["OrderStatus"] = new AttributeValue
{
    S = "IN TRANSIT"
};
item["ShipmentTrackingId"] = new AttributeValue
{
    S = "176493"
};
putItemRequest = new PutItemRequest
{
    TableName = tableName,
    Item = item,
    ReturnItemCollectionMetrics = "SIZE"
};
client.PutItem(putItemRequest);

item = new Dictionary<string, AttributeValue>();
item["CustomerId"] = new AttributeValue
{
    S = "bob@example.com"
};
```

```
            item["OrderId"] = new AttributeValue
            {
                N = "1"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130111"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Movie"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Calm Before The Storm"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "SHIPPING DELAY"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "859323"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "2"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
```

```
            {
                N = "20130124"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "E-Z Listening"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "756943"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "3"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130221"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
```

```
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Symphony 9"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "645193"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "4"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130222"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Hardware"
            };
            item["ProductName"] = new AttributeValue
            {
```

```
            S = "Extra Heavy Hammer"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "PACKING ITEMS"
        };
        /* no ShipmentTrackingId attribute */
        putItemRequest = new PutItemRequest
        {
            TableName = tableName,
            Item = item,
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);

        item = new Dictionary<string, AttributeValue>();
        item["CustomerId"] = new AttributeValue
        {
            S = "bob@example.com"
        };
        item["OrderId"] = new AttributeValue
        {
            N = "5"
        };
        /* no IsOpen attribute */
        item["OrderCreationDate"] = new AttributeValue
        {
            N = "20130309"
        };
        item["ProductCategory"] = new AttributeValue
        {
            S = "Book"
        };
        item["ProductName"] = new AttributeValue
        {
            S = "How To Cook"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "IN TRANSIT"
        };
        item["ShipmentTrackingId"] = new AttributeValue
        {
            S = "440185"
```

```
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "6"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130318"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Luggage"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Really Big Suitcase"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "893927"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
```

```
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "7"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130324"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Golf"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "PGA Pro II"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "OUT FOR DELIVERY"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "383283"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);
        }

        private static void WaitUntilTableReady(string tableName)
        {
```

```
            string status = null;
            // Let us wait until table is created. Call DescribeTable.
            do
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                catch (ResourceNotFoundException)
                {
                    // DescribeTable is eventually consistent. So you might
                    // get resource not found. So we handle the potential exception.
                }
            } while (status != "ACTIVE");
        }

        private static void WaitForTableToBeDeleted(string tableName)
        {
            bool tablePresent = true;

            while (tablePresent)
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                }
                catch (ResourceNotFoundException)
```

```
                {
                    tablePresent = false;
                }
            }
        }
    }
}
```

## DynamoDB AWS CLI의 로컬 보조 인덱스로 작업

AWS CLI를 사용하여 하나 이상의 로컬 보조 인덱스가 있는 Amazon DynamoDB 테이블을 생성하고, 테이블의 인덱스를 설명하고, 인덱스를 사용하여 쿼리를 수행할 수 있습니다.

주제

- 로컬 보조 인덱스가 있는 테이블 생성
- 로컬 보조 인덱스가 있는 테이블 설명
- 로컬 보조 인덱스 쿼리

### 로컬 보조 인덱스가 있는 테이블 생성

로컬 보조 인덱스는 테이블을 생성할 때 동시에 생성해야 합니다. 이렇게 하려면 `create-table` 파라미터를 사용하고 하나 이상의 로컬 보조 인덱스에 대한 사양을 제공합니다. 다음 예제는 보유한 음악 파일에 있는 곡의 정보를 담은 테이블(`Music`)을 생성합니다. 파티션 키는 `Artist`이고 정렬 키는 `SongTitle`입니다. 보조 인덱스인 `AlbumTitle` 속성의 `AlbumTitleIndex`는 앨범 제목을 사용해 쿼리를 쉽게 수행하는 데 사용합니다.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S  \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[{\"IndexName\": \"AlbumTitleIndex\",
        \"KeySchema\":[{\"AttributeName\":\"Artist\",\"KeyType\":\"HASH\"},
                       {\"AttributeName\":\"AlbumTitle\",\"KeyType\":\"RANGE\"}],
```

```
        \"Projection\":{\"ProjectionType\":\"INCLUDE\",  \"NonKeyAttributes\":[\"Genre
\", \"Year\"]}}]"
```

DynamoDB에서 테이블을 만들고 테이블 상태가 ACTIVE로 설정될 때까지 기다려야 합니다. 그런 다음 테이블에 데이터 항목을 입력할 수 있습니다. describe table을 사용하여 테이블 생성 상태를 결정할 수 있습니다.

로컬 보조 인덱스가 있는 테이블 설명

테이블의 로컬 보조 인덱스에 관한 자세한 내용은 describe-table 파라미터를 참조하세요. 각 인덱스에 대해 인덱스의 이름, 키 스키마 및 프로젝션된 속성에 액세스할 수 있습니다.

```
aws dynamodb describe-table --table-name Music
```

로컬 보조 인덱스 쿼리

테이블을 query할 때와 거의 동일한 방식으로 로컬 보조 인덱스의 query 작업을 사용할 수 있습니다. 인덱스 이름, 인덱스 정렬 키의 쿼리 기준, 반환하려는 속성을 지정해야 합니다. 이 예제에서 인덱스는 AlbumTitleIndex이고 인덱스 정렬 키는 AlbumTitle입니다.

인덱스로 프로젝션된 속성만 반환됩니다. 키가 아닌 속성을 선택하도록 이 쿼리를 수정할 수도 있지만, 그렇게 하려면 비교적 많은 비용이 드는 테이블 가져오기 작업이 필요합니다. 테이블 가져오기에 대한 자세한 내용은 속성 프로젝션 단원을 참조하세요.

```
aws dynamodb query \
    --table-name Music \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v_artist and AlbumTitle = :v_title" \
    --expression-attribute-values  '{":v_artist":{"S":"Acme Band"},":v_title":
{"S":"Songs About Life"} }'
```

# DynamoDB Transactions를 사용하여 복잡한 워크플로 관리

Amazon DynamoDB Transactions는 테이블 내나 테이블 간 모두에서 여러 항목을 조정된 양자택일 방식으로 변경하는 개발자 경험을 간소화합니다. 트랜잭션은 DynamoDB에서 원자성, 일관성, 격리 및 내구성(ACID)을 제공하여 애플리케이션에서 데이터 정확성을 유지하는 데 도움이 됩니다.

DynamoDB 트랜잭션 읽기 및 쓰기 API를 사용하면 여러 항목을 추가, 업데이트 또는 삭제해야 하는 복잡한 비즈니스 워크플로우를 한 번의 양자택일 방식 작업으로 관리할 수 있습니다. 예를 들어, 비디오

게임 개발자는 게임에서 아이템을 교환하거나 게임 내 구매 시 플레이어의 프로필이 올바르게 업데이트 되도록 할 수 있습니다.

트랜잭션 쓰기 API에서는 여러 `Put`, `Update`, `Delete` 및 `ConditionCheck` 작업을 그룹화할 수 있습니다. 그런 다음, 이들을 하나의 단위로 성공하거나 실패하는 단일 `TransactWriteItems` 작업으로 제출할 수 있습니다. 그룹화한 후 단일 `TransactGetItems` 작업으로 제출할 수 있는 여러 `Get` 작업의 경우도 마찬가지입니다.

DynamoDB 테이블에 대해 트랜잭션을 활성화하는 데 추가 비용이 들지 않습니다. 트랜잭션에 포함되는 읽기 또는 쓰기에 대해서만 비용을 지불하면 됩니다. DynamoDB는 트랜잭션의 모든 항목에 대해 두 개의 기본 읽기 또는 쓰기를 수행합니다. 하나는 트랜잭션을 준비하고, 하나는 트랜잭션을 커밋하기 위한 것입니다. 이 두 개의 기본 읽기/쓰기 작업은 Amazon CloudWatch 지표로 표시됩니다.

DynamoDB 트랜잭션을 시작하려면 최신 AWS SDK 또는 AWS Command Line Interface(AWS CLI)를 다운로드합니다. 그런 다음 DynamoDB Transactions 예 섹션을 따르세요.

다음 섹션에서는 트랜잭션 API에 대한 세부 개요를 제공하고 DynamoDB에서 트랜잭션을 사용하는 방법을 설명합니다.

주제

- Amazon DynamoDB Transactions: 작동 방식
- DynamoDB Transactions에서 IAM 사용
- DynamoDB Transactions 예

## Amazon DynamoDB Transactions: 작동 방식

Amazon DynamoDB Transactions를 사용하면 여러 작업을 그룹화하고 하나의 양자택일 `TransactWriteItems` 또는 `TransactGetItems` 작업으로 제출할 수 있습니다. 다음 섹션에서는 DynamoDB에서 트랜잭션 작업 사용과 관련된 API 작업, 용량 관리, 모범 사례 및 기타 세부 정보를 설명합니다.

주제

- TransactWriteItems API
- TransactGetItems API
- DynamoDB Transactions의 격리 수준
- DynamoDB의 트랜잭션 충돌 처리

- DynamoDB Accelerator(DAX)에서 트랜잭션 API 사용
- 트랜잭션 용량 관리
- 트랜잭션 모범 사례
- 전역 테이블에 트랜잭션 API 사용
- DynamoDB Transactions와 AWSLabs 트랜잭션 클라이언트 라이브러리 비교

## TransactWriteItems API

`TransactWriteItems`는 최대 100개의 쓰기 작업을 한 번에 모두 수행하거나 아무것도 수행하지 않는 작업으로 그룹화하는 동기식 idempotent 쓰기 작업입니다. 이러한 작업에서는 동일한 AWS 계정과 동일한 리전에 속한 하나 이상의 DynamoDB 테이블에서 최대 100개의 개별 항목을 대상으로 지정할 수 있습니다. 트랜잭션 내 항목의 총 크기는 4MB를 초과할 수 없습니다. 작업은 원자 단위로 완료되므로 작업이 모두 성공하거나 모두 실패하게 됩니다.

> **Note**
>
> - `TransactWriteItems` 작업은 `BatchWriteItem` 작업과 달리 포함된 작업이 모두 성공적으로 완료되거나 전혀 변경되지 않아야 합니다. `BatchWriteItem` 작업에서는 배치 내 일부 작업만 성공하고 나머지는 실패할 수 있습니다.
> - 인덱스를 사용하여 트랜잭션을 수행할 수 없습니다.

동일한 트랜잭션 내에서 동일한 항목을 여러 작업의 대상으로 지정할 수 없습니다. 예를 들어 동일한 트랜잭션에서 동일한 항목에 대해 `ConditionCheck` 작업과 `Update` 작업을 동시에 수행할 수 없습니다.

다음과 같은 유형의 작업은 동일한 트랜잭션에 추가할 수 있습니다.

- `Put` - `PutItem` 작업을 시작하여 조건부로 또는 조건을 지정하지 않고 새 항목을 생성하거나 이전 항목을 새 항목으로 바꿉니다.
- `Update` - `UpdateItem` 작업을 시작하여 기존 항목의 속성을 편집합니다. 아직 항목이 없는 경우 새 항목을 테이블에 추가합니다. 이 작업을 사용하여 조건부로 또는 조건을 적용하지 않고 기존 항목에 대한 속성을 추가, 삭제 또는 업데이트할 수 있습니다.
- `Delete` - `DeleteItem` 작업을 시작하여 테이블에서 기본 키로 식별된 단일 항목을 삭제합니다.
- `ConditionCheck` - 항목이 있는지 확인하거나 항목의 특정 속성에 대한 조건을 검사합니다.

DynamoDB에서 트랜잭션이 완료되면 변경 사항이 글로벌 보조 인덱스(GSI), 스트림 및 백업으로 전파되기 시작합니다. 이 전파는 점진적으로 발생합니다. 동일한 트랜잭션의 스트림 레코드가 서로 다른 시간에 나타날 수 있으며 다른 트랜잭션의 레코드와 인터리브될 수 있습니다. 스트림 소비자는 트랜잭션 원자성 또는 순서가 보장된다고 가정해서는 안 됩니다.

트랜잭션에서 수정된 항목의 원자 스냅샷을 보장하려면 TransactGetItems 작업을 사용하여 모든 관련 항목을 함께 읽습니다. 이 작업은 데이터에 대한 일관된 보기를 제공하므로 완료된 트랜잭션의 모든 변경 사항을 보거나 전혀 볼 수 없습니다.

전파는 즉시 이루어지지 않기 때문에 전파 중간에 테이블을 백업에서 복원하거나 (RestoreTableFromBackup) 특정 시점으로 내보낸(ExportTableToPointInTime) 경우에는 최근 트랜잭션 동안 수행된 변경 사항의 일부만이 포함될 수 있습니다.

멱등성

`TransactWriteItems`를 호출할 때 요청이 멱등성인지 확인하기 위해 클라이언트 토큰을 선택적으로 포함할 수 있습니다. 트랜잭션을 멱등성으로 지정하면 연결 시간 초과 또는 기타 연결 문제로 인해 동일한 작업이 여러 번 제출될 경우에 애플리케이션 오류를 방지할 수 있습니다.

원래 `TransactWriteItems` 호출이 성공한 경우 동일한 클라이언트 토큰을 포함하는 후속 `TransactWriteItems` 호출은 변경 없이 성공적으로 반환됩니다. `ReturnConsumedCapacity` 파라미터를 설정한 경우 초기 `TransactWriteItems` 호출은 변경하는 데 사용된 쓰기 용량 단위 수를 반환합니다. 동일한 클라이언트 토큰을 포함하는 후속 `TransactWriteItems` 호출은 항목을 읽는 데 사용된 읽기 용량 단위 수를 반환합니다.

멱등성에 대한 중요 사항

- 클라이언트 토큰은 해당 토큰을 사용하는 요청이 완료된 후 10분 동안 유효합니다. 10분이 경과한 이후에는 동일한 클라이언트 토큰을 사용하는 모든 요청이 새 요청으로 처리됩니다. 10분이 경과한 이후에는 동일한 요청에 대해 동일한 클라이언트 토큰을 재사용하면 안 됩니다.
- 10분 멱등성 기간 내에 동일한 클라이언트 토큰으로 요청을 반복하지만 일부 다른 요청 파라미터를 변경할 경우 DynamoDB는 `IdempotentParameterMismatch` 예외를 반환합니다.

쓰기 오류 처리

다음과 같은 경우 쓰기 트랜잭션이 실패합니다.

- 조건식 중 하나의 조건이 충족되지 않는 경우
- 동일한 `TransactWriteItems` 작업 내 여러 작업에서 동일한 항목을 대상으로 지정하여 트랜잭션 검증 오류가 발생하는 경우

- `TransactWriteItems` 요청이 `TransactWriteItems` 요청에 있는 한 개 이상의 항목에 대한 지속적 `TransactWriteItems` 작업과 충돌하는 경우 이 경우 요청은 실패하고 `TransactionCanceledException`이 반환됩니다.
- 트랜잭션을 완료하기 위한 프로비저닝 용량이 부족한 경우
- 트랜잭션에 의한 변경으로 인해 항목 크기가 지나치게 커지거나(400KB 초과), 로컬 보조 인덱스(LSI)가 너무 커지거나, 유사한 검증 오류가 발생하는 경우
- 사용자 오류(예: 잘못된 데이터 형식)가 발생하는 경우

`TransactWriteItems` 작업과의 충돌을 처리하는 방법에 대한 자세한 내용은 DynamoDB의 트랜잭션 충돌 처리 섹션을 참조하세요.

## TransactGetItems API

`TransactGetItems`는 최대 100개의 `Get` 작업을 함께 그룹화하는 동기식 읽기 작업입니다. 이러한 작업에서는 동일한 AWS 계정과 리전에 속한 하나 이상의 DynamoDB 테이블에서 최대 100개의 개별 항목을 대상으로 지정할 수 있습니다. 트랜잭션 내 항목의 총 크기는 4MB를 초과할 수 없습니다.

`Get` 작업은 원자 단위로 수행되므로 작업이 모두 성공하거나 모두 실패하게 됩니다.

- `Get` - `GetItem` 작업을 시작하여 지정된 기본 키의 항목에 대한 속성 집합을 검색합니다. 일치하는 항목이 없으면 `Get`이 데이터를 반환하지 않습니다.

### 읽기 오류 처리

다음과 같은 경우 읽기 트랜잭션이 실패합니다.

- `TransactGetItems` 요청이 `TransactGetItems` 요청에 있는 한 개 이상의 항목에 대한 지속적 `TransactWriteItems` 작업과 충돌하는 경우 이 경우 요청은 실패하고 `TransactionCanceledException`이 반환됩니다.
- 트랜잭션을 완료하기 위한 프로비저닝 용량이 부족한 경우
- 사용자 오류(예: 잘못된 데이터 형식)가 발생하는 경우

`TransactGetItems` 작업과의 충돌을 처리하는 방법에 대한 자세한 내용은 DynamoDB의 트랜잭션 충돌 처리 섹션을 참조하세요.

## DynamoDB Transactions의 격리 수준

트랜잭션 작업(TransactWriteItems 또는 TransactGetItems) 및 기타 작업의 격리 수준은 다음과 같습니다.

직렬화

직렬화 격리 수준에서는 여러 동시 작업의 결과가 이전 작업이 완료된 때까지 다른 작업이 시작되지 않는 경우와 동일합니다.

다음과 같은 작업 유형 간에는 직렬화 격리가 존재합니다.

- 트랜잭션 작업과 표준 쓰기 작업(PutItem, UpdateItem 또는 DeleteItem) 사이
- 트랜잭션 작업과 표준 읽기 작업(GetItem) 사이
- TransactWriteItems 작업과 TransactGetItems 작업 사이

트랜잭션 작업과 BatchWriteItem 작업의 각 표준 쓰기 사이에는 직렬화 격리가 존재하지만, 트랜잭션과 단위 BatchWriteItem 작업 사이에는 직렬화 격리가 없습니다.

그리고 트랜잭션 작업과 BatchGetItem 작업의 개별 GetItems사이의 격리 수준도 직렬화할 수 있습니다. 그러나 트랜잭션과 하나의 유닛으로서 BatchGetItem 작업 사이의 격리 수준은 읽기 커밋됩니다.

단일 GetItem 요청은 TransactWriteItems 요청 이전이나 이후에 두 가지 방법 중 하나로 TransactWriteItems 요청을 기준으로 직렬화할 수 있습니다. 동시 TransactWriteItems 요청의 키에 대한 여러 GetItem 요청은 원하는 순서로 실행할 수 있으므로 결과는 읽기 커밋됩니다.

예를 들어 항목 A와 항목 B에 대한 GetItem 요청이 항목 A와 항목 B를 모두 수정하는 TransactWriteItems 요청과 동시에 실행되는 경우 네 가지 가능성이 있습니다.

- 두 GetItem 요청이 TransactWriteItems 요청 전에 실행됩니다.
- 두 GetItem 요청이 TransactWriteItems 요청 후에 실행됩니다.
- 항목 A에 대한 GetItem 요청이 TransactWriteItems 요청 전에 실행됩니다. 항목 B의 경우는 GetItem이 TransactWriteItems 후에 실행됩니다.
- 항목 B에 대한 GetItem 요청이 TransactWriteItems 요청 전에 실행됩니다. 항목 A의 경우는 GetItem이 TransactWriteItems 후에 실행됩니다.

여러 GetItem 요청에 직렬화 가능 격리 수준을 사용하는 것을 선호하면 TransactGetItems를 사용해야 합니다.

진행 중인 동일한 트랜잭션 쓰기 요청에 포함되었던 여러 항목에 대해 비트랜잭션 읽기가 수행되면 일부 항목의 새 상태와 다른 항목의 이전 상태를 읽는 것이 가능할 수 있습니다. 트랜잭션 쓰기에 대한 응답이 성공적으로 수신된 경우에만(트랜잭션이 완료되었음을 의미) 트랜잭션 쓰기 요청에 포함되었던 모든 항목의 새 상태를 읽을 수 있습니다.

트랜잭션이 성공적으로 완료되고 응답이 수신된 후, DynamoDB의 최종적 일관성 모델로 인해 후속 최종적으로 일관된 읽기 작업이 여전히 단기간 동안 이전 상태를 반환할 수 있습니다. 트랜잭션 직후 최신 데이터를 읽도록 보장하려면 ConsistentRead를 true로 설정하여 강력하게 일관된 읽기를 사용해야 합니다.

읽기 커밋됨

읽기 커밋됨 격리는 읽기 작업이 항상 항목에 대해 커밋된 값을 반환하도록 합니다. 따라서 읽기는 최종적으로 성공하지 못한 트랜잭션 쓰기의 상태를 나타내는 항목에 대한 보기를 제공하지 않습니다. 읽기 커밋됨 격리는 읽기 작업 직후에 항목에 대한 수정을 막지 않습니다.

트랜잭션 작업과 여러 표준 읽기(BatchGetItem, Query 또는 Scan)를 포함하는 읽기 작업 사이에는 읽기 커밋됨 격리 수준이 있습니다. 컴퓨터에서 DynamoDB v2.5.3 이상을 실행하려면 Java 런타임 환경(JRE) 버전 17.x 이상이 필요합니다.

작업 요약

요약하면 다음 표는 트랜잭션 작업(TransactWriteItems 또는 TransactGetItems)과 다른 작업 사이의 격리 수준을 보여줍니다.

| Operation | 격리 수준 |
| --- | --- |
| DeleteItem | 직렬화 |
| PutItem | 직렬화 |
| UpdateItem | 직렬화 |
| GetItem | 직렬화 |
| BatchGetItem | 읽기 커밋됨* |
| BatchWriteItem | 직렬화 불가* |

| Operation | 격리 수준 |
| --- | --- |
| Query | 읽기 커밋됨 |
| Scan | 읽기 커밋됨 |
| 기타 트랜잭션 작업 | 직렬화 |

별표(*) 표시된 수준은 작업에 하나의 단위로 적용됩니다. 하지만 이러한 작업 내의 개별 작업에는 직렬화 격리 수준이 있습니다.

## DynamoDB의 트랜잭션 충돌 처리

트랜잭션 안의 항목에 대해 항목 수준에서 동시에 여러 요청이 있을 경우 트랜잭션 충돌이 발생할 수 있습니다. 다음과 같은 시나리오에서 트랜잭션 충돌이 발생할 수 있습니다.

- 한 항목에 대한 `PutItem`, `UpdateItem` 또는 `DeleteItem` 요청이 이 항목을 포함하는 진행 중인 `TransactWriteItems` 요청과 충돌합니다.
- `TransactWriteItems` 안의 한 항목이 진행 중인 다른 `TransactWriteItems` 요청에 속합니다.
- `TransactGetItems` 안의 한 항목이 진행 중인 `TransactWriteItems`, `BatchWriteItem`, `PutItem`, `UpdateItem` 또는 `DeleteItem` 요청에 속합니다.

> **Note**
>
> - `PutItem`, `UpdateItem` 또는 `DeleteItem` 요청이 거부되면 `TransactionConflictException`와 함께 해당 요청에 실패합니다.
> - `TransactWriteItems` 또는 `TransactGetItems` 안의 항목 수준 요청이 하나라도 거부되면 `TransactionCanceledException`과 함께 해당 요청에 실패합니다. 해당 요청이 실패하면 AWS SDK는 요청을 다시 시도하지 않습니다.
>
>   AWS SDK for Java를 사용 중인 경우 CancellationReasons의 목록이 `TransactItems` 요청 파라미터의 항목 목록에 따라 정렬되어 예외에 포함됩니다. 기타 언어는 목록의 문자열 표현이 예외의 오류 메시지에 포함됩니다.
> - 지속적 `TransactWriteItems` 또는 `TransactGetItems` 작업이 동시 `GetItem` 요청과 충돌하는 경우에는 두 작업 모두 성공할 수 있습니다.

실패한 각 항목 수준의 요청에 대해 TransactionConflict CloudWatch 지표가 증가합니다.

## DynamoDB Accelerator(DAX)에서 트랜잭션 API 사용

DynamoDB와 격리 수준이 동일한 DynamoDB Accelerator(DAX)에서는 `TransactWriteItems` 및 `TransactGetItems`가 모두 지원됩니다.

`TransactWriteItems`는 DAX를 통해 씁니다. DAX는 `TransactWriteItems` 호출을 DynamoDB에 전달하고 응답을 반환합니다. 쓰기 후 캐시를 채우기 위해 DAX는 `TransactWriteItems` 작업의 각 항목에 대해 백그라운드에서 `TransactGetItems`를 호출하며, 이 작업은 추가 읽기 용량 단위를 사용합니다. (자세한 내용은 트랜잭션 용량 관리 섹션을 참조하세요.) 이 기능을 사용하면 애플리케이션 로직을 단순하게 유지할 수 있으며 트랜잭션 작업과 비트랜잭션 작업 모두에 대해 DAX를 사용할 수 있습니다.

`TransactGetItems` 호출은 항목을 로컬로 캐시하지 않고 DAX를 통해 전달됩니다. 이는 DAX의 강력한 읽기 일관성 API에서와 동일한 동작입니다.

## 트랜잭션 용량 관리

DynamoDB 테이블에 대해 트랜잭션을 활성화하는 데 추가 비용이 들지 않습니다. 트랜잭션에 포함되는 읽기 또는 쓰기에 대해서만 비용을 지불하면 됩니다. DynamoDB는 트랜잭션의 모든 항목에 대해 두 개의 기본 읽기 또는 쓰기를 수행합니다. 하나는 트랜잭션을 준비하고, 하나는 트랜잭션을 커밋하기 위한 것입니다. 이 두 개의 기본 읽기/쓰기 작업은 Amazon CloudWatch 지표로 표시됩니다.

테이블에 대한 용량을 프로비저닝할 때 트랜잭션 API에 필요한 추가 읽기 및 쓰기를 계획합니다. 예를 들어, 애플리케이션은 초당 1개의 트랜잭션을 실행하고 각 트랜잭션은 테이블에서 500바이트 항목 3개를 쓴다고 가정합니다. 각 항목은 각각 트랜잭션 준비 및 커밋을 위한 2개의 쓰기 용량 단위(WCU)가 필요합니다. 따라서 테이블에 대해 6 WCU를 프로비저닝해야 합니다.

이전 예제에서 DynamoDB Accelerator(DAX)를 사용하는 경우에도 `TransactWriteItems` 호출의 각 항목에 대해 두 개의 읽기 용량 단위(RCU)를 사용하게 됩니다. 따라서 테이블에 대해 6 WCU를 추가로 프로비저닝해야 합니다.

마찬가지로 애플리케이션이 초당 1개의 쓰기 트랜잭션을 실행하고 각 트랜잭션이 테이블에서 500바이트 항목 3개를 읽는 경우 테이블에 대해 6개의 읽기 용량 단위(RCU)를 프로비저닝해야 합니다. 각 항목을 읽는 데 2 RCU(트랜잭션 준비 및 커밋)가 필요합니다.

또한 기본 SDK 동작에서는 `TransactionInProgressException` 예외가 발생할 경우 트랜잭션을 다시 시도합니다. 이러한 재시도에서 사용되는 추가 읽기 용량 단위(RCU)를 계획합니다.

ClientRequestToken을 사용하여 자체 코드에서 트랜잭션을 재시도하는 경우에도 마찬가지입니다.

## 트랜잭션 모범 사례

DynamoDB 트랜잭션을 사용할 경우 다음과 같은 권장 방법을 고려하세요.

- 테이블에서 Auto Scaling을 활성화하거나, 트랜잭션의 모든 항목에 대해 2회의 읽기 또는 쓰기 작업을 수행하는 데 충분한 처리량을 프로비저닝했는지 확인합니다.
- AWS 제공 SDK를 사용하지 않을 경우 TransactWriteItems를 호출할 때 요청이 멱등성인지 확인하도록 ClientRequestToken 속성을 포함합니다.
- 필요하지 않은 경우 트랜잭션에서 작업을 그룹화하지 않습니다. 예를 들어, 애플리케이션 동시성을 훼손하지 않으면서 10개 작업을 포함하는 단일 트랜잭션을 여러 트랜잭션으로 분류할 수 있는 경우 트랜잭션을 분할하는 것이 좋습니다. 트랜잭션이 간단할수록 처리량이 향상되어 성공할 가능성이 더 높아집니다.
- 여러 트랜잭션에서 동일한 항목을 동시에 업데이트할 경우 충돌이 발생하여 트랜잭션이 취소될 수 있습니다. 그런 충돌을 최소화하려면 데이터 모델링에 대한 DynamoDB 모범 사례를 따르는 것이 좋습니다.
- 종종 속성 세트가 단일 트랜잭션의 일부로 여러 항목에 걸쳐 업데이트되는 경우 속성을 단일 항목으로 그룹화하여 트랜잭션의 범위를 좁히는 것이 좋습니다.
- 대량으로 데이터를 수집하는 트랜잭션을 수행하지 마세요. 대량 쓰기의 경우 BatchWriteItem을 사용하는 것이 좋습니다.

## 전역 테이블에 트랜잭션 API 사용

트랜잭션 작업은 쓰기 API가 간접적으로 호출된 AWS 리전에서만 ACID(원자성, 일관성, 격리 및 내구성) 보장을 제공합니다. 글로벌 테이블에서는 트랜잭션이 리전 간에 지원되지 않습니다. 예를 들어 미국 동부(오하이오) 및 미국 서부(오레곤) 리전에 복제본이 있는 전역 테이블이 있고 미국 동부(버지니아 북부) 리전에서 TransactWriteItems 작업을 수행한다고 가정합니다. 변경 사항이 복제될 때 미국 서부(오리건) 리전에서 부분적으로 완료된 트랜잭션을 관찰할 수 있습니다. 변경 사항은 소스 리전에서 커밋된 이후에만 다른 리전에 복제됩니다.

## DynamoDB Transactions와 AWSLabs 트랜잭션 클라이언트 라이브러리 비교

DynamoDB Transactions는 AWSLabs 트랜잭션 클라이언트 라이브러리보다 경제적이고 강력하고 성능이 우수한 대체 방법입니다. 기본 서버측 트랜잭션 API를 사용하도록 애플리케이션을 업데이트하는 것이 좋습니다.

# DynamoDB Transactions에서 IAM 사용

AWS Identity and Access Management(IAM)를 사용하여 Amazon DynamoDB에서 트랜잭션 작업이 수행할 수 있는 작업을 제한할 수 있습니다. DynamoDB에서 IAM 정책을 사용하는 방법에 대한 자세한 내용은 DynamoDB에 대한 자격 증명 기반 정책 섹션을 참조하세요.

`Put`, `Update`, `Delete` 및 `Get` 작업에 대한 권한은 기본 `PutItem`, `UpdateItem`, `DeleteItem` 및 `GetItem` 작업에 사용되는 권한에 따라 규제됩니다. `ConditionCheck` 작업의 경우 IAM 정책에서 `dynamodb:ConditionCheckItem` 권한을 사용할 수 있습니다.

다음은 DynamoDB 트랜잭션을 구성하는 데 사용할 수 있는 IAM 정책의 예입니다.

## 예 1: 트랜잭션 작업 허용

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ]
        }
    ]
}
```

## 예 2: 트랜잭션 작업만 허용

JSON

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        }
    ]
}
```

### 예 3: 비트랜잭션 읽기 및 쓰기 허용, 트랜잭션 읽기 및 쓰기 차단

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
```

```
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
             "Action": [
                 "dynamodb:PutItem",
                 "dynamodb:DeleteItem",
                 "dynamodb:GetItem",
                 "dynamodb:UpdateItem"
             ],
             "Resource": [
                 "arn:aws:dynamodb:*:*:table/table04"
             ]
         }
    ]
}
```

## 예 4: ConditionCheck 실패 시 정보 반환 방지

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
```

```
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/table01",
            "Condition": {
                "StringEqualsIfExists": {
                    "dynamodb:ReturnValues": "NONE"
                }
            }
        }
    ]
}
```

## DynamoDB Transactions 예

Amazon DynamoDB Transactions가 유용할 수 있는 상황의 예로 온라인 마켓플레이스용 이 샘플 Java 애플리케이션을 고려하세요.

이 애플리케이션의 백엔드에는 다음 세 개의 DynamoDB 테이블이 있습니다.

- `Customers` - 이 테이블은 마켓플레이스 고객에 대한 세부 정보를 저장합니다. 기본 키는 `CustomerId` 고유 식별자입니다.
- `ProductCatalog` - 이 테이블은 마켓플레이스에서 판매할 제품에 대한 가격, 가용성 등의 세부 정보를 저장합니다. 기본 키는 `ProductId` 고유 식별자입니다.
- `Orders` - 이 테이블은 마켓플레이스에서 수행된 주문에 대한 세부 정보를 저장합니다. 기본 키는 `OrderId` 고유 식별자입니다.

### 주문하기

다음 코드 조각은 DynamoDB 트랜잭션을 사용하여 주문을 생성 및 처리하는 데 필요한 여러 단계를 조정하는 방법을 보여 줍니다. 하나의 양자택일 방식 작업을 사용하면 트랜잭션의 일부라도 실패할 경우 트랜잭션에서 아무 작업도 실행되지 않고 아무 내용도 변경되지 않습니다.

이 예제에서는 `customerId`이 `09e8e9c8-ec48`인 고객으로부터 주문을 설정합니다. 그리고 다음과 같이 간단한 주문 처리 워크플로를 사용하여 단일 트랜잭션으로 이를 실행합니다.

1. 고객 ID가 유효한지 확인합니다.
2. 제품이 `IN_STOCK` 상태인지 확인하고 제품 상태를 `SOLD`로 업데이트합니다.
3. 주문이 아직 없는지 확인하고 주문을 생성합니다.

고객 검증

먼저 customerId가 09e8e9c8-ec48인 고객이 고객 테이블에 있는지 확인하는 작업을 정의합니다.

```
final String CUSTOMER_TABLE_NAME = "Customers";
final String CUSTOMER_PARTITION_KEY = "CustomerId";
final String customerId = "09e8e9c8-ec48";
final HashMap<String, AttributeValue> customerItemKey = new HashMap<>();
customerItemKey.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));

ConditionCheck checkCustomerValid = new ConditionCheck()
    .withTableName(CUSTOMER_TABLE_NAME)
    .withKey(customerItemKey)
    .withConditionExpression("attribute_exists(" + CUSTOMER_PARTITION_KEY + ")");
```

제품 상태 업데이트

그런 다음 제품 상태가 현재 IN_STOCK으로 설정되어 있는지 확인하는 조건이 true인 경우 제품 상태를 SOLD로 업데이트하도록 하는 작업을 정의합니다. ReturnValuesOnConditionCheckFailure 파라미터를 설정하면 항목의 제품 상태 속성이 IN_STOCK이 아닌 경우 해당 항목을 반환합니다.

```
final String PRODUCT_TABLE_NAME = "ProductCatalog";
final String PRODUCT_PARTITION_KEY = "ProductId";
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
expressionAttributeValues.put(":new_status", new AttributeValue("SOLD"));
expressionAttributeValues.put(":expected_status", new AttributeValue("IN_STOCK"));

Update markItemSold = new Update()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey)
    .withUpdateExpression("SET ProductStatus = :new_status")
    .withExpressionAttributeValues(expressionAttributeValues)
    .withConditionExpression("ProductStatus = :expected_status")

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD);
```

주문 생성

마지막으로 OrderId를 가진 주문이 아직 없는 경우 주문을 생성합니다.

```
final String ORDER_PARTITION_KEY = "OrderId";
final String ORDER_TABLE_NAME = "Orders";

HashMap<String, AttributeValue> orderItem = new HashMap<>();
orderItem.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));
orderItem.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));
orderItem.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));
orderItem.put("OrderStatus", new AttributeValue("CONFIRMED"));
orderItem.put("OrderTotal", new AttributeValue("100"));

Put createOrder = new Put()
    .withTableName(ORDER_TABLE_NAME)
    .withItem(orderItem)

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD)
    .withConditionExpression("attribute_not_exists(" + ORDER_PARTITION_KEY + ")");
```

트랜잭션 실행

다음 예제는 앞에서 정의한 작업을 하나의 양자택일 방식 작업으로 실행하는 방법을 보여 줍니다.

```
    Collection<TransactWriteItem> actions = Arrays.asList(
        new TransactWriteItem().withConditionCheck(checkCustomerValid),
        new TransactWriteItem().withUpdate(markItemSold),
        new TransactWriteItem().withPut(createOrder));

    TransactWriteItemsRequest placeOrderTransaction = new TransactWriteItemsRequest()
        .withTransactItems(actions)
        .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

    // Run the transaction and process the result.
    try {
        client.transactWriteItems(placeOrderTransaction);
        System.out.println("Transaction Successful");

    } catch (ResourceNotFoundException rnf) {
        System.err.println("One of the table involved in the transaction is not found"
 + rnf.getMessage());
    } catch (InternalServerErrorException ise) {
        System.err.println("Internal Server Error" + ise.getMessage());
    } catch (TransactionCanceledException tce) {
        System.out.println("Transaction Canceled " + tce.getMessage());
```

```
    }
```

## 주문 세부 정보 읽기

다음 예제는 Orders 및 ProductCatalog 테이블에서 완료된 주문 트랜잭션을 읽는 방법을 보여줍니다.

```
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

HashMap<String, AttributeValue> orderKey = new HashMap<>();
orderKey.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));

Get readProductSold = new Get()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey);
Get readCreatedOrder = new Get()
    .withTableName(ORDER_TABLE_NAME)
    .withKey(orderKey);

Collection<TransactGetItem> getActions = Arrays.asList(
    new TransactGetItem().withGet(readProductSold),
    new TransactGetItem().withGet(readCreatedOrder));

TransactGetItemsRequest readCompletedOrder = new TransactGetItemsRequest()
    .withTransactItems(getActions)
    .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

// Run the transaction and process the result.
try {
    TransactGetItemsResult result = client.transactGetItems(readCompletedOrder);
    System.out.println(result.getResponses());
} catch (ResourceNotFoundException rnf) {
    System.err.println("One of the table involved in the transaction is not found" +
 rnf.getMessage());
} catch (InternalServerErrorException ise) {
    System.err.println("Internal Server Error" + ise.getMessage());
} catch (TransactionCanceledException tce) {
    System.err.println("Transaction Canceled" + tce.getMessage());
}
```

# Amazon DynamoDB를 사용하여 변경 데이터 캡처

많은 애플리케이션에서는 DynamoDB 테이블에 저장된 항목의 변경 내용을 해당 변경이 발생하는 시점에 캡처하는 기능을 활용할 수 있습니다. 다음은 사용 사례에 대한 몇 가지 예제입니다.

- 인기 있는 모바일 앱에서는 초당 수천 개 업데이트의 비율로 DynamoDB 테이블의 데이터를 수정합니다. 또 다른 애플리케이션에서 이러한 업데이트 사항을 캡처 후 저장하여 모바일 앱의 사용량을 거의 실시간으로 측정합니다.
- 재무 애플리케이션은 DynamoDB 테이블에서 주식 시장 데이터를 수정합니다. 병렬로 실행되는 여러 애플리케이션에서 이러한 변경 사항을 실시간으로 추적하고, 최대손실금액을 계산하고, 주가 변동에 따라 포트폴리오를 자동으로 리밸런스합니다.
- 운송 차량 및 산업용 장비의 센서에서 데이터를 DynamoDB 테이블로 전송합니다. 여러 애플리케이션에서 성능을 모니터링하고 문제가 감지되면 메시징 경고를 보내고, Machine Learning 알고리즘을 적용하여 잠재적 결함을 예측하고, 데이터를 압축하여 Amazon Simple Storage Service(Amazon S3)에 보관합니다.
- 그룹에 속한 어떤 친구가 사진을 새로 업로드하면 애플리케이션에서 그룹에 속한 모든 친구들에게 즉시 알림을 전송합니다.
- 새로운 고객이 생기면 DynamoDB 테이블에 데이터가 추가됩니다. 이러한 이벤트가 발생하면 새 고객에게 환영 이메일을 발송하는 애플리케이션이 호출됩니다.

DynamoDB는 항목 수준 변경 데이터 캡처 레코드의 스트리밍을 거의 실시간으로 지원합니다. 이러한 스트림을 소비하고 해당 내용을 바탕으로 조치를 취하는 애플리케이션을 빌드할 수 있습니다.

> **Note**
>
> DynamoDB Streams에 태그를 추가하고 DynamoDB Streams와 함께 속성 기반 액세스 제어(ABAC)를 사용하는 것은 지원되지 않습니다.

다음 비디오에서는 변경 데이터 캡처의 개념을 소개합니다.

[테이블 용량 모드]

주제

- 변경 데이터 캡처의 스트리밍 옵션
- Kinesis Data Streams를 사용하여 DynamoDB 변경 사항 캡처

- DynamoDB Streams에 대한 변경 데이터 캡처

## 변경 데이터 캡처의 스트리밍 옵션

DynamoDB는 변경 데이터 캡처를 위한 두 가지 스트리밍 모델인 DynamoDB용 Kinesis Data Streams와 DynamoDB Streams를 제공합니다.

애플리케이션에 적합한 솔루션 선택에 도움을 주기 위해 다음 표에서는 각 스트리밍 모델의 기능을 요약합니다.

| 속성 | DynamoDB용 Amazon Kinesis Data Streams | DynamoDB Streams |
| --- | --- | --- |
| 데이터 보존 | 최대 1년 | 24시간 |
| Kinesis Client Library(KCL) 지원 | KCL 버전 1.X, 2.X, 3.X 지원 | KCL 버전 1.X 및 2.X 지원 |
| 소비자 수 | 샤드당 최대 5명의 동시 소비자 또는 향상된 팬아웃을 사용하는 샤드당 최대 20명의 동시 소비자 | 샤드당 최대 2명의 동시 소비자 |
| 처리량 할당량 | 무제한. | DynamoDB 테이블 및 AWS 리전의 처리량 할당량에 따라 다름 |
| 레코드 전송 모델 | GetRecords를 사용하여 HTTP를 통해 모델을 풀하고 향상된 팬아웃을 사용하도록 설정하고 Kinesis Data Streams는 SubscribeToShard를 사용하여 HTTP/2를 통해 레코드를 푸시합니다. | GetRecords를 사용하여 HTTP를 통해 모델을 가져옵니다. |
| 레코드 순서 지정 | 각 스트림 레코드의 타임스탬프 속성을 사용하여 DynamoDB 테이블에서 변경이 | DynamoDB 테이블에서 수정된 각 항목의 스트림 레코드는 항 |

| 속성 | DynamoDB용 Amazon Kinesis Data Streams | DynamoDB Streams |
| --- | --- | --- |
| | 발생한 실제 순서를 식별할 수 있습니다. | 목의 실제 수정과 동일한 순서로 표시됩니다. |
| 중복 레코드 | 경우에 따라 중복 레코드가 스트림에 표시될 수 있습니다. | 중복 레코드가 스트림에 표시되지 않습니다. |
| 스트림 처리 옵션 | AWS Lambda, Amazon Managed Service for Apache Flink, Kinesis Data Firehose 또는 AWS Glue 스트리밍 ETL을 사용하여 스트림 레코드를 처리합니다. | AWS Lambda 또는 DynamoDB Streams Kinesis 어댑터를 사용하여 스트림 레코드를 처리합니다. |
| 내구성 수준 | 중단 없이 자동 장애 조치를 제공하는 가용 영역. | 중단 없이 자동 장애 조치를 제공하는 가용 영역. |

동일한 DynamoDB 테이블에서 두 스트리밍 모델을 모두 활성화할 수 있습니다.

다음 비디오에 두 옵션의 차이점이 자세히 나와 있습니다.

DynamoDB Streams와 Kinesis Data Streams

## Kinesis Data Streams를 사용하여 DynamoDB 변경 사항 캡처

Amazon Kinesis Data Streams를 사용하여 Amazon DynamoDB 변경 사항을 캡처할 수 있습니다.

Kinesis Data Streams는 모든 DynamoDB 테이블에서 항목 수준 수정 사항을 캡처하여 Kinesis 데이터 스트림에 복제합니다. 애플리케이션에서는 이 스트림에 액세스하고 항목 수준 변경 사항을 거의 실시간으로 볼 수 있습니다. 시간당 수 테라바이트의 데이터를 지속적으로 캡처하고 저장할 수 있습니다. 더 긴 데이터 보존 시간을 활용할 수 있으며, 향상된 팬아웃(fan-out) 기능을 통해 2개 이상의 다운스트림 애플리케이션에 동시에 액세스할 수 있습니다. 다른 이점으로는 추가적인 감사 및 보안 투명성이 있습니다.

Kinesis Data Streams는 Amazon Data Firehose 및 Amazon Managed Service for Apache Flink에 대한 액세스도 제공합니다. 이러한 서비스를 통해 실시간 대시보드를 지원하고, 알림을 생성하고, 동적

요금 및 광고를 구현하고, 정교한 데이터 분석 및 기계 학습 알고리즘을 구현하는 애플리케이션을 더 효과적으로 구축할 수 있습니다.

> **Note**
>
> DynamoDB용 Kinesis Data Streams를 사용하면 데이터 스트림의 Kinesis Data Streams 요금과 소스 테이블의 DynamoDB 요금이 모두 적용됩니다.

## Kinesis Data Streams가 DynamoDB에서 작동하는 방식

DynamoDB 테이블에 대해 Kinesis 데이터 스트림이 활성화되면 테이블은 해당 테이블의 데이터에 대한 변경 사항을 캡처하는 데이터 레코드를 전송합니다. 이 데이터 레코드에는 다음이 포함됩니다.

- 항목이 최근에 생성, 업데이트 또는 삭제된 특정 시간
- 해당 항목의 기본 키
- 수정 전 레코드의 스냅샷
- 수정 후 레코드의 스냅샷

이러한 데이터 레코드는 거의 실시간으로 캡처되고 게시됩니다. 데이터가 Kinesis 데이터 스트림에 기록되면 다른 레코드와 마찬가지로 읽을 수 있습니다. Kinesis 클라이언트 라이브러리와 AWS Lambda를 사용하고 Kinesis Data Streams API 및 기타 연결된 서비스를 호출할 수 있습니다. 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 Amazon Kinesis Data Streams에서 데이터 읽기를 참조하세요.

이러한 데이터 변경 사항도 비동기적으로 캡처됩니다. Kinesis는 데이터가 스트리밍되는 테이블의 성능에 영향을 미치지 않습니다. 또한 Kinesis 데이터 스트림에 저장된 스트림 레코드는 유휴 시 암호화됩니다. 자세한 내용은 Amazon Kinesis Data Streams의 데이터 보호를 참조하세요.

Kinesis 데이터 스트림 레코드는 항목 변경 사항이 발생했을 때와 다른 순서로 표시될 수 있습니다. 동일한 항목 알림이 스트림에 두 번 이상 표시될 수도 있습니다. `ApproximateCreationDateTime` 속성을 확인하여 항목 수정이 발생한 순서를 식별하고 중복 레코드를 식별할 수 있습니다.

Kinesis 데이터 스트림을 DynamoDB 테이블의 스트리밍 대상으로 활성화하면 `ApproximateCreationDateTime` 값의 정밀도를 밀리초 또는 마이크로초 단위로 구성할 수 있습니다. 기본적으로 `ApproximateCreationDateTime`은 변경 시간(밀리초)을 나타냅니다. 또한 활성 스트리밍 대상에서 이 값을 변경할 수 있습니다. 이러한 업데이트 후 Kinesis에 기록된 스트림 레코드는 원하는 정밀도의 `ApproximateCreationDateTime` 값을 갖게 됩니다.

DynamoDB에 기록되는 바이너리 값은 base64 인코딩 형식으로 인코딩되어야 합니다. 그러나 데이터 레코드가 Kinesis 데이터 스트림에 기록될 때 이러한 인코딩된 바이너리 값은 base64 인코딩 형식으로 다시 한 번 인코딩됩니다. Kinesis 데이터 스트림에서 이러한 레코드를 읽을 때 원시 바이너리 값을 검색하려면 애플리케이션에서 이러한 값을 두 번 디코딩해야 합니다.

DynamoDB에서는 Kinesis Data Streams 사용에 대해 변경 데이터 캡처 단위로 요금을 부과합니다. 단일 항목당 변경 1KB가 변경 데이터 캡처 단위 하나로 계산됩니다. 각 항목의 변경 KB는 쓰기 작업에 대한 용량 단위 소비와 동일한 로직을 사용하여 스트림에 기록된 항목의 '이전' 및 '이후' 이미지 중 더 큰 이미지를 기준으로 계산됩니다. DynamoDB 온디맨드 모드의 작동 방식과 유사하게 변경 데이터 캡처 단위의 용량 처리량을 프로비저닝할 필요가 없습니다.

DynamoDB 테이블에 대한 Kinesis 데이터 스트림 켜기

AWS Management Console, AWS SDK 또는AWS Command Line Interface(AWS CLI)를 사용하여 기존 DynamoDB 테이블에서 Kinesis로의 스트리밍을 활성화하거나 비활성화할 수 있습니다.

- 테이블과 동일한 AWS 계정 및 AWS 리전에서만 Amazon DynamoDB에서 Kinesis Data Streams으로 데이터를 스트리밍할 수 있습니다.
- DynamoDB 테이블에서 하나의 Kinesis 데이터 스트림으로만 데이터를 스트리밍할 수 있습니다.

DynamoDB 테이블의 Kinesis Data Streams 대상 변경

기본적으로 모든 Kinesis 데이터 스트림 레코드에는 `ApproximateCreationDateTime` 속성이 포함됩니다. 이 속성은 각 레코드가 생성된 대략적인 시간을 밀리초 단위로 표시한 타임스탬프를 나타냅니다. https://console.aws.amazon.com/kinesis, SDK 또는 AWS CLI를 사용하여 이러한 값의 정밀도를 변경할 수 있습니다.

## Amazon DynamoDB용 Kinesis Data Streams 시작하기

이 섹션에서는 Amazon DynamoDB용 Kinesis Data Streams 테이블을 Amazon DynamoDB 콘솔, AWS Command Line Interface(AWS CLI) 및 API와 함께 사용하는 방법을 설명합니다.

활성 Amazon Kinesis Data Streams 생성

다음의 모든 예에서는 DynamoDB 시작하기 자습서의 일부로 생성된 `Music` DynamoDB 테이블을 사용합니다.

소비자를 빌드하고 Kinesis 데이터 스트림을 다른 AWS 서비스에 연결하는 방법에 대한 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 Amazon Kinesis Data Streams에서 데이터 읽기를 참조하세요.

> **Note**
>
> KDS 샤드를 처음 사용할 때는 사용 패턴에 따라 샤드를 스케일 업 또는 스케일 다운하도록 설정하는 것이 좋습니다. 사용 패턴에 대한 데이터를 더 많이 축적한 후에는 스트림의 샤드를 이에 맞게 조정할 수 있습니다.

Console

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/kinesis/에서 Kinesis 콘솔을 엽니다.
2. Create data stream(데이터 스트림 생성)을 선택하고 지침에 따라 `samplestream`이라는 스트림을 생성합니다.
3. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
4. 콘솔 왼쪽의 탐색 창에서 테이블을 선택합니다.
5. Music 테이블을 선택합니다.
6. 내보내기 및 스트림(Exports and streams) 탭을 선택합니다.
7. (선택 사항) Amazon Kinesis 데이터 스트림 세부 정보에서 레코드 타임스탬프 정밀도를 마이크로초(기본값)에서 밀리초로 변경할 수 있습니다.
8. 드롭다운 목록에서 samplestream을 선택합니다.
9. 켜기 버튼을 선택합니다.

AWS CLI

1. create-stream command를 사용하여 `samplestream`이라는 Kinesis Data Streams를 생성합니다.

   ```
   aws kinesis create-stream --stream-name samplestream --shard-count 3
   ```

   Kinesis 데이터 스트림의 샤드 수를 설정하기 전에 Kinesis Data Streams에 대한 샤드 관리 고려 사항 섹션을 참조하세요.
2. describe-stream command를 사용하여 Kinesis 스트림이 활성 상태이고 사용할 준비가 되어 있는지 확인합니다.

   ```
   aws kinesis describe-stream --stream-name samplestream
   ```

3. DynamoDB `enable-kinesis-streaming-destination` 명령을 사용하여 DynamoDB 테이블에서 Kinesis 스트리밍을 활성화합니다. `stream-arn` 값은 이전 단계에 `describe-stream`에서 반환한 값으로 바꿉니다. 선택적으로 각 레코드에서 반환되는 타임스탬프 값을 마이크로초 단위로 더 정밀하게 설정하여 스트리밍을 활성화할 수 있습니다.

   마이크로초의 타임스탬프 정밀도로 스트리밍 활성화:

   ```
   aws dynamodb enable-kinesis-streaming-destination \
     --table-name Music \
     --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
     --enable-kinesis-streaming-configuration
    ApproximateCreationDateTimePrecision=MICROSECOND
   ```

   또는 기본 타임스탬프 정밀도(밀리초)로 스트리밍 활성화:

   ```
   aws dynamodb enable-kinesis-streaming-destination \
     --table-name Music \
     --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
   ```

4. DynamoDB `describe-kinesis-streaming-destination` 명령을 사용하여 테이블에서 Kinesis 스트리밍이 활성 상태인지 확인합니다.

   ```
   aws dynamodb describe-kinesis-streaming-destination --table-name Music
   ```

5. DynamoDB 개발자 안내서에 설명된 대로 `put-item` 명령을 사용하여 DynamoDB 테이블에 데이터를 씁니다.

   ```
   aws dynamodb put-item \
       --table-name Music  \
       --item \
           '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me
    Today"}, "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

   aws dynamodb put-item \
       --table-name Music \
       --item \
           '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
    "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"} }'
   ```

6. Kinesis get-records CLI 명령을 사용하여 Kinesis 스트림 콘텐츠를 검색합니다. 그 후 다음 코드 조각을 사용하여 스트림 콘텐츠를 역직렬화합니다.

```
/**
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we want to fetch the value
     * of this attribute from the new item image. The following code fetches
 this value.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
    System.out.println(attributeValue);
}

private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
}
```

Java

1. Kinesis Data Streams 개발자 안내서의 지침에 따라 Java를 사용하여 `samplestream`이라는 Kinesis 데이터 스트림을 생성합니다.

Kinesis 데이터 스트림의 샤드 수를 설정하기 전에 Kinesis Data Streams에 대한 샤드 관리 고려 사항 섹션을 참조하세요.

2. 다음 코드 조각을 사용하여 DynamoDB 테이블에서 Kinesis 스트리밍을 활성화합니다. 선택적으로 각 레코드에서 반환되는 타임스탬프 값을 마이크로초 단위로 더 정밀하게 설정하여 스트리밍을 활성화할 수 있습니다.

   마이크로초의 타임스탬프 정밀도로 스트리밍 활성화:

```
EnableKinesisStreamingConfiguration enableKdsConfig =
 EnableKinesisStreamingConfiguration.builder()

 .approximateCreationDateTimePrecision(ApproximateCreationDateTimePrecision.MICROSECOND)
  .build();

EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .enableKinesisStreamingConfiguration(enableKdsConfig)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

   또는 기본 타임스탬프 정밀도(밀리초)로 스트리밍 활성화:

```
EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

3. Kinesis Data Streams 개발자 안내서의 지침에 따라 생성한 데이터 스트림에서 읽어옵니다.

4. 다음 코드 조각을 사용하여 스트림 콘텐츠를 역직렬화합니다.

```
/**
```

```
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we wanted to fetch the value
     * of this attribute from the new item image, the below code would fetch
 this.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
    System.out.println(attributeValue);
}

private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
}
```

### 활성 Amazon Kinesis 데이터 스트림에 변경 사항 적용

이 섹션에서는 콘솔, AWS CLI 및 API를 사용하여 DynamoDB에 대한 활성 Kinesis Data Streams 설정을 변경하는 방법을 설명합니다.

#### AWS Management Console

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.

2. 테이블로 이동합니다.
3. 내보내기 및 스트림을 선택합니다.

AWS CLI

1. describe-kinesis-streaming-destination을 직접 호출하여 스트림이 ACTIVE임을 확인합니다.
2. 다음 예와 같이 UpdateKinesisStreamingDestination을 직접 호출합니다.

```
aws dynamodb update-kinesis-streaming-destination --table-name
 enable_test_table --stream-arn arn:aws:kinesis:us-east-1:12345678901:stream/
enable_test_stream --update-kinesis-streaming-configuration
 ApproximateCreationDateTimePrecision=MICROSECOND
```

3. describe-kinesis-streaming-destination을 직접 호출하여 스트림이 UPDATING임을 확인합니다.
4. 스트리밍 상태가 다시 ACTIVE가 될 때까지 주기적으로 describe-kinesis-streaming-destination을 직접 호출합니다. 타임스탬프 정밀도 업데이트가 적용되려면 일반적으로 최대 5분이 걸립니다. 이 상태가 업데이트되면 업데이트가 완료되었으며 새 정밀도 값이 향후 레코드에 적용될 것임을 나타냅니다.
5. putItem을 사용하여 테이블에 기록합니다.
6. Kinesis get-records 명령을 사용하여 스트림 콘텐츠를 가져옵니다.
7. 쓰기의 ApproximateCreationDateTime 정밀도가 원하는 수준인지 확인합니다.

Java API

1. UpdateKinesisStreamingDestination 요청 및 UpdateKinesisStreamingDestination 응답을 구성하는 코드 스니펫을 제공합니다.
2. DescribeKinesisStreamingDestination 요청 및 DescribeKinesisStreamingDestination response를 구성하는 코드 스니펫을 제공합니다.
3. 스트리밍 상태가 다시 ACTIVE가 될 때까지 주기적으로 describe-kinesis-streaming-destination을 직접 호출합니다. 이 상태는 업데이트가 완료되고 향후 레코드에 새 정밀도 값이 적용될 것임을 나타냅니다.
4. 테이블에 대한 쓰기를 수행합니다.

5. 스트림에서 읽고 스트림 콘텐츠를 역직렬화합니다.
6. 쓰기의 `ApproximateCreationDateTime` 정밀도가 원하는 수준인지 확인합니다.

## DynamoDB Streams 및 Kinesis Data Streams로 샤드 및 지표 사용

### Kinesis Data Streams에 대한 샤드 관리 고려 사항

Kinesis 데이터 스트림은 샤드로 처리량을 계산합니다. Amazon Kinesis Data Streams에서 데이터 스트림에 대해 온디맨드 모드와 프로비저닝된 모드 중에서 선택할 수 있습니다.

DynamoDB 쓰기 워크로드가 매우 가변적이고 예측할 수 없는 경우 Kinesis Data Stream에 온디맨드 모드를 사용하는 것이 좋습니다. 온디맨드 모드를 사용하면 Kinesis Data Streams가 필요한 처리량을 제공하기 위해 샤드를 자동으로 관리하므로 용량 계획이 필요하지 않습니다.

예측 가능한 워크로드의 경우 Kinesis Data Stream에 프로비저닝 모드를 사용할 수 있습니다. 프로비저닝된 모드에서는 DynamoDB의 데이터 캡처 레코드 변화에 맞춰 데이터 스트림의 샤드 수를 지정해야 합니다. Kinesis 데이터 스트림이 DynamoDB 테이블을 지원하는 데 필요한 샤드 수를 확인하려면 다음 입력 값이 필요합니다.

- DynamoDB 테이블 레코드의 평균 크기(바이트)(`average_record_size_in_bytes`)
- 초당 DynamoDB 테이블에서 수행하는 쓰기 작업의 최대 수. 여기에는 애플리케이션에서 수행하는 생성, 삭제 및 업데이트 작업뿐만 아니라 Time To Live(TTL)로 생성된 삭제 작업(`write_throughput`)과 같이 자동으로 생성된 작업도 포함됩니다.
- 생성 또는 삭제 작업 대비 테이블에서 수행하는 업데이트 및 덮어쓰기 작업의 백분율(`percentage_of_updates`). 업데이트 및 덮어쓰기 작업은 수정된 항목의 이전 이미지와 새 이미지를 모두 스트림에 복제한다는 점에 유의하세요. 따라서 DynamoDB 항목의 2배 크기 레코드가 생성됩니다.

다음 형식의 입력 값을 사용하여 Kinesis 데이터 스트림에 필요한 샤드 수(`number_of_shards`)를 계산할 수 있습니다.

```
number_of_shards = ceiling( max( ((write_throughput * (4+percentage_of_updates) *
 average_record_size_in_bytes) / 1024 / 1024), (write_throughput/1000)), 1)
```

예를 들어 평균 레코드 크기가 800바이트(`average_record_size_in_bytes`))일 때 최대 처리량이 초당 1,040회의 쓰기 작업(`write_throughput`)일 수 있습니다. 해당 쓰기 작업의 25%가 업데이트 작업(`percentage_of_updates`)인 경우 DynamoDB 스트리밍 처리량을 수용하기 위해 2개의 샤드(`number_of_shards`)가 필요합니다.

```
ceiling( max( ((1040 * (4+25/100) * 800)/ 1024 / 1024), (1040/1000)), 1).
```

공식을 사용하여 Kinesis 데이터 스트림의 프로비저닝 모드에서 필요한 샤드 수를 계산하기 전에 다음 사항을 고려하세요.

- 이 공식은 DynamoDB 변경 데이터 레코드를 수용하는 데 필요한 샤드 수를 추정하는 데 도움이 됩니다. 추가 Kinesis 데이터 스트림 소비자를 지원하는 데 필요한 샤드 수와 같이 Kinesis 데이터 스트림에 필요한 샤드의 전체 수를 나타내지 않습니다.
- 최대 처리량을 처리하도록 데이터 스트림을 구성하지 않으면 프로비저닝된 모드에서 읽기 및 쓰기 처리량 예외가 계속 발생할 수 있습니다. 이 경우 데이터 트래픽을 수용할 수 있도록 데이터 스트림 규모를 수동으로 조정해야 합니다.
- 이 공식은 변경 로그 데이터 레코드를 Kinesis Data Stream으로 스트리밍하기 전에 DynamoDB에서 생성되는 추가 팽창을 고려합니다.

Kinesis Data Stream의 용량 모드에 대해 자세히 알아보려면 Choosing the Data Stream Capacity Mode를 참조하세요. 각기 다른 용량 모드 간의 요금 차이에 대해 자세히 알아보려면 Amazon Kinesis Data Streams 요금을 참조하세요.

Kinesis Data Streams를 사용하여 변경 데이터 캡처 모니터링

DynamoDB는 Kinesis로 변경 데이터 캡처 복제를 모니터링하는 데 도움이 되는 몇 가지 Amazon CloudWatch 지표를 제공합니다. 전체 CloudWatch 지표 목록은 DynamoDB 지표 및 차원 섹션을 참조하세요.

스트림의 용량이 충분한지 확인하려면 스트림을 활성화하는 중과 프로덕션 단계 모두에 다음 항목을 모니터링하는 것이 좋습니다.

- `ThrottledPutRecordCount`: Kinesis 데이터 스트림 용량이 부족하여 Kinesis 데이터 스트림에 의해 제한된 레코드 수입니다. 예외적으로 사용량이 최고조에 달할 때 약간의 제한이 발생할 수 있지만, `ThrottledPutRecordCount`는 가능한 한 낮게 유지되어야 합니다. DynamoDB에서 제한된 레코드를 Kinesis 데이터 스트림으로 보내려고 다시 시도하지만, 이로 인해 복제 대기 시간이 증가할 수 있습니다.

  제한이 지나치게 많고 자주 발생하는 경우 관찰된 테이블의 쓰기 처리량에 비례하여 Kinesis 스트림 샤드 수를 늘려야 할 수 있습니다. Kinesis 데이터 스트림의 크기 결정에 대한 자세한 내용은 Kinesis 데이터 스트림의 초기 크기 결정을 참조하세요.
- `AgeOfOldestUnreplicatedRecord`: 지금까지 Kinesis 데이터 스트림에 복제할 항목 수준 변경 중 가장 오래된 내용이 DynamoDB 테이블에 표시된 이후 경과된 시간. 정상 작동 시

`AgeOfOldestUnreplicatedRecord`는 밀리초 단위여야 합니다. 이 수는 실패한 복제 시도가 고객이 제어하는 구성 선택으로 인해 발생한 경우 시도 횟수를 기준으로 증가합니다.

`AgeOfOldestUnreplicatedRecord` 지표가 168시간을 초과하는 경우 DynamoDB 테이블에서 Kinesis 데이터 스트림으로의 항목 수준 변경 복제가 자동으로 비활성화됩니다.

복제 시도 실패로 이어질 수 있는 고객이 제어하는 구성의 예로는 Kinesis 데이터 스트림 용량이 과소 프로비저닝되어 과도한 제한이 발생하는 경우 또는 Kinesis 데이터 스트림의 액세스 정책을 수동으로 업데이트하여 DynamoDB가 데이터 스트림에 데이터를 추가하지 못하는 경우가 있습니다. 이 지표를 가능한 한 낮게 유지하려면 적절한 Kinesis 데이터 스트림 용량을 프로비저닝하고 DynamoDB의 권한이 변경되지 않도록 해야 할 수 있습니다.

- `FailedToReplicateRecordCount`: DynamoDB가 Kinesis 데이터 스트림으로 복제하지 못한 레코드 수입니다. 34KB보다 큰 특정 항목은 Kinesis Data Streams의 1MB 항목 크기 제한보다 큰 데이터 레코드를 변경하기 위해 크기가 확장될 수 있습니다. 이 크기 확장은 34KB보다 큰 이러한 항목에 많은 수의 부울 또는 빈 속성 값을 포함할 때 발생합니다. 부울 및 빈 속성 값은 DynamoDB에 1바이트로 저장되지만, Kinesis Data Streams 복제를 위한 표준 JSON을 사용하여 직렬화되면 최대 5바이트까지 확장합니다. DynamoDB는 이러한 변경 레코드를 Kinesis 데이터 스트림에 복제할 수 없습니다. DynamoDB는 이러한 변경 데이터 레코드를 건너뛰고 자동으로 후속 레코드의 복제를 계속합니다.

앞의 지표 중 하나가 특정 임계값을 초과할 경우 알리기 위해 Amazon Simple Notification Service(Amazon SNS) 메시지를 보내는 Amazon CloudWatch 경보를 생성할 수 있습니다.

## Amazon Kinesis Data Streams 및 Amazon DynamoDB에 대한 IAM 정책 사용

Amazon DynamoDB용 Amazon Kinesis Data Streams를 처음 활성화하면, DynamoDB는 사용자를 위한 AWS Identity and Access Management(IAM) 서비스 연결 역할을 자동으로 생성합니다. 이 역할 `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`을 사용하면 DynamoDB가 사용자 대신 Kinesis Data Streams으로의 항목 수준 변경 사항 복제를 관리할 수 있습니다. 이 서비스 연결 역할을 삭제하지 마세요.

서비스 연결 역할에 대한 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 사용을 참조하세요

> **Note**
>
> DynamoDB는 IAM 정책에 대한 태그 기반 조건을 지원하지 않습니다.

Amazon DynamoDB용 Amazon Kinesis Data Streams를 활성화하려면 테이블에 대한 다음 권한이 있어야 합니다.

- `dynamodb:EnableKinesisStreamingDestination`
- `kinesis:ListStreams`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`

지정된 DynamoDB 테이블에 대한 Amazon DynamoDB용 Amazon Kinesis Data Streams를 설명하려면 테이블에 대한 다음 권한이 있어야 합니다.

- `dynamodb:DescribeKinesisStreamingDestination`
- `kinesis:DescribeStreamSummary`
- `kinesis:DescribeStream`

Amazon DynamoDB용 Amazon Kinesis Data Streams를 비활성화하려면 테이블에 대한 다음 권한이 있어야 합니다.

- `dynamodb:DisableKinesisStreamingDestination`

Amazon DynamoDB용 Amazon Kinesis Data Streams를 업데이트하려면 테이블에 대한 다음 권한이 있어야 합니다.

- `dynamodb:UpdateKinesisStreamingDestination`

다음 예제에서는 IAM 정책을 사용하여 Amazon DynamoDB용 Amazon Kinesis Data Streams에 대한 권한을 부여하는 방법을 보여 줍니다.

예제: Amazon DynamoDB용 Amazon Kinesis Data Streams 활성화

다음 IAM 정책은 `Music` 테이블에 대해 Amazon DynamoDB용 Amazon Kinesis Data Streams를 활성화하는 권한을 부여합니다. `Music` 테이블에 대한 DynamoDB용 Kinesis Data Streams를 비활성화, 업데이트 또는 설명하는 권한은 부여하지 않습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/kinesisreplication.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBKinesisDataStreamsReplication",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "kinesisreplication.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

예: Amazon DynamoDB용 Amazon Kinesis Data Streams 업데이트

다음 IAM 정책은 `Music` 테이블에 대해 Amazon DynamoDB용 Amazon Kinesis Data Streams를 업데이트하는 권한을 부여합니다. `Music` 테이블에 대한 Amazon DynamoDB용 Amazon Kinesis Data Streams를 활성화, 비활성화 또는 설명하는 권한은 부여하지 않습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

예제: Amazon DynamoDB용 Amazon Kinesis Data Streams 비활성화

다음 IAM 정책은 `Music` 테이블에 대해 Amazon DynamoDB용 Amazon Kinesis Data Streams를 비활성화하는 권한을 부여합니다. `Music` 테이블에 대한 Amazon DynamoDB용 Amazon Kinesis Data Streams를 활성화, 업데이트 또는 설명하는 권한은 부여하지 않습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

예: 리소스를 기반으로 Amazon DynamoDB용 Amazon Kinesis Data Streams에 대한 권한을 선택적으로 적용

다음 IAM 정책은 `Music` 테이블에 대해 Amazon DynamoDB용 Amazon Kinesis Data Streams를 활성화할 권한을 부여하고 `Orders` 테이블에 대해 Amazon DynamoDB용 Amazon Kinesis Data Streams를 비활성할 권한을 거부합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination",
                "dynamodb:DescribeKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Orders"
        }
    ]
}
```

DynamoDB용 Kinesis Data Streams에 대한 서비스 연결 역할 사용

Amazon DynamoDB용 Amazon Kinesis Data Streams는 AWS Identity and Access Management(IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 DynamoDB용 Kinesis Data Streams에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 DynamoDB용 Kinesis Data Streams에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 DynamoDB용 Kinesis Data Streams를 더 쉽게 설정할 수 있습니다. DynamoDB용 Kinesis Data Streams는 서비스 연결 역할의 권한을 정의하며 달리 정의하지 않는 한 DynamoDB용 Kinesis Data Streams만 해당 역할을 맡을 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 IAM으로 작업하는 AWS 서비스를 참조해 서비스 연결 역할(Service-Linked Role) 열이 예(Yes)인 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 링크가 있는 예를 선택합니다.

### DynamoDB용 Kinesis Data Streams에 대한 서비스 연결 역할 권한

DynamoDB용 Kinesis Data Streams는 AWSServiceRoleForDynamoDBKinesisDataStreamsReplication라는 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 Amazon DynamoDB가 사용자 대신 Kinesis Data Streams에 대한 항목 수준 변경 사항의 복제를 관리할 수 있도록 하기 위한 것입니다.

`AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `kinesisreplication.dynamodb.amazonaws.com`

역할 권한 정책은 DynamoDB용 Kinesis Data Streams가 지정된 리소스에서 다음 작업을 완료하도록 허용합니다.

- 작업: `Kinesis stream`에 대한 `Put records and describe`
- 작업: 사용자 생성 AWS KMS 키를 사용하여 암호화된 Kinesis 스트림에 데이터를 저장하기 위해 `AWS KMS`에서 `Generate data keys`

정책 문서의 정확한 내용은 DynamoDBKinesisReplicationServiceRolePolicy를 참조하세요.

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한 섹션을 참조하세요.

### DynamoDB용 Kinesis Data Streams에 대한 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWSAPI에서 DynamoDB용 Kinesis Data Streams를 활성화하면 DynamoDB용 Kinesis Data Streams는 사용자를 위한 서비스 연결 역할을 생성합니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. DynamoDB용 Kinesis Data Streams를 활성화하면 DynamoDB용 Kinesis Data Streams는 사용자를 위한 서비스 연결 역할을 다시 생성합니다.

### DynamoDB용 Kinesis Data Streams에 대한 서비스 연결 역할 편집

DynamoDB용 Amazon Kinesis Data Streams에서는 `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때

문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

DynamoDB용 Kinesis Data Streams에 대한 서비스 연결 역할 삭제

또한 IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 서비스 연결 역할을 수동으로 삭제할 수 있습니다. 단, 서비스 연결 역할에 대한 리소스를 먼저 정리해야 수동으로 삭제할 수 있습니다.

> **Note**
>
> 리소스를 삭제하려 할 때 DynamoDB용 Kinesis Data Streams 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

IAM을 사용하여 수동으로 서비스 연결 역할을 삭제하려면 다음을 수행하세요.

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하세요.

## DynamoDB Streams에 대한 변경 데이터 캡처

DynamoDB Streams는 DynamoDB 테이블에서 시간 순서에 따라 항목 수준 수정을 캡처하고 이 정보를 최대 24시간 동안 로그에 저장합니다. 로그와 데이터 항목은 변경 전후 거의 실시간으로 나타나므로 애플리케이션에서 이러한 로그와 데이터에 액세스할 수 있습니다.

유휴 시 암호화는 DynamoDB Streams의 데이터를 암호화합니다. 자세한 내용은 DynamoDB 저장 데이터 암호화 섹션을 참조하세요.

DynamoDB 스트림은 DynamoDB 테이블 항목의 변경 사항에 대한 정렬된 정보 흐름입니다. 테이블에서 스트림을 활성화하면 DynamoDB가 테이블 데이터 항목의 모든 수정에 대한 정보를 캡처합니다.

애플리케이션에서 테이블 항목을 생성, 업데이트 또는 삭제할 때마다 DynamoDB Streams는 수정된 항목의 기본 키 속성을 사용하여 스트림 레코드를 작성합니다. 스트림 레코드에는 DynamoDB 테이블의 단일 항목에 대한 데이터 수정 정보가 포함되어 있습니다. 수정된 항목의 "이전" 및 "이후" 이미지 등과 같이 스트림 레코드에서 추가 정보를 캡처하도록 스트림을 구성할 수 있습니다.

DynamoDB Streams는 다음을 보장할 수 있습니다.

- 각 스트림 기록은 스트림에서 한 번만 나타납니다.

- DynamoDB 테이블에서 수정된 각 항목의 스트림 레코드는 항목의 실제 수정과 동일한 순서로 표시됩니다.

DynamoDB Streams는 거의 실시간으로 스트림 레코드를 작성하므로 이러한 스트림을 소비하고 내용을 바탕으로 조치를 취할 수 있는 애플리케이션을 빌드할 수 있습니다.

주제


- DynamoDB Streams에 대한 엔드포인트
- 스트림 활성화
- 스트림 판독 및 처리
- DynamoDB Streams 및 유지 시간(TTL)
- DynamoDB Streams Kinesis 어댑터를 사용하여 스트림 레코드 처리
- DynamoDB Streams 하위 수준 API: Java 예시
- DynamoDB Streams 및 AWS Lambda 트리거
- DynamoDB 스트림 및 Apache Flink


## DynamoDB Streams에 대한 엔드포인트

AWS에서는 DynamoDB 및 DynamoDB Streams의 엔드포인트가 별개입니다. 데이터베이스 테이블 및 인덱스를 사용하려면 애플리케이션이 DynamoDB 엔드포인트에 액세스해야 합니다. DynamoDB Streams 레코드를 읽고 처리하려면 애플리케이션이 동일한 리전의 DynamoDB Streams 엔드포인트에 액세스해야 합니다.

DynamoDB Streams 엔드포인트의 이름은 `streams.dynamodb.<region>.amazonaws.com` 형식으로 지정되어야 합니다. 예를 들어 `dynamodb.us-west-2.amazonaws.com` 엔드포인트를 사용하여 DynamoDB에 액세스하는 경우 `streams.dynamodb.us-west-2.amazonaws.com` 엔드포인트를 사용하여 DynamoDB Streams에 액세스합니다.

> **Note**
>
> DynamoDB 및 DynamoDB Streams 리전 및 엔드포인트의 전체 목록은 AWS 일반 참조의 리전 및 엔드포인트를 참조하세요.

AWS SDK는 DynamoDB와 DynamoDB Streams의 클라이언트를 따로 제공합니다. 요구 사항에 따라 애플리케이션은 DynamoDB 엔드포인트, DynamoDB Streams 엔드포인트 또는 두 엔드포인트

에 동시에 액세스할 수 있습니다. 두 엔드포인트에 모두 연결하려면 애플리케이션이 2개 클라이언트(DynamoDB용 클라이언트 하나와 DynamoDB Streams용 클라이언트 하나)를 인스턴스화해야 합니다.

## 스트림 활성화

AWS CLI 또는 AWS SDK 중 하나를 사용하여 새 테이블을 생성할 때 테이블에서 스트림을 활성화할 수 있습니다. 또한 기존 테이블에서도 스트림을 활성화하거나 비활성화 할 수 있으며, 스트림 설정도 변경할 수 있습니다. DynamoDB Streams는 비동기식으로 작동하므로 스트림을 활성화하더라도 테이블의 성능에 영향을 미치지 않습니다.

DynamoDB Streams를 관리하는 가장 용이한 방법은 AWS Management Console을 사용하는 것입니다.

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. DynamoDB 콘솔 대시보드에서 Tables(테이블)를 선택하고 기존 테이블을 선택합니다.
3. 내보내기 및 스트림(Exports and streams) 탭을 선택합니다.
4. DynamoDB 스트림 세부 정보 섹션에서 켜기를 선택합니다.
5. DynamoDB 스트림 켜기 페이지에서 테이블의 데이터가 수정될 때마다 스트림에 기록할 정보를 선택합니다.

   - 키 속성만(Key attributes only) - 수정된 항목의 키 속성만을 표시합니다.
   - 새로운 이미지 - 항목의 수정 후 전체 모습을 보여 줍니다.
   - 이전 이미지 - 항목의 수정 전 전체 모습을 보여 줍니다.
   - 새 이미지와 이전 이미지 - 항목의 새 이미지와 이전 이미지를 모두 보여 줍니다.

   원하는 대로 설정되었으면 스트림 켜기를 선택합니다.
6. (선택 사항) 기존 스트림을 비활성화하려면 DynamoDB 스트림 세부 정보 아래의 끄기를 선택합니다.

또한 `CreateTable` 또는 `UpdateTable` API 작업을 사용하여 스트림을 활성화하거나 수정할 수도 있습니다. `StreamSpecification` 파라미터는 스트림 구성 방식을 결정합니다.

- `StreamEnabled` - 테이블에서 스트림을 활성화(`true`)할지 비활성화(`false`)할지 여부를 지정합니다.

- `StreamViewType` - 테이블의 데이터가 수정될 때마다 스트림에 쓸 정보를 지정합니다.
  - `KEYS_ONLY` - 수정된 항목의 키 속성만 표시합니다.
  - `NEW_IMAGE` - 항목의 수정 후 전체 모습을 보여 줍니다.
  - `OLD_IMAGE` - 항목의 수정 전 전체 모습을 보여 줍니다.
  - `NEW_AND_OLD_IMAGES` - 항목의 새 이미지와 이전 이미지를 모두 보여 줍니다.

언제든지 스트림을 활성화 또는 비활성화할 수 있습니다. 그러나 이미 스트림을 가지고 있는 테이블에서 스트림을 활성화하려고 하면 `ValidationException`이 수신됩니다. 또한 스트림을 가지고 있지 않은 테이블에서 스트림을 비활성화하려고 하면 `ValidationException`이 수신됩니다.

`StreamEnabled`를 `true`로 설정하면 DynamoDB가 배정된 스트림 서술자를 사용하여 새 스트림을 생성합니다. 테이블에서 스트림을 비활성화한 후 다시 활성화하면 다른 스트림 서술자를 갖는 새 스트림이 생성됩니다.

모든 스트림은 Amazon 리소스 이름(ARN)을 통해 고유 식별됩니다. 아래에는 이름이 `TestTable`인 DynamoDB 테이블에 위치한 스트림의 ARN이 예시되어 있습니다.

```
arn:aws:dynamodb:us-west-2:111122223333:table/TestTable/stream/2015-05-11T21:21:33.291
```

테이블의 가장 최근 스트림 서술자를 확인하려면 DynamoDB `DescribeTable` 요청을 실행한 뒤 응답에서 `LatestStreamArn` 요소를 찾습니다.

> **Note**
>
> 스트림을 설정한 후에는 `StreamViewType`을 편집할 수 없습니다. 스트림을 설정한 후 변경해야 하는 경우 현재 스트림을 비활성화하고 새 스트림을 생성해야 합니다.

## 스트림 판독 및 처리

스트림을 판독하고 처리하려면 애플리케이션이 DynamoDB Streams 엔드포인트에 연결되어 API 요청을 실행해야 합니다.

스트림은 스트림 기록으로 구성되어 있습니다. 각 스트림 레코드는 스트림이 속한 DynamoDB 테이블에서의 단일 데이터 수정을 나타냅니다. 각 스트림 레코드에는 레코드가 스트림에 게시되는 순서를 반영하는 시퀀스 번호가 할당됩니다.

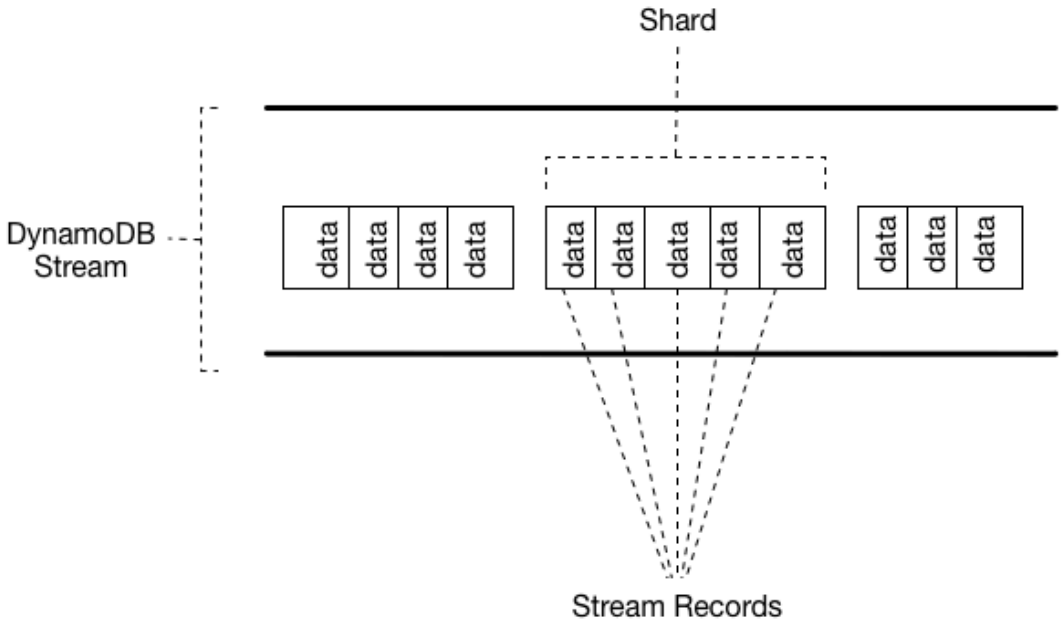
스트림 기록은 그룹 또는 샤드로 구성되어 있습니다. 각 샤드는 다중 스트림 레코드 저장소 역할을 하며 이러한 기록 액세스 및 반복 처리에 필요한 정보를 담고 있습니다. 샤드 내 스트림 레코드는 24시간 후 자동으로 제거됩니다.

샤드는 한시적입니다. 필요에 따라 자동으로 생성되었다가 삭제됩니다. 또한 모든 샤드는 여러 개의 새로운 샤드로 분할될 수 있으며 이 프로세스도 자동으로 이루어집니다. (상위 샤드가 하위 하드를 하나만 가질 수 있다는 점도 유의하세요.) 상위 테이블에서 쓰기 활동이 매우 활발할 경우 애플리케이션이 여러 샤드로부터 동시에 레코드를 처리할 수 있도록 샤드가 분할될 수 있습니다.

스트림을 비활성화하면 열려 있는 샤드가 모두 닫힙니다. 스트림의 데이터는 24시간 동안 읽기 가능한 상태로 유지됩니다.

샤드에는 계보(상위 및 하위)가 있으므로 애플리케이션은 항상 상위 샤드를 하위 샤드보다 먼저 처리해야 합니다. 그래야 스트림 레코드도 올바른 순서로 처리됩니다. (DynamoDB Streams Kinesis 어댑터를 사용하는 경우 이 프로세스가 자동으로 처리됩니다. 애플리케이션이 올바른 순서로 샤드와 스트림 레코드를 처리하고 자동으로 새로운 또는 만료된 샤드, 그리고 애플리케이션이 실행되는 동안 분할된 샤드를 처리합니다. 자세한 내용은 DynamoDB Streams Kinesis 어댑터를 사용하여 스트림 레코드 처리 섹션을 참조하세요.)

스트림, 스트림 샤드 및 샤드에 포함된 스트림 기록 간 관계가 다음 도표에 나타나 있습니다.



> **Note**
>
> 항목 안의 어떠한 데이터도 변경하지 않는 `PutItem` 또는 `UpdateItem` 작업을 수행하면 DynamoDB Streams는 해당 작업에 대하여 스트림 레코드를 작성하지 않습니다.

스트림에 액세스하여 그 안에 포함된 스트림 기록을 처리하는 과정은 다음과 같습니다.

- 액세스하고자 하는 스트림의 고유 ARN을 선택합니다.
- 처리하고자 하는 스트림 레코드가 포함된 샤드를 스트림에서 선택합니다.
- 샤드에 액세스한 뒤 원하는 스트림 레코드를 조회합니다.

> **Note**
>
> 최대 2개의 프로세스까지 동일한 스트림의 샤드에서 동시에 읽을 수 있습니다. 샤드당 읽기 프로세스가 2개를 초과하면 병목이 발생할 수 있습니다.

DynamoDB Streams API에서는 애플리케이션 프로그램에서 사용하기 위한 다음과 같은 작업을 제공합니다.

- `ListStreams` - 현재 계정 및 엔드포인트에 대한 스트림 서술자 목록을 반환합니다. 선택 사항으로 특정 테이블 이름에 대하여 스트림 서술자만 요청할 수 있습니다.
- `DescribeStream` — 스트림의 현재 상태, Amazon 리소스 이름(ARN), 샤드의 구성 및 해당되는 DynamoDB 테이블을 포함하여 스트림에 대한 정보를 반환합니다. 선택적으로 `ShardFilter` 필드를 사용하여 상위 샤드와 연결된 기존 하위 샤드를 검색할 수 있습니다.
- `GetShardIterator` - 샤드 내 위치를 설명하는 샤드 반복자를 반환합니다. 반복자가 스트림의 가장 오래된 지점, 최신 지점 및 특정 지점에 대한 액세스를 제공하도록 요청할 수 있습니다.
- `GetRecords` - 지정된 샤드 내에서 스트림 레코드를 반환합니다. `GetShardIterator` 요청으로부터 반환된 샤드 반복자를 제공해야 합니다.

요청 및 응답 예제를 포함하여 이러한 API 작업에 대한 전체 설명은 Amazon DynamoDB Streams API 참조를 참조하세요.

샤드 검색

두 가지 강력한 방법을 사용하여 DynamoDB 스트림에서 새 샤드를 검색합니다. Amazon DynamoDB Streams 사용자는 다음 두 가지 효과적인 방법으로 새 샤드를 추적하고 식별할 수 있습니다.

전체 스트림 토폴로지 폴링

`DescribeStream` API를 사용하여 스트림을 정기적으로 폴링합니다. 그러면 생성된 새 샤드를 포함하여 스트림의 모든 샤드가 반환됩니다. 시간 경과에 따른 결과를 비교하면 새로 추가된 샤드를 감지할 수 있습니다.

하위 샤드 검색

`ShardFilter` 파라미터와 함께 `DescribeStream` API를 사용하여 샤드의 하위 집합을 찾습니다. 요청에 상위 샤드를 지정하면 DynamoDB Streams가 바로 밑에 있는 하위 샤드를 반환합니다. 이 접근 방식은 전체 스트림을 스캔하지 않고 샤드 계보만 추적하면 되는 경우에 유용합니다.

DynamoDB Streams에서 데이터를 사용하는 애플리케이션은 `ShardFilter` 파라미터를 사용하여 닫힌 샤드를 읽는 것에서 하위 샤드로 효율적으로 전환할 수 있으므로 모든 닫힌 샤드와 열린 샤드에 대한 샤드 맵을 검색하고 통과하기 위해 `DescribeStream` API를 반복적으로 직접 호출하지 않아도 됩니다. 이렇게 하면 상위 샤드가 닫힌 후 하위 샤드를 빠르게 검색하여 스트림 처리 애플리케이션의 응답성과 비용 효과를 높일 수 있습니다.

두 방법 모두 DynamoDB Streams의 진화하는 구조를 계속해서 추적할 수 있으므로 중요한 데이터 업데이트 또는 샤드 수정을 놓치지 않을 수 있도록 합니다.

DynamoDB Streams에 대한 데이터 보존 제한

DynamoDB Streams의 모든 데이터는 24시간 동안 유지됩니다. 특정 테이블에 대한 지난 24시간 동안의 활동을 조회하고 분석할 수 있습니다. 그러나 24시간이 지난 데이터는 언제든 트리밍(제거)될 수 있습니다.

테이블에서 스트림을 비활성화해도 스트림에 포함된 데이터는 24시간 동안 읽기 가능한 상태가 유지됩니다. 이 시긴 이후 데이터는 만료되며 스트림 기록은 자동으로 삭제됩니다. 기존 스트림을 수동으로 삭제하기 위한 메커니즘은 없습니다. 보유 제한이 만료(24시간)될 때까지 기다려야 하며, 모든 스트림 레코드가 삭제됩니다.

## DynamoDB Streams 및 유지 시간(TTL)

테이블에서 Amazon DynamoDB Streams를 활성화하고 만료된 항목의 스트림 레코드를 처리하여 유지 시간(TTL)에 의해 삭제된 항목을 백업하거나 처리할 수 있습니다. 자세한 내용은 스트림 판독 및 처리 섹션을 참조하세요.

스트림 레코드에는 사용자 ID 필드 `Records[<index>].userIdentity`가 포함되어 있습니다.

만료 후 유지 시간(TTL) 프로세스에 의해 삭제된 항목에는 다음 필드가 있습니다.

- `Records[<index>].userIdentity.type`

  `"Service"`
- `Records[<index>].userIdentity.principalId`

  `"dynamodb.amazonaws.com"`

> **Note**
>
> 글로벌 테이블에서 TTL을 사용하는 경우 TTL이 수행된 리전에 `userIdentity` 필드가 설정됩니다. 삭제가 복제될 때 다른 리전에서는 이 필드가 설정되지 않습니다.

다음 JSON은 단일 스트림 레코드의 해당 부분을 보여 줍니다.

```
"Records": [
    {
        ...

        "userIdentity": {
            "type": "Service",
            "principalId": "dynamodb.amazonaws.com"
        }

        ...

    }
]
```

DynamoDB Streams 및 Lambda를 사용하여 TTL 삭제 항목 보관

DynamoDB 유지 시간(TTL), DynamoDB Streams 및 AWS Lambda를 결합하면 데이터 보관을 간소화하고 DynamoDB 스토리지 비용을 절감하며 코드 복잡성을 줄일 수 있습니다. Lambda를 스트림 소비자로 사용할 경우 많은 이점이 있습니다. 특히 Kinesis Client Library(KCL)와 같은 여타 소비자에 비해 비용이 절감됩니다. Lambda를 사용하여 이벤트를 소비할 때 DynamoDB 스트림에서 `GetRecords` API 호출에 대한 요금이 부과되지 않으며, Lambda는 스트림 이벤트에서 JSON 패턴을 식별하여 이벤

트 필터링을 제공할 수 있습니다. 이벤트 패턴 콘텐츠 필터링을 통해 최대 5가지 필터를 정의하여 처리를 위해 Lambda로 전송되는 이벤트를 제어할 수 있습니다. 이렇게 함으로써 Lambda 함수의 호출 수가 감소하고 코드가 간소화되며 전체 비용이 절감됩니다.

DynamoDB Streams에는 `Create`, `Modify` 및 `Remove` 작업과 같은 모든 데이터 수정 사항이 포함되어 있지만 이로 인해 아카이브 Lambda 함수가 원치 않게 호출될 수 있습니다. 예를 들어 시간당 200만 개의 데이터 수정 사항이 스트림으로 유입되는 테이블이 있는데 이 중 5% 미만이 TTL 프로세스를 통해 만료되고 보관해야 하는 항목 삭제라고 가정해 보겠습니다. Lambda 이벤트 소스 필터를 사용하면 Lambda 함수가 시간당 10만 회만 호출합니다. 이벤트 필터링의 결과로, 이벤트 필터링 없이 수행될 200만 회의 호출 대신 필요한 호출에 대해서만 요금이 부과됩니다.

이벤트 필터링은 선택한 이벤트(DynamoDB 스트림)에서 읽기를 수행하고 Lambda 함수를 호출하는 리소스인 Lambda 이벤트 소스 매핑에 적용됩니다. 다음 다이어그램에서는 스트림 및 이벤트 필터를 사용하여 Lambda 함수에 의해 유지 시간(TTL) 삭제 항목이 소비되는 방법을 볼 수 있습니다.




DynamoDB 유지 시간(TTL) 이벤트 필터 패턴

이벤트 소스 매핑 필터 기준에 다음 JSON을 추가하면 TTL 삭제 항목에 대해서만 Lambda 함수를 호출할 수 있습니다.

```
{
    "Filters": [
        {
            "Pattern": { "userIdentity": { "type": ["Service"], "principalId":
 ["dynamodb.amazonaws.com"] } }
        }
    ]
}
```

AWS Lambda 이벤트 소스 매핑 생성

다음 코드 조각을 사용하여 테이블의 DynamoDB 스트림에 연결할 수 있는 필터링된 이벤트 소스 매핑을 생성합니다. 각 코드 블록에는 이벤트 필터 패턴이 포함됩니다.

AWS CLI

```
aws lambda create-event-source-mapping \
```

```
--event-source-arn 'arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000' \
--batch-size 10 \
--enabled \
--function-name test_func \
--starting-position LATEST \
--filter-criteria '{"Filters": [{"Pattern": "{\"userIdentity\":{\"type\":[\"Service
\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}"}]}'
```

Java

```
LambdaClient client = LambdaClient.builder()
        .region(Region.EU_WEST_1)
        .build();

Filter userIdentity = Filter.builder()
        .pattern("{\"userIdentity\":{\"type\":[\"Service\"],\"principalId\":
[\"dynamodb.amazonaws.com\"]}}")
        .build();

FilterCriteria filterCriteria = FilterCriteria.builder()
        .filters(userIdentity)
        .build();

CreateEventSourceMappingRequest mappingRequest =
 CreateEventSourceMappingRequest.builder()
        .eventSourceArn("arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000")
        .batchSize(10)
        .enabled(Boolean.TRUE)
        .functionName("test_func")
        .startingPosition("LATEST")
        .filterCriteria(filterCriteria)
        .build();

try{
    CreateEventSourceMappingResponse eventSourceMappingResponse =
 client.createEventSourceMapping(mappingRequest);
    System.out.println("The mapping ARN is
 "+eventSourceMappingResponse.eventSourceArn());

}catch (ServiceException e){
    System.out.println(e.getMessage());
```

```
}
```

### Node

```
const client = new LambdaClient({ region: "eu-west-1" });

const input = {
    EventSourceArn: "arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000",
    BatchSize: 10,
    Enabled: true,
    FunctionName: "test_func",
    StartingPosition: "LATEST",
    FilterCriteria: { "Filters": [{ "Pattern": "{\"userIdentity\":{\"type\":
[\"Service\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}" }] }
}

const command = new CreateEventSourceMappingCommand(input);

try {
    const results = await client.send(command);
    console.log(results);
} catch (err) {
    console.error(err);
}
```

### Python

```
session = boto3.session.Session(region_name = 'eu-west-1')
client = session.client('lambda')

try:
    response = client.create_event_source_mapping(
        EventSourceArn='arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000',
        BatchSize=10,
        Enabled=True,
        FunctionName='test_func',
        StartingPosition='LATEST',
        FilterCriteria={
            'Filters': [
                {
```

```
                    'Pattern': "{\"userIdentity\":{\"type\":[\"Service\"],
\"principalId\":[\"dynamodb.amazonaws.com\"]}}"
                },
            ]
        }
    )
    print(response)
except Exception as e:
    print(e)
```

JSON

```
{
  "userIdentity": {
     "type": ["Service"],
     "principalId": ["dynamodb.amazonaws.com"]
   }
}
```

## DynamoDB Streams Kinesis 어댑터를 사용하여 스트림 레코드 처리

Amazon Kinesis 어댑터 사용은 Amazon DynamoDB의 스트림을 소비할 때 권장되는 방법입니다. DynamoDB Streams API는 Kinesis Data Streams와 유사합니다. 두 서비스 모두 데이터 스트림이 샤드로 구성되어 있습니다. 샤드란 스트림 레코드의 컨테이너입니다. 두 서비스의 API 모두 Kinesis Data Streams의 API와 의도적으로 유사하게 설계된 DynamoDB Streams API를 포함하고 있습니다. 두 서비스 모두 데이터 스트림이 샤드로 구성되어 있습니다. 샤드란 스트림 레코드의 컨테이너입니다. 두 서비스의 API에는 `ListStreams`, `DescribeStream`, `GetShards` 및 `GetShardIterator` 작업이 포함되어 있습니다. (이러한 DynamoDB Streams 작업은 Kinesis Data Streams의 해당 작업과 유사하지만 100% 동일하지는 않습니다.)

DynamoDB Streams 사용자라면 Kinesis Client Library(KCL) 내에서 찾을 수 있는 설계 패턴을 활용해 DynamoDB Streams 샤드와 스트림 레코드를 처리할 수 있습니다. 이렇게 하려면 DynamoDB Streams Kinesis 어댑터를 사용합니다. Kinesis 어댑터는 DynamoDB Streams의 레코드를 사용 및 처리하는 데 KCL을 사용할 수 있도록 Kinesis Data Streams 인터페이스를 구현합니다. DynamoDB Streams Kinesis 어댑터를 설정하고 설치하는 방법에 대한 지침은 GitHub 리포지토리를 참조하세요.

DynamoDB Streams 사용자는 KCL에 있는 디자인 패턴을 활용하여 DynamoDB Streams 샤드와 스트림 레코드를 처리할 수 있습니다. 이렇게 하려면 DynamoDB Streams Kinesis 어댑터를 사용합니다. Kinesis 어댑터는 DynamoDB Streams의 레코드를 사용 및 처리하는 데 KCL을 사용할 수 있도록

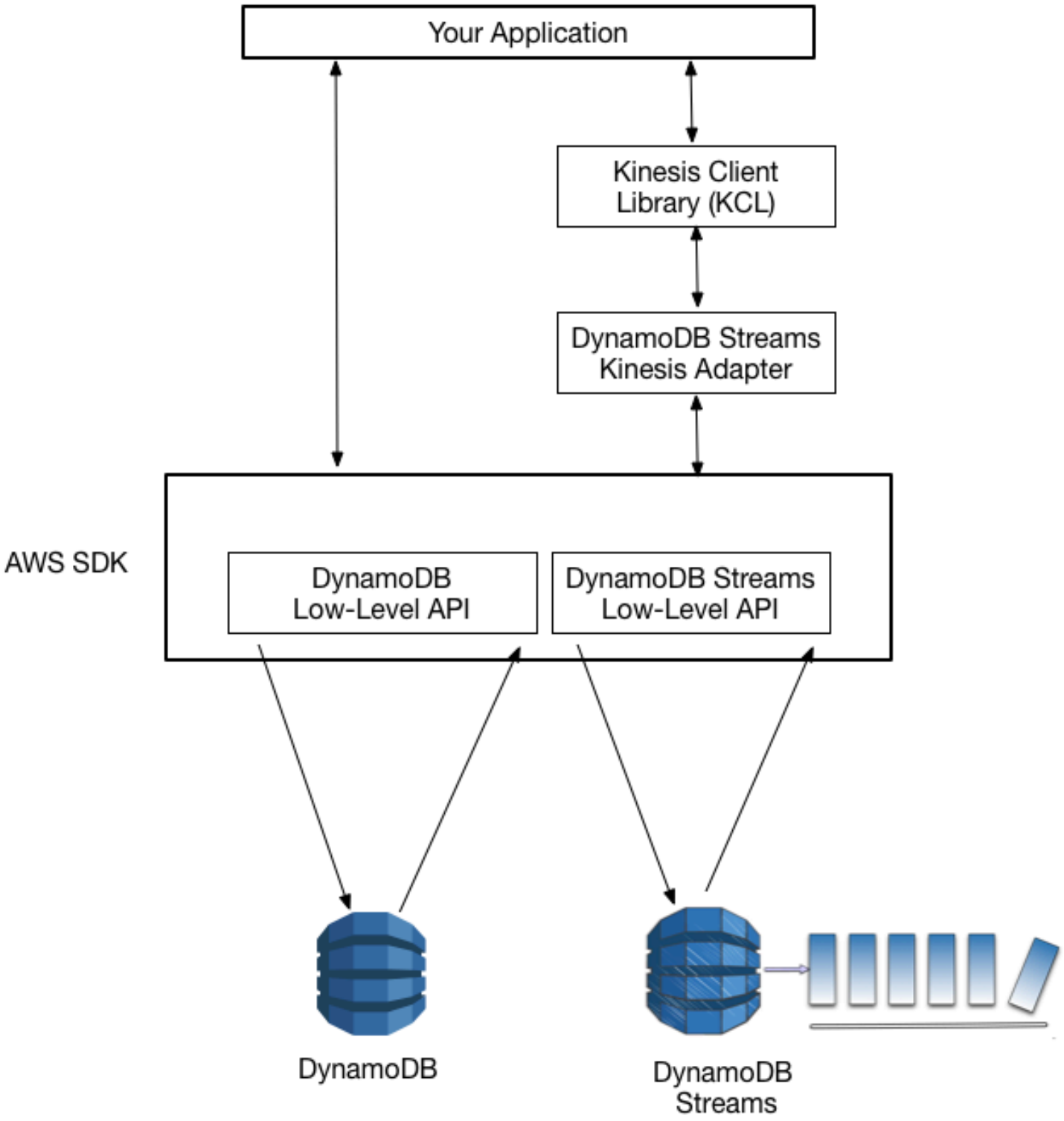
Kinesis Data Streams 인터페이스를 구현합니다. DynamoDB Streams Kinesis 어댑터를 설정하고 설치하는 방법에 대한 지침은 GitHub 리포지토리를 참조하세요.

Kinesis Client Library(KCL)를 사용하여 Kinesis Data Streams의 애플리케이션을 작성할 수 있습니다. KCL은 하위 수준의 Kinesis Data Streams API에 유용한 추상화를 제공하여 코딩을 단순화합니다. KCL에 대한 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 Kinesis Client Library를 사용하여 소비자 개발을 참조하세요.

DynamoDB는 AWS SDK for Java v2.x와 함께 KCL 버전 3.x를 사용할 것을 권장합니다. AWS SDK for Java v1.x를 사용하는 현재 KCL 버전 1.x는 AWS SDK 및 도구 유지 관리 정책에 따라 전환 기간 동안 의도한 대로 수명 주기 전체에 걸쳐 계속 완전히 지원됩니다.

> **Note**
>
> Amazon Kinesis Client Library(KCL) 버전 1.x 및 2.x는 오래되었습니다. KCL 1.x는 2026년 1월 30일에 지원이 종료됩니다. 버전 1.x를 사용하는 KCL 애플리케이션을 2026년 1월 30일 이전에 최신 KCL 버전으로 마이그레이션하는 것이 좋습니다. 최신 KCL 버전을 찾으려면 GitHub의 Amazon Kinesis Client Library 페이지를 참조하세요. 최신 KCL 버전에 대한 자세한 내용은 Kinesis Client Library 사용을 참조하세요. KCL 1.x에서 KCL 3.x로의 마이그레이션에 대한 자세한 내용은 KCL 1.x에서 KCL 3.x로의 마이그레이션을 참조하세요.

다음 다이어그램은 이러한 라이브러리가 서로 상호 작용하는 방법을 보여 줍니다.




DynamoDB Streams Kinesis 어댑터가 준비되어 있으면 DynamoDB Streams 엔드포인트로 원활하게 전달되는 API 호출을 통해 KCL 인터페이스 개발을 시작할 수 있습니다.

애플리케이션이 시작되면 KCL을 호출하여 작업자를 인스턴스화합니다. 작업자에게 애플리케이션의 구성 정보를 제공해야 합니다. 제공해야 하는 구성 정보에는 스트림 서술자와 AWS 자격 증명, 제공하는 레코드 프로세서 클래스의 이름 등이 있습니다. 레코드 프로세서에서 코드를 실행하면 작업자는 다음 작업을 수행합니다.

- 스트림에 연결합니다
- 스트림 내 샤드를 열거합니다.
- 스트림 내에서 닫힌 상위 샤드의 하위 샤드를 확인하고 열거
- 스트림 내에서 닫힌 상위 샤드의 하위 샤드를 확인하고 열거
- 샤드 연결을 다른 작업자(있는 경우)와 조정합니다.
- 관리하는 모든 샤드의 레코드 프로세서를 인스턴스화합니다
- 스트림에서 레코드를 가져옵니다.
- 해당하는 레코드 프로세서로 레코드를 푸시합니다
- 처리된 레코드에 대해 체크포인트를 수행합니다
- 작업자 인스턴스 수가 변경되면 샤드-작업자 연결을 조정합니다
- 샤드가 분할되면 샤드-작업자 연결을 조정합니다.

> **Note**
>
> 여기에 나온 KCL 개념에 대한 설명은 Amazon Kinesis Data Streams 개발자 안내서의 Kinesis Client Library를 사용하여 소비자 개발을 참조하세요.
> AWS Lambda에 스트림을 사용하는 방법에 대한 자세한 내용은 DynamoDB Streams 및 AWS Lambda 트리거 섹션을 참조하세요.

KCL 1.x에서 KCL 3.x로 마이그레이션

개요

이 가이드에서는 소비자 애플리케이션을 KCL 1.x에서 KCL 3.x로 마이그레이션하기 위한 지침을 제공합니다. KCL 1.x와 KCL 3.x의 아키텍처 차이로 인해 마이그레이션하려면 호환성을 보장하기 위해 여러 구성 요소를 업데이트해야 합니다.

KCL 1.x는 KCL 3.x와 다른 클래스 및 인터페이스를 사용합니다. KCL 1.x는 KCL 3.x와 다른 클래스 및 인터페이스를 사용합니다. 먼저 레코드 프로세서, 레코드 프로세서 팩토리 및 워커 클래스를 KCL 3.x 호환 형식으로 마이그레이션하고 KCL 1.x에서 KCL 3.x로 마이그레이션하는 단계를 따라야 합니다.

마이그레이션 단계

주제


- 1단계: 레코드 프로세서 마이그레이션
- 2단계: 레코드 프로세서 팩토리 마이그레이션
- 3단계: 워커 마이그레이션
- 4단계: KCL 3.x 구성 개요 및 권장 사항
- 5단계: KCL 2.x에서 KCL 3.x로 마이그레이션


## 1단계: 레코드 프로세서 마이그레이션

다음은 KCL 1.x DynamoDB Streams Kinesis 어댑터에 구현된 레코드 프로세서를 보여주는 예제입니다.

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownInput;

public class StreamsRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
    @Override
    public void initialize(InitializationInput initializationInput) {
        //
        // Setup record processor
        //
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        for (Record record : processRecordsInput.getRecords()) {
```

```
            String data = new String(record.getData().array(),
 Charset.forName("UTF-8"));
            System.out.println(data);
            if (record instanceof RecordAdapter) {
                // record processing and checkpointing logic
            }
        }
    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @Override
    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
        try {
            checkpointer.checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow exception
            //
            e.printStackTrace();
        }
    }
}
```

RecordProcessor 클래스를 마이그레이션하려면

1. 다음과 같이 인터페이스를 `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` 및 `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat` 서 `software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordP` 변경합니다.

```
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;

import
 software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
```

2. `initialize` 및 `processRecords` 메서드에 대한 import 문을 업데이트합니다.

```
// import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import software.amazon.kinesis.lifecycle.events.InitializationInput;

// import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import
 software.amazon.dynamodb.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
```

3. `shutdown` 메서드를 새 메서드인 `leaseLost`, `shardEnded`, `shutdownRequested`으로 바꿉니다.

```
//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shardEnded(...) and
 shutdownRequested(ShutdownReauestedInput)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
```

```
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
```

다음은 업데이트된 버전의 레코드 프로세서 클래스입니다.

```
package com.amazonaws.codesamples;

import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import
 software.amazon.dynamodb.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import
 software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor;
import
 software.amazon.dynamodb.streamsadapter.adapter.DynamoDBStreamsKinesisClientRecord;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
```

```
import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import software.amazon.awssdk.services.dynamodb.model.Record;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {

    @Override
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
    public void processRecords(DynamoDBStreamsProcessRecordsInput processRecordsInput)
 {
        for (DynamoDBStreamsKinesisClientRecord record: processRecordsInput.records())
            Record ddbRecord = record.getRecord();
            // processing and checkpointing logic for the ddbRecord
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
```

```
            //
            e.printStackTrace();
        }
    }
}
```

> **Note**
>
> DynamoDB Streams Kinesis Adapter는 이제 SDKv2 Record 모델을 사용합니다. SDKv2에서 복합 `AttributeValue` 객체(BS, NS, M, L, SS)는 null을 반환하지 않습니다. `hasBs()`, `hasNs()`, `hasM()`, `hasL()`, `hasSs()` 메서드를 사용하여 이러한 값이 존재하는지 확인합니다.

2단계: 레코드 프로세서 팩토리 마이그레이션

레코드 프로세스 팩토리는 리스가 필요할 경우 레코드 프로세서 생성을 담당합니다. 다음은 KCL 1.x 팩토리의 예입니다.

```
package com.amazonaws.codesamples;

import software.amazon.dynamodb.AmazonDynamoDB;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements IRecordProcessorFactory {

    @Override
    public IRecordProcessor createProcessor() {
        return new StreamsRecordProcessor(dynamoDBClient, tableName);
    }
}
```

**`RecordProcessorFactory`**를 마이그레이션하려면

- 구현된 인터페이스를 다음과 같이 `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa`서 `software.amazon.kinesis.processor.ShardRecordProcessorFactory`로 변경합니다.

```
com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory
 to software.amazon.kinesis.processor.ShardRecordProcessorFactory, as follows.

// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

// public class TestRecordProcessorFactory implements IRecordProcessorFactory {
public class StreamsRecordProcessorFactory implements ShardRecordProcessorFactory {

Change the return signature for createProcessor.

// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

다음은 3.0의 레코드 프로세서 팩토리의 예입니다.

```
package com.amazonaws.codesamples;

import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements ShardRecordProcessorFactory {

    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new StreamsRecordProcessor();
    }
}
```

3단계: 워커 마이그레이션

KCL 버전 3.0에서는 Scheduler라는 새로운 클래스가 Worker 클래스를 대체합니다. 다음은 KCL 1.x 워커의 예입니다.

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
```

```
final Worker worker = StreamsWorkerFactory.createDynamoDbStreamsWorker(
        recordProcessorFactory,
        workerConfig,
        adapterClient,
        amazonDynamoDB,
        amazonCloudWatchClient);
```

작업자를 마이그레이션하려면

1. Worker 클래스에 대한 import 문을 Scheduler 및 ConfigsBuilder 클래스에 대한 가져오기 문으로 변경합니다.

```
// import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.common.ConfigsBuilder;
```

2. StreamTracker를 가져오고 StreamsWorkerFactory의 가져오기를 StreamsSchedulerFactory로 변경합니다.

```
import software.amazon.kinesis.processor.StreamTracker;
// import software.amazon.dynamodb.streamsadapter.StreamsWorkerFactory;
import software.amazon.dynamodb.streamsadapter.StreamsSchedulerFactory;
```

3. 애플리케이션을 시작할 위치를 선택합니다. TRIM_HORIZON 또는 LATEST일 수 있습니다.

```
import software.amazon.kinesis.common.InitialPositionInStream;
import software.amazon.kinesis.common.InitialPositionInStreamExtended;
```

4. StreamTracker 인스턴스를 만듭니다.

```
StreamTracker streamTracker = StreamsSchedulerFactory.createSingleStreamTracker(
        streamArn,

 InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON)
);
```

5. AmazonDynamoDBStreamsAdapterClient 객체를 생성합니다.

```
import software.amazon.dynamodb.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
```

```
...

AwsCredentialsProvider credentialsProvider = DefaultCredentialsProvider.create();

AmazonDynamoDBStreamsAdapterClient adapterClient = new
 AmazonDynamoDBStreamsAdapterClient(
        credentialsProvider, awsRegion);
```

6. `ConfigsBuilder` 객체를 생성합니다.

```
import software.amazon.kinesis.common.ConfigsBuilder;

...
ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                applicationName,
                adapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                UUID.randomUUID().toString(),
                new StreamsRecordProcessorFactory());
```

7. 다음 예와 같이 `Scheduler`를 생성합니다.

```
import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;

import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;

...


DynamoDbAsyncClient dynamoClient =
 DynamoDbAsyncClient.builder().region(region).build();
CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();
```

```
DynamoDBStreamsPollingConfig pollingConfig = new
 DynamoDBStreamsPollingConfig(adapterClient);
pollingConfig.idleTimeBetweenReadsInMillis(idleTimeBetweenReadsInMillis);

RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
retrievalConfig.retrievalSpecificConfig(pollingConfig);

CoordinatorConfig coordinatorConfig = configsBuilder.coordinatorConfig();
coordinatorConfig.clientVersionConfig(CoordinatorConfig.ClientVersionConfig.CLIENT_VERSION_

Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                coordinatorConfig,
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
                adapterClient
        );
```

> **Important**
>
> `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` 설정은 DynamoDB Streams Kinesis Adapter KCL v2와 v3가 아닌 KCL v3와 KCL v1 간의 호환성을 유지합니다.

4단계: KCL 3.x 구성 개요 및 권장 사항

KCL 1.x 이후에 도입된 KCL 3.x 관련 구성에 대한 자세한 설명은 KCL 구성 및 KCL 마이그레이션 클라이언트 구성을 참조하세요.

KCL 3.x에서 업데이트 기본값을 사용한 구성

**`billingMode`**

KCL 버전 1.x에서는 `billingMode`의 기본값이 `PROVISIONED`로 설정됩니다. 그러나 KCL 버전 3.x에서는 기본 `billingMode`가 `PAY_PER_REQUEST`(온디맨드 모드)입니다. 사용량에 따라 용량을 자동으로 조정하려면 리스 테이블에 온디맨드 용량 모드를 사용하는 것이 좋습니다. 리스 테이

블에 프로비저닝된 용량을 사용하는 방법에 대한 지침은 프로비저닝된 용량 모드를 사용하는 리스 테이블의 모범 사례를 참조하세요.

**`idleTimeBetweenReadsInMillis`**

KCL 버전 1.x에서는 `idleTimeBetweenReadsInMillis`의 기본값이 1,000(또는 1초)으로 설정됩니다. KCL 버전 3.x에서는 `idleTimeBetweenReadsInMillis`의 기본값이 1,500(또는 1.5초)으로 설정되지만 Amazon DynamoDB Streams Kinesis Adapter는 이 기본값을 1,000(또는 1초)으로 재정의합니다.

KCL 3.x의 새로운 구성

**`leaseAssignmentIntervalMillis`**

이 구성은 새로 검색된 샤드가 처리를 시작하기 전의 시간 간격을 정의하며, 1.5 × `leaseAssignmentIntervalMillis`로 계산됩니다. 이 설정을 명시적으로 구성하지 않으면 시간 간격은 기본적으로 1.5 × `failoverTimeMillis`로 설정됩니다. 새 샤드를 처리하려면 리스 테이블을 스캔하고 리스 테이블에서 글로벌 보조 인덱스(GSI)를 쿼리해야 합니다. `leaseAssignmentIntervalMillis`를 낮추면 이러한 스캔 및 쿼리 작업의 빈도가 증가하여 DynamoDB 비용이 증가합니다. 새 샤드 처리 지연을 최소화하려면 이 값을 2,000으로 설정하는 것이 좋습니다.

**`shardConsumerDispatchPollIntervalMillis`**

이 구성은 상태 전환을 트리거하기 위해 샤드 소비자가 연속 폴링하는 간격을 정의합니다. KCL 버전 1.x에서 이 동작은 구성 가능한 설정으로 노출되지 않은 `idleTimeInMillis` 파라미터에 의해 제어되었습니다. KCL 버전 3.x에서는 KCL 버전 1.x 설정에서 `idleTimeInMillis`에 사용된 값과 일치하도록 이 구성을 설정하는 것이 좋습니다.

Amazon DynamoDB Streams Kinesis Adapter에서 재정의되지 않는 구성

**`shardSyncIntervalMillis`**

KCL 버전 1.x와 호환되는 DynamoDB Streams Kinesis Adapter는 명시적으로 `shardSyncIntervalMillis`를 0으로 설정합니다. 반면, KCL 버전 3.x와 호환되는 DynamoDB Streams Kinesis Adapter는 더 이상 이 구성에 대한 값을 설정하지 않습니다. 버전 1.x와 동일한 어댑터 동작을 유지하려면 이 구성의 값을 0으로 설정합니다.

**`leasesRecoveryAuditorExecutionFrequencyMillis`**

KCL 버전 1.x와 호환되는 DynamoDB Streams Kinesis Adapter는 명시적으로 `leasesRecoveryAuditorExecutionFrequencyMillis`를 1,000으로 설정합니다. 반면, KCL 버전 3.x와 호환되는 DynamoDB Streams Kinesis Adapter는 더 이상 이 구성에 대한 기본값을 설정하지 않습니다. 버전 1.x와 동일한 어댑터 동작을 유지하려면 이 구성의 값을 1,000으로 설정합니다.

5단계: KCL 2.x에서 KCL 3.x로 마이그레이션

원활한 전환과 최신 Kinesis Client Library(KCL) 버전과의 호환성을 보장하려면 KCL 2.x에서 KCL 3.x로 업그레이드하기 위한 마이그레이션 가이드의 지침 5~8단계를 따르세요.

이전 KCL 버전으로 롤백

이 주제에서는 소비자 애플리케이션을 이전 KCL 버전으로 롤백하는 방법을 설명합니다. 롤백 프로세스는 다음 두 단계로 구성됩니다.

1. KCL 마이그레이션 도구를 실행합니다.
2. 이전 KCL 버전 코드를 재배포합니다.

1단계: KCL 마이그레이션 도구 실행

이전 KCL 버전으로 롤백해야 하는 경우 KCL 마이그레이션 도구를 실행해야 합니다. 이 도구는 다음 두 가지 중요한 작업을 수행합니다.

- DynamoDB의 리스 테이블에서 워커 지표 테이블이라고 하는 메타데이터 테이블과 글로벌 보조 인덱스를 제거합니다. 이러한 아티팩트는 KCL 3.x에서 생성되지만 이전 버전으로 롤백하는 경우 필요하지 않습니다.
- 모든 워커가 KCL 1.x와 호환되는 모드로 실행되고 이전 KCL 버전에서 사용된 로드 밸런싱 알고리즘을 사용하도록 합니다. KCL 3.x의 새 로드 밸런싱 알고리즘에 문제가 있는 경우 해당 문제를 즉시 완화합니다.

> **Important**
>
> DynamoDB의 조정자 상태 테이블은 반드시 존재해야 하며 마이그레이션, 롤백, 롤포워드 프로세스 중에 삭제되어서는 안 됩니다.

> **Note**
>
> 소비자 애플리케이션의 모든 워커가 지정된 시간에 동일한 로드 밸런싱 알고리즘을 사용하는 것이 중요합니다. KCL 마이그레이션 도구를 사용하면 KCL 3.x 소비자 애플리케이션의 모든 워커가 KCL 1.x 호환 모드로 전환되므로 이전 KCL 버전으로 애플리케이션을 롤백하는 동안 모든 워커가 동일한 로드 밸런싱 알고리즘을 실행하게 됩니다.

KCL GitHub 리포지토리의 스크립트 디렉터리에서 KCL 마이그레이션 도구를 다운로드할 수 있습니다. 조정자 상태 테이블, 워커 지표 테이블 및 리스 테이블에 쓸 수 있는 적절한 권한이 있는 워커 또는 호스트에서 스크립트를 실행합니다. KCL 소비자 애플리케이션에 대해 적절한 IAM 권한이 구성되어 있는지 확인합니다. 지정된 명령을 사용하여 KCL 애플리케이션당 한 번만 스크립트를 실행합니다.

```
python3 ./KclMigrationTool.py --region region --mode rollback [--
application_name applicationName] [--lease_table_name leaseTableName]
 [--coordinator_state_table_name coordinatorStateTableName] [--
worker_metrics_table_name workerMetricsTableName]
```

파라미터

`--region`

*##*을 사용자의 AWS 리전으로 바꿉니다.

`--application_name`

이 파라미터는 DynamoDB 메타데이터 테이블(리스 테이블, 조정자 상태 테이블 및 워커 지표 테이블)의 기본 이름을 사용하는 경우 필요합니다. 이러한 테이블에 사용자 지정 이름을 지정한 경우 이 파라미터를 생략할 수 있습니다. *applicationName*을 실제 KCL 애플리케이션 이름으로 바꿉니다. 이 도구는 사용자 지정 이름이 제공되지 않은 경우 이 이름을 사용하여 기본 테이블 이름을 파생합니다.

`--lease_table_name`

이 파라미터는 KCL 구성에서 리스 테이블에 사용자 지정 이름을 설정한 경우에 필요합니다. 기본 테이블 이름을 사용하는 경우 이 파라미터를 생략할 수 있습니다. *leaseTableName*을 리스 테이블에 지정한 사용자 지정 테이블 이름으로 바꿉니다.

`--coordinator_state_table_name`

이 파라미터는 KCL 구성에서 조정자 상태 테이블에 사용자 지정 이름을 설정한 경우에 필요합니다. 기본 테이블 이름을 사용하는 경우 이 파라미터를 생략할 수 있습니다.

*coordinatorStateTableName*을 조정자 상태 테이블에 지정한 사용자 지정 테이블 이름으로 바꿉니다.

`--worker_metrics_table_name`

이 파라미터는 KCL 구성에서 워커 지표 테이블에 사용자 지정 이름을 설정한 경우에 필요합니다. 기본 테이블 이름을 사용하는 경우 이 파라미터를 생략할 수 있습니다. *workerMetricsTableName*을 워커 지표 테이블에 지정한 사용자 지정 테이블 이름으로 바꿉니다.

2단계: 이전 KCL 버전으로 코드 재배포

> **Important**
>
> KCL 마이그레이션 도구에서 생성된 출력에 버전 2.x가 언급된 경우 KCL 버전 1.x를 의미하는 것으로 해석해야 합니다. 스크립트를 실행해도 전체 롤백이 수행되는 것은 아니며, 로드 밸런싱 알고리즘만 KCL 버전 1.x에서 사용된 알고리즘으로 전환됩니다.

롤백을 위해 KCL 마이그레이션 도구를 실행하면 다음 메시지 중 하나가 표시됩니다.

메시지 1

"Rollback completed. Your application was running 2x compatible functionality. Please rollback to your previous application binaries by deploying the code with your previous KCL version."

필요한 작업: 이는 워커가 KCL 1.x 호환 모드로 실행 중이었음을 의미합니다. 이전 KCL 버전으로 워커에 코드를 재배포합니다.

메시지 2

"Rollback completed. Your KCL Application was running 3x functionality and will rollback to 2x compatible functionality. If you don't see mitigation after a short period of time, please rollback to your previous application binaries by deploying the code with your previous KCL version."

필요한 작업: 이는 워커가 KCL 3.x 모드로 실행 중이었고 KCL 마이그레이션 도구가 모든 워커를 KCL 1.x 호환 모드로 전환했음을 의미합니다. 이전 KCL 버전으로 워커에 코드를 재배포합니다.

메시지 3

"Application was already rolled back. Any KCLv3 resources that could be deleted were cleaned up to avoid charges until the application can be rolled forward with migration."

필요한 작업: 이는 워커가 KCL 1.x 호환 모드로 실행되도록 이미 롤백되었음을 의미합니다. 이전 KCL 버전으로 워커에 코드를 재배포합니다.

### 롤백 후 KCL 3.x로 롤포워드

이 주제에서는 롤백 후 소비자 애플리케이션을 KCL 3.x로 롤포워드하는 방법을 설명합니다. 롤포워드가 필요한 경우 2단계 프로세스를 완료해야 합니다.

1. KCL 마이그레이션 도구를 실행합니다.
2. KCL 3.x로 코드를 배포합니다.

#### 1단계: KCL 마이그레이션 도구 실행

다음 명령으로 KCL 마이그레이션 도구를 실행하여 KCL 3.x로 롤포워드합니다.

```
python3 ./KclMigrationTool.py --region region --mode rollforward [--
application_name applicationName] [--
coordinator_state_table_name coordinatorStateTableName]
```

파라미터

`--region`

*##*을 사용자의 AWS 리전으로 바꿉니다.

`--application_name`

조정자 상태 테이블에 기본 이름을 사용하는 경우 이 파라미터가 필요합니다. 조정자 상태 테이블에 사용자 지정 이름을 지정한 경우 이 파라미터를 생략할 수 있습니다. *applicationName*을 실제 KCL 애플리케이션 이름으로 바꿉니다. 이 도구는 사용자 지정 이름이 제공되지 않은 경우 이 이름을 사용하여 기본 테이블 이름을 파생합니다.

`--coordinator_state_table_name`

이 파라미터는 KCL 구성에서 조정자 상태 테이블에 사용자 지정 이름을 설정한 경우에 필요합니다. 기본 테이블 이름을 사용하는 경우 이 파라미터를 생략할 수 있습니다. *coordinatorStateTableName*을 조정자 상태 테이블에 지정한 사용자 지정 테이블 이름으로 바꿉니다.

롤포워드 모드로 마이그레이션 도구를 실행한 후 KCL은 KCL 3.x에 필요한 다음과 같은 DynamoDB 리소스를 생성합니다.

- 리스 테이블의 글로벌 보조 인덱스
- 워커 지표 테이블

2단계: KCL 3.x로 코드 배포

롤포워드를 위해 KCL 마이그레이션 도구를 실행한 후 KCL 3.x로 워커에 코드를 배포합니다. 마이그레이션을 완료하려면 8단계: 마이그레이션 완료를 참조하세요.

연습: DynamoDB Streams Kinesis 어댑터

이번 섹션에서는 Amazon Kinesis Client Library와 Amazon DynamoDB Streams Kinesis 어댑터를 사용하는 Java 애플리케이션에 대해 살펴보겠습니다. 이 애플리케이션은 한 테이블의 쓰기 작업이 두 번째 테이블에도 적용되면서 두 테이블의 내용이 동기화를 유지하는 데이터 복제의 예로 설명됩니다. 소스 코드는 전체 프로그램: DynamoDB Streams Kinesis 어댑터 섹션을 참조하세요.

이 프로그램에서는 다음 작업을 수행합니다.

1. `KCL-Demo-src`와 `KCL-Demo-dst`라는 이름의 DynamoDB 테이블 2개를 생성합니다. 두 테이블 모두 스트림이 활성화되어 있습니다.
2. 항목을 추가, 업데이트 및 삭제하여 원본 테이블을 업데이트합니다. 이렇게 하면 데이터가 테이블의 스트림으로 기록됩니다.
3. 스트림에서 레코드를 읽고 DynamoDB 요청으로 재작성한 다음 대상 테이블에 요청을 적용합니다.
4. 원본 테이블과 대상 테이블을 스캔하여 내용이 동일한지 확인합니다.
5. 두 테이블을 삭제합니다.

이러한 단계는 다음 섹션에서 설명하며, 전체 애플리케이션은 연습 끝에 나와 있습니다.

주제

- 1단계: DynamoDB 테이블 생성
- 2단계: 소스 테이블의 업데이트 활동 생성
- 3단계: 스트림 처리
- 4단계: 양 테이블에 동일한 콘텐츠가 있는지 확인
- 5단계: 정리

- 전체 프로그램: DynamoDB Streams Kinesis 어댑터

1단계: DynamoDB 테이블 생성

첫 번째 단계에서 두 개의 DynamoDB 테이블(소스 테이블과 대상 테이블)을 생성합니다. 원본 테이블 스트림의 StreamViewType은 NEW_IMAGE입니다. 이 말은 원본 테이블 항목이 변경될 때마다 항목의 "사후" 이미지가 스트림에 기록된다는 것을 의미합니다. 이러한 방식으로 스트림이 테이블의 모든 쓰기 작업을 추적합니다.

다음은 두 테이블 생성에 사용된 코드를 보여주는 예제입니다.

```
java.util.List<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Id").withAttributeType("N"));

java.util.List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
keySchema.add(new
 KeySchemaElement().withAttributeName("Id").withKeyType(KeyType.HASH)); // Partition

  // key

ProvisionedThroughput provisionedThroughput = new
 ProvisionedThroughput().withReadCapacityUnits(2L)
    .withWriteCapacityUnits(2L);

StreamSpecification streamSpecification = new StreamSpecification();
streamSpecification.setStreamEnabled(true);
streamSpecification.setStreamViewType(StreamViewType.NEW_IMAGE);
CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName)
    .withAttributeDefinitions(attributeDefinitions).withKeySchema(keySchema)

 .withProvisionedThroughput(provisionedThroughput).withStreamSpecification(streamSpecification)
```

2단계: 소스 테이블의 업데이트 활동 생성

다음 단계는 원본 테이블의 쓰기 작업입니다. 이 작업을 하면 원본 테이블의 스트림 역시 거의 실시간으로 업데이트됩니다.

이 애플리케이션은 데이터 기록을 위해 PutItem, UpdateItem 및 DeleteItem API 작업을 호출하는 메서드를 사용하여 헬퍼 클래스를 정의합니다. 다음은 이러한 메서드의 사용 방법을 나타낸 코드 예제입니다.

```
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "101", "test1");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "101", "test2");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "101");
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "102", "demo3");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "102", "demo4");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "102");
```

3단계: 스트림 처리

이제 프로그램이 스트림을 처리합니다. DynamoDB Streams Kinesis 어댑터가 KCL과 DynamoDB Streams 엔드포인트 사이에서 투명 계층의 역할을 하기 때문에 코드에서 하위 수준 DynamoDB Streams를 호출할 필요 없이 KCL을 최대한 이용할 수 있습니다. 프로그램이 실행하는 작업은 다음과 같습니다.

- KCL 인터페이스 정의를 준수하는 메서드인 initialize, processRecords 및 shutdown을 사용하여 레코드 프로세서 클래스인 StreamsRecordProcessor를 정의합니다. processRecords 메서드에는 원본 테이블의 스트림에서 데이터를 읽어 대상 테이블에 기록하는 데 필요한 로직이 저장됩니다.
- 레코드 프로세서 클래스의 클래스 팩토리(StreamsRecordProcessorFactory)를 정의합니다. Java 프로그램이 KCL을 사용하려면 이 팩토리가 필요합니다.
- 새로운 KCL Worker를 인스턴스화하여 클래스 팩토리와 연동시킵니다.
- 레코드 처리가 완료되면 Worker를 종료합니다.

KCL 인터페이스 정의에 대한 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 Kinesis Client Library를 사용하여 소비자 개발을 참조하세요.

다음은 StreamsRecordProcessor의 메인 루프를 나타낸 코드 예제입니다. case 문은 스트림 레코드에 표시되는 OperationType에 따라 어떤 작업을 실행할지 결정합니다.

```
for (Record record : records) {
    String data = new String(record.getData().array(), Charset.forName("UTF-8"));
    System.out.println(data);
```

```
    if (record instanceof RecordAdapter) {
                software.amazon.dynamodb.model.Record streamRecord = ((RecordAdapter)
 record)
                    .getInternalObject();

                switch (streamRecord.getEventName()) {
                    case "INSERT":
                    case "MODIFY":
                        StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getNewImage());
                        break;
                    case "REMOVE":
                        StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getKeys().get("Id").getN());
                }
    }
    checkpointCounter += 1;
    if (checkpointCounter % 10 == 0) {
        try {
            checkpointer.checkpoint();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4단계: 양 테이블에 동일한 콘텐츠가 있는지 확인

이 시점에서는 원본 테이블과 대상 테이블의 내용이 동기화 상태를 유지합니다. 애플리케이션이 두 테이블에 대해 Scan 요청을 하여 내용이 실제로 동일한지 확인합니다.

DemoHelper 클래스에는 하위 수준 Scan API를 호출하는 ScanTable 메서드가 포함되어 있습니다. 다음 예제는 이 작업을 수행하는 방법을 보여줍니다.

```
if (StreamsAdapterDemoHelper.scanTable(dynamoDBClient, srcTable).getItems()
    .equals(StreamsAdapterDemoHelper.scanTable(dynamoDBClient, destTable).getItems()))
 {
    System.out.println("Scan result is equal.");
}
else {
    System.out.println("Tables are different!");
```

```
}
```

5단계: 정리

데모가 완료되면 애플리케이션이 원본 테이블과 대상 테이블을 삭제합니다. 다음 코드 예제를 참조하세요. 하지만 테이블이 삭제된 후에도 스트림은 최대 24시간까지 사용 가능하며, 이 시간이 지나면 자동 삭제됩니다.

```
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
```

전체 프로그램: DynamoDB Streams Kinesis 어댑터

다음은 연습: DynamoDB Streams Kinesis 어댑터에서 설명한 작업을 실행하는 전체 Java 프로그램입니다. 프로그램을 실행하면 다음과 비슷한 출력 화면이 보여야 합니다.

```
Creating table KCL-Demo-src
Creating table KCL-Demo-dest
Table is active.
Creating worker for stream: arn:aws:dynamodb:us-west-2:111122223333:table/KCL-Demo-src/
stream/2015-05-19T22:48:56.601
Starting worker...
Scan result is equal.
Done.
```

> **Important**
>
> 이 프로그램을 실행하려면 클라이언트 애플리케이션이 정책을 사용하여 DynamoDB 및 Amazon CloudWatch에 액세스할 수 있어야 합니다. 자세한 내용은 DynamoDB에 대한 자격 증명 기반 정책 섹션을 참조하세요.

소스 코드는 4개의 `.java` 파일로 구성됩니다.

- `StreamsAdapterDemo.java`
- `StreamsRecordProcessor.java`
- `StreamsRecordProcessorFactory.java`
- `StreamsAdapterDemoHelper.java`

StreamsAdapterDemo.java

```
package com.amazonaws.codesamples;

import
 com.amazonaws.services.dynamodbv2.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import com.amazonaws.services.dynamodbv2.streamsadapter.StreamsSchedulerFactory;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.polling.DynamoDBStreamsPollingConfig;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.InitialPositionInStream;
import software.amazon.kinesis.common.InitialPositionInStreamExtended;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;
import software.amazon.kinesis.processor.StreamTracker;
import software.amazon.kinesis.retrieval.RetrievalConfig;

public class StreamsAdapterDemo {

    private static DynamoDbAsyncClient dynamoDbAsyncClient;
    private static CloudWatchAsyncClient cloudWatchAsyncClient;
    private static AmazonDynamoDBStreamsAdapterClient
 amazonDynamoDbStreamsAdapterClient;

    private static String tablePrefix = "KCL-Demo";
    private static String streamArn;

    private static Region region = Region.US_EAST_1;
    private static AwsCredentialsProvider credentialsProvider =
 DefaultCredentialsProvider.create();

    public static void main( String[] args ) throws Exception {
        System.out.println("Starting demo...");
        dynamoDbAsyncClient = DynamoDbAsyncClient.builder()
                .credentialsProvider(credentialsProvider)
                .region(region)
                .build();
```

```
        cloudWatchAsyncClient = CloudWatchAsyncClient.builder()
                .credentialsProvider(credentialsProvider)
                .region(region)
                .build();
        amazonDynamoDbStreamsAdapterClient = new
AmazonDynamoDBStreamsAdapterClient(credentialsProvider, region);

        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";

        setUpTables();

        StreamTracker streamTracker =
StreamsSchedulerFactory.createSingleStreamTracker(streamArn,

InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON));

        ShardRecordProcessorFactory shardRecordProcessorFactory =
                new StreamsAdapterDemoProcessorFactory(dynamoDbAsyncClient, destTable);

        ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                "streams-adapter-demo",
                amazonDynamoDbStreamsAdapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                "streams-demo-worker",
                shardRecordProcessorFactory
        );

        DynamoDBStreamsPollingConfig pollingConfig = new
DynamoDBStreamsPollingConfig(amazonDynamoDbStreamsAdapterClient);
        RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
        retrievalConfig.retrievalSpecificConfig(pollingConfig);

        System.out.println("Creating scheduler for stream " + streamArn);
        Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
```

```
                amazonDynamoDbStreamsAdapterClient
        );

        System.out.println("Starting scheduler...");
        Thread t = new Thread(scheduler);
        t.start();

        Thread.sleep(250000);

        System.out.println("Stopping scheduler...");
        scheduler.shutdown();
        t.join();

        if (StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient, srcTable).items()
                .equals(StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient,
destTable).items())) {
            System.out.println("Scan result is equal.");
        } else {
            System.out.println("Tables are different!");
        }

        System.out.println("Done.");
        cleanupAndExit(0);
    }

    private static void setUpTables() {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        streamArn = StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient,
srcTable);
        StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient, destTable);

        awaitTableCreation(srcTable);

        performOps(srcTable);
    }

    private static void awaitTableCreation(String tableName) {
        Integer retries = 0;
        Boolean created = false;
        while (!created && retries < 100) {
            DescribeTableResponse result =
StreamsAdapterDemoHelper.describeTable(dynamoDbAsyncClient, tableName);
            created = result.table().tableStatusAsString().equals("ACTIVE");
```

```
            if (created) {
                System.out.println("Table is active.");
                return;
            } else {
                retries++;
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    // do nothing
                }
            }
        }
        System.out.println("Timeout after table creation. Exiting...");
        cleanupAndExit(1);
    }

    private static void performOps(String tableName) {
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "101",
 "test1");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "101",
 "test2");
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "101");
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "102",
 "demo3");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "102",
 "demo4");
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "102");
    }

    private static void cleanupAndExit(Integer returnValue) {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";

 dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(srcTable).build());

 dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(destTable).build());
        System.exit(returnValue);
    }
}
```

### StreamsRecordProcessor.java

```
package com.amazonaws.codesamples;
```

```
import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;

import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {

    private Integer checkpointCounter;

    private final DynamoDbAsyncClient dynamoDbAsyncClient;
    private final String tableName;

    public StreamsRecordProcessor(DynamoDbAsyncClient dynamoDbAsyncClient, String
 tableName) {
        this.dynamoDbAsyncClient = dynamoDbAsyncClient;
        this.tableName = tableName;
    }

    @Override
    public void initialize(InitializationInput initializationInput) {
        this.checkpointCounter = 0;
    }

    @Override
    public void processRecords(DynamoDBStreamsProcessRecordsInput
 dynamoDBStreamsProcessRecordsInput) {
        for (DynamoDBStreamsClientRecord record:
 dynamoDBStreamsProcessRecordsInput.records()) {
            String data = new String(record.data().array(), StandardCharsets.UTF_8);
            System.out.println(data);
```

```
            Record streamRecord = record.getRecord();

            switch (streamRecord.eventName()) {
                case INSERT:
                case MODIFY:
                    StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().newImage());
                case REMOVE:
                    StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().keys().get("Id").n());
            }
            checkpointCounter += 1;
            if (checkpointCounter % 10 == 0) {
                try {
                    dynamoDBStreamsProcessRecordsInput.checkpointer().checkpoint();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
        System.out.println("Lease Lost");
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
    }
```

```
}
```

StreamsRecordProcessorFactory.java

```
package com.amazonaws.codesamples;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class StreamsAdapterDemoProcessorFactory implements ShardRecordProcessorFactory
 {
    private final String tableName;
    private final DynamoDbAsyncClient dynamoDbAsyncClient;

    public StreamsAdapterDemoProcessorFactory(DynamoDbAsyncClient asyncClient, String
 tableName) {
        this.tableName = tableName;
        this.dynamoDbAsyncClient = asyncClient;
    }

    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new StreamsRecordProcessor(dynamoDbAsyncClient, tableName);
    }
}
```

StreamsAdapterDemoHelper.java

```
package com.amazonaws.codesamples;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
```

```
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.ResourceInUseException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.StreamViewType;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class StreamsAdapterDemoHelper {

    /**
     * @return StreamArn
     */
    public static String createTable(DynamoDbAsyncClient client, String tableName) {
        List<AttributeDefinition> attributeDefinitions = new ArrayList<>();
        attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName("Id")
                .attributeType("N")
                .build());

        List<KeySchemaElement> keySchema = new ArrayList<>();
        keySchema.add(KeySchemaElement.builder()
                .attributeName("Id")
                .keyType(KeyType.HASH) // Partition key
                .build());

        StreamSpecification streamSpecification = StreamSpecification.builder()
                .streamEnabled(true)
                .streamViewType(StreamViewType.NEW_IMAGE)
                .build();

        CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(attributeDefinitions)
                .keySchema(keySchema)
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .streamSpecification(streamSpecification)
                .build();
```

```
        try {
            System.out.println("Creating table " + tableName);
            CreateTableResponse result = client.createTable(createTableRequest).join();
            return result.tableDescription().latestStreamArn();
        } catch (Exception e) {
            if (e.getCause() instanceof ResourceInUseException) {
                System.out.println("Table already exists.");
                return describeTable(client, tableName).table().latestStreamArn();
            }
            throw e;
        }
    }

    public static DescribeTableResponse describeTable(DynamoDbAsyncClient client,
String tableName) {
        return client.describeTable(DescribeTableRequest.builder()
                        .tableName(tableName)
                        .build())
                .join();
    }

    public static ScanResponse scanTable(DynamoDbAsyncClient dynamoDbClient, String
tableName) {
        return dynamoDbClient.scan(ScanRequest.builder()
                        .tableName(tableName)
                        .build())
                .join();
    }

    public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
String id, String val) {
        Map<String, AttributeValue> item = new HashMap<>();
        item.put("Id", AttributeValue.builder().n(id).build());
        item.put("attribute-1", AttributeValue.builder().s(val).build());

        putItem(dynamoDbClient, tableName, item);
    }

    public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
                               Map<String, AttributeValue> items) {
        PutItemRequest putItemRequest = PutItemRequest.builder()
                .tableName(tableName)
                .item(items)
                .build();
```

```
        dynamoDbClient.putItem(putItemRequest).join();
    }

    public static void updateItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
 String id, String val) {
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("Id", AttributeValue.builder().n(id).build());

        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#attr2", "attribute-2");

        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":val", AttributeValue.builder().s(val).build());

        UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #attr2 = :val")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

        dynamoDbClient.updateItem(updateItemRequest).join();
    }

    public static void deleteItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
 String id) {
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("Id", AttributeValue.builder().n(id).build());

        DeleteItemRequest deleteItemRequest = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .build();
        dynamoDbClient.deleteItem(deleteItemRequest).join();
    }
}
```

# DynamoDB Streams 하위 수준 API: Java 예시

> **Note**
>
> 이 페이지의 코드는 완전한 코드가 아니며, Amazon DynamoDB Streams 사용 시나리오 중 일부만 처리합니다. DynamoDB에서 스트림 레코드를 사용하기 위해 권장되는 방법은 DynamoDB Streams Kinesis 어댑터를 사용하여 스트림 레코드 처리 섹션에 기술된 것처럼 Amazon Kinesis 어댑터를 통해 Kinesis Client Library(KCL)를 사용하는 것입니다.

이 섹션에는 실행 중인 DynamoDB Streams를 보여 주는 Java 프로그램이 나와 있습니다. 이 프로그램에서는 다음 작업을 수행합니다.

1. 스트림을 활성화하여 DynamoDB 테이블을 생성합니다.
2. 이 테이블의 스트림 설정을 설명합니다.
3. 테이블의 데이터를 수정합니다.
4. 스트림의 샤드를 설명합니다.
5. 샤드의 스트림 레코드를 읽습니다.
6. 하위 샤드를 가져오고 레코드를 계속 읽습니다.
7. 그리고 정리합니다.

프로그램을 실행하면 다음과 비슷한 출력 화면이 나타납니다.

```
Testing Streams Demo
Creating an Amazon DynamoDB table TestTableForStreams with a simple primary key: Id
Waiting for TestTableForStreams to be created...
Current stream ARN for TestTableForStreams: arn:aws:dynamodb:us-
east-2:123456789012:table/TestTableForStreams/stream/2018-03-20T16:49:55.208
Stream enabled: true
Update view type: NEW_AND_OLD_IMAGES

Performing write activities on TestTableForStreams
Processing item 1 of 100
Processing item 2 of 100
Processing item 3 of 100
...
Processing item 100 of 100
```

```
Shard: {ShardId: shardId-1234567890-...,SequenceNumberRange: {StartingSequenceNumber:
 100002572486797508907,},}
    Shard iterator: EjYFEkX2a26eVTWe...
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=200000158404754583390 9, SizeBytes=22,
 StreamViewType=NEW_AND_OLD_IMAGES)
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=210000360486976789270 1, SizeBytes=55,
 StreamViewType=NEW_AND_OLD_IMAGES)
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, SequenceNumber=220000109977111289843 4, SizeBytes=36,
 StreamViewType=NEW_AND_OLD_IMAGES)
...
Deleting the table...
Table StreamsDemoTable deleted.
Demo complete
```

## Example 예시

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsRequest;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsResponse;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorRequest;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.Shard;
import software.amazon.awssdk.services.dynamodb.model.ShardFilter;
import software.amazon.awssdk.services.dynamodb.model.ShardFilterType;
import software.amazon.awssdk.services.dynamodb.model.ShardIteratorType;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.streams.DynamoDbStreamsClient;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

public class StreamsLowLevelDemo {


    public static void main(String[] args) {
        final String usage = "Testing Streams Demo";
        try {
            System.out.println(usage);

            String tableName = "StreamsDemoTable";
            String key = "Id";
            System.out.println("Creating an Amazon DynamoDB table " + tableName + "
 with a simple primary key: " + key);
            Region region = Region.US_WEST_2;
            DynamoDbClient ddb = DynamoDbClient.builder()
                    .region(region)
                    .build();

            DynamoDbStreamsClient ddbStreams = DynamoDbStreamsClient.builder()
                    .region(region)
                    .build();
            DescribeTableRequest describeTableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();
            TableDescription tableDescription = null;
```

```
            try{
                tableDescription = ddb.describeTable(describeTableRequest).table();
            }catch (Exception e){
                System.out.println("Table " + tableName + " does not exist.");
                tableDescription = createTable(ddb, tableName, key);
            }

            // Print the stream settings for the table
            String streamArn = tableDescription.latestStreamArn();

            StreamSpecification streamSpec = tableDescription.streamSpecification();
            System.out.println("Current stream ARN for " + tableDescription.tableName()
+ ": " +
                   streamArn);
            System.out.println("Stream enabled: " + streamSpec.streamEnabled());
            System.out.println("Update view type: " + streamSpec.streamViewType());
            System.out.println();
            // Generate write activity in the table
            System.out.println("Performing write activities on " + tableName);
            int maxItemCount = 100;
            for (Integer i = 1; i <= maxItemCount; i++) {
                System.out.println("Processing item " + i + " of " + maxItemCount);
                // Write a new item
                putItemInTable(key, i, tableName, ddb);
                // Update the item
                updateItemInTable(key, i, tableName, ddb);
                // Delete the item
                deleteDynamoDBItem(key, i, tableName, ddb);
            }

            // Process Stream
            processStream(streamArn, maxItemCount, ddb, ddbStreams, tableName);

            // Delete the table
            System.out.println("Deleting the table...");
            DeleteTableRequest deleteTableRequest = DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build();
            ddb.deleteTable(deleteTableRequest);
            System.out.println("Table " + tableName + " deleted.");
            System.out.println("Demo complete");
            ddb.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
```

```
        }
    }

    private static void processStream(String streamArn, int maxItemCount,
DynamoDbClient ddb, DynamoDbStreamsClient ddbStreams, String tableName) {
        // Get all the shard IDs from the stream. Note that DescribeStream returns
        // the shard IDs one page at a time.
        String lastEvaluatedShardId = null;
        do {
            DescribeStreamRequest describeStreamRequest =
DescribeStreamRequest.builder()
                    .streamArn(streamArn)
                    .exclusiveStartShardId(lastEvaluatedShardId).build();
            DescribeStreamResponse describeStreamResponse =
ddbStreams.describeStream(describeStreamRequest);

            List<Shard> shards = describeStreamResponse.streamDescription().shards();

            // Process each shard on this page

            fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams, shards);

            // If LastEvaluatedShardId is set, then there is
            // at least one more page of shard IDs to retrieve
            lastEvaluatedShardId =
describeStreamResponse.streamDescription().lastEvaluatedShardId();

        } while (lastEvaluatedShardId != null);

    }

    private static void fetchShardsAndReadRecords(String streamArn, int maxItemCount,
DynamoDbStreamsClient ddbStreams, List<Shard> shards) {
        for (Shard shard : shards) {
            String shardId = shard.shardId();
            System.out.println("Shard: " + shard);

            // Get an iterator for the current shard
            GetShardIteratorRequest shardIteratorRequest =
GetShardIteratorRequest.builder()
                    .streamArn(streamArn).shardId(shardId)
                    .shardIteratorType(ShardIteratorType.TRIM_HORIZON).build();
```

```
            GetShardIteratorResponse getShardIteratorResult =
ddbStreams.getShardIterator(shardIteratorRequest);

            String currentShardIter = getShardIteratorResult.shardIterator();

            // Shard iterator is not null until the Shard is sealed (marked as
READ_ONLY).
            // To prevent running the loop until the Shard is sealed, we process only
the
            // items that were written into DynamoDB and then exit.
            int processedRecordCount = 0;
            while (currentShardIter != null && processedRecordCount < maxItemCount) {
                // Use the shard iterator to read the stream records
                GetRecordsRequest getRecordsRequest = GetRecordsRequest.builder()
                        .shardIterator(currentShardIter).build();
                GetRecordsResponse getRecordsResult =
ddbStreams.getRecords(getRecordsRequest);
                List<Record> records = getRecordsResult.records();
                for (Record record : records) {
                    System.out.println("        " + record.dynamodb());
                }
                processedRecordCount += records.size();
                currentShardIter = getRecordsResult.nextShardIterator();
            }
            if (currentShardIter == null){
                System.out.println("Shard has been fully processed. Shard iterator is
null.");
                System.out.println("Fetch the child shard to continue processing
instead of bulk fetching all shards");
                DescribeStreamRequest describeStreamRequestForChildShards =
DescribeStreamRequest.builder()
                        .streamArn(streamArn)
                        .shardFilter(ShardFilter.builder()
                                .type(ShardFilterType.CHILD_SHARDS)
                                .shardId(shardId).build())
                        .build();
                DescribeStreamResponse describeStreamResponseChildShards =
ddbStreams.describeStream(describeStreamRequestForChildShards);
                fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams,
describeStreamResponseChildShards.streamDescription().shards());
            }
        }
    }
```

```
   private static void putItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {
       Map<String, AttributeValue> item = new HashMap<>();
       item.put(key, AttributeValue.builder()
               .s(i.toString())
               .build());
       item.put("Message", AttributeValue.builder()
               .s("New Item!")
               .build());
       PutItemRequest request = PutItemRequest.builder()
               .tableName(tableName)
               .item(item)
               .build();
       ddb.putItem(request);
   }

   private static void updateItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {

       HashMap<String, AttributeValue> itemKey = new HashMap<>();
       itemKey.put(key, AttributeValue.builder()
               .s(i.toString())
               .build());


       HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
       updatedValues.put("Message", AttributeValueUpdate.builder()
               .value(AttributeValue.builder().s("This is an updated item").build())
               .action(AttributeAction.PUT)
               .build());

       UpdateItemRequest request = UpdateItemRequest.builder()
               .tableName(tableName)
               .key(itemKey)
               .attributeUpdates(updatedValues)
               .build();
       ddb.updateItem(request);
   }

   public static void deleteDynamoDBItem(String key, Integer i, String tableName,
DynamoDbClient ddb) {
       HashMap<String, AttributeValue> keyToGet = new HashMap<>();
       keyToGet.put(key, AttributeValue.builder()
               .s(i.toString())
```

```
                .build());

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();
        ddb.deleteItem(deleteReq);
    }

    public static TableDescription createTable(DynamoDbClient ddb, String tableName,
String key) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        StreamSpecification streamSpecification = StreamSpecification.builder()
                .streamEnabled(true)
                .streamViewType("NEW_AND_OLD_IMAGES")
                .build();
        CreateTableRequest request = CreateTableRequest.builder()
                .attributeDefinitions(AttributeDefinition.builder()
                        .attributeName(key)
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(KeySchemaElement.builder()
                        .attributeName(key)
                        .keyType(KeyType.HASH)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
                .tableName(tableName)
                .streamSpecification(streamSpecification)
                .build();

        TableDescription newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();

            System.out.println("Waiting for " + tableName + " to be created...");

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
```

```
            newTable = response.tableDescription();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return null;
    }



}
```

## DynamoDB Streams 및 AWS Lambda 트리거

Amazon DynamoDB는 DynamoDB Streams의 이벤트에 자동으로 응답하는 코드 조각인 트리거를 만들 수 있도록 AWS Lambda와 통합되어 있습니다. 트리거를 사용하면 DynamoDB 테이블의 데이터 수정에 응답하는 애플리케이션을 빌드할 수 있습니다.

주제

- 자습서 #1: 필터를 사용하여 AWS CLI로 AWS Lambda 및 Amazon DynamoDB에서 모든 이벤트 처리
- 자습서 #2: 필터를 사용하여 DynamoDB 및 Lambda에서 일부 이벤트 처리
- DynamoDB Streams를 Lambda와 함께 사용하는 모범 사례

테이블에서 DynamoDB Streams를 활성화할 경우 스트림 Amazon 리소스 이름(ARN)을 사용자가 작성하는 AWS Lambda 함수에 연결할 수 있습니다. 그러면 해당 DynamoDB 테이블에 대한 모든 변형 작업을 스트림의 항목으로 캡처할 수 있습니다. 예를 들어, 테이블의 항목이 수정될 때 새 레코드가 해당 테이블의 스트림에 즉시 나타나도록 트리거를 설정할 수 있습니다.

> **Note**
>
> 하나의 DynamoDB 스트림에 3개 이상의 Lambda 함수를 구독하는 경우 읽기 제한이 발생할 수 있습니다.

AWS Lambda 서비스는 초당 4번 새 레코드에 대한 스트림을 폴링합니다. 새 스트림 레코드를 사용할 수 있게 되면 Lambda 함수가 동기식으로 호출됩니다. 동일한 DynamoDB 스트림에 최대 2개의

Lambda 함수를 구독할 수 있습니다. 동일한 DynamoDB 스트림에 3개 이상의 Lambda 함수를 구독하는 경우 읽기 제한이 발생할 수 있습니다.

Lambda 함수는 알림을 보내거나 워크플로를 시작하거나 사용자가 지정하는 기타 여러 작업을 수행할 수 있습니다. 각 스트림 레코드를 Amazon S3 File Gateway(Amazon S3)와 같은 영구 스토리지에 간단하게 복사하는 Lambda 함수를 작성하여 테이블의 쓰기 작업에 대한 영구 감사 추적을 만들 수 있습니다. `GameScores` 테이블에 쓰는 모바일 게임 앱이 있다고 가정해 보겠습니다. `GameScores` 테이블의 `TopScore` 속성이 업데이트될 때마다 해당하는 스트림 레코드가 테이블 스트림에 기록됩니다. 그런 다음 이 이벤트는 소셜 미디어 네트워크에 축하 메시지를 게시하는 Lambda 함수를 트리거합니다. 이 함수는 `GameScores`에 업데이트되지 않거나 `TopScore` 속성을 수정하지 않는 모든 스트림 레코드를 무시하도록 작성할 수도 있습니다.

함수가 오류를 반환하면 Lambda는 성공적으로 처리되거나 데이터가 만료될 때까지 배치(batch)를 재시도합니다. 또한 더 작은 배치로 재시도하고, 재시도 횟수를 제한하고, 너무 오래된 레코드를 폐기하고, 기타 옵션을 사용하도록 Lambda를 구성할 수 있습니다.

성능 모범 사례에 따라 Lambda 함수는 수명이 짧아야 합니다. 불필요한 처리 지연을 방지하기 위해 복잡한 로직도 실행하지 않아야 합니다. 특히 고속 스트림의 경우 장기 실행 중인 동기식 Lambda보다 비동기식 사후 처리 단계 함수 워크플로를 트리거하는 것이 좋습니다.

여러 AWS 계정에서 동일한 Lambda 트리거를 사용할 수 없습니다. DynamoDB 테이블과 Lambda 함수는 모두 동일한 AWS 계정에 속해야 합니다.

AWS Lambda에 대한 자세한 내용은 AWS Lambda개발자 가이드를 참조하세요.

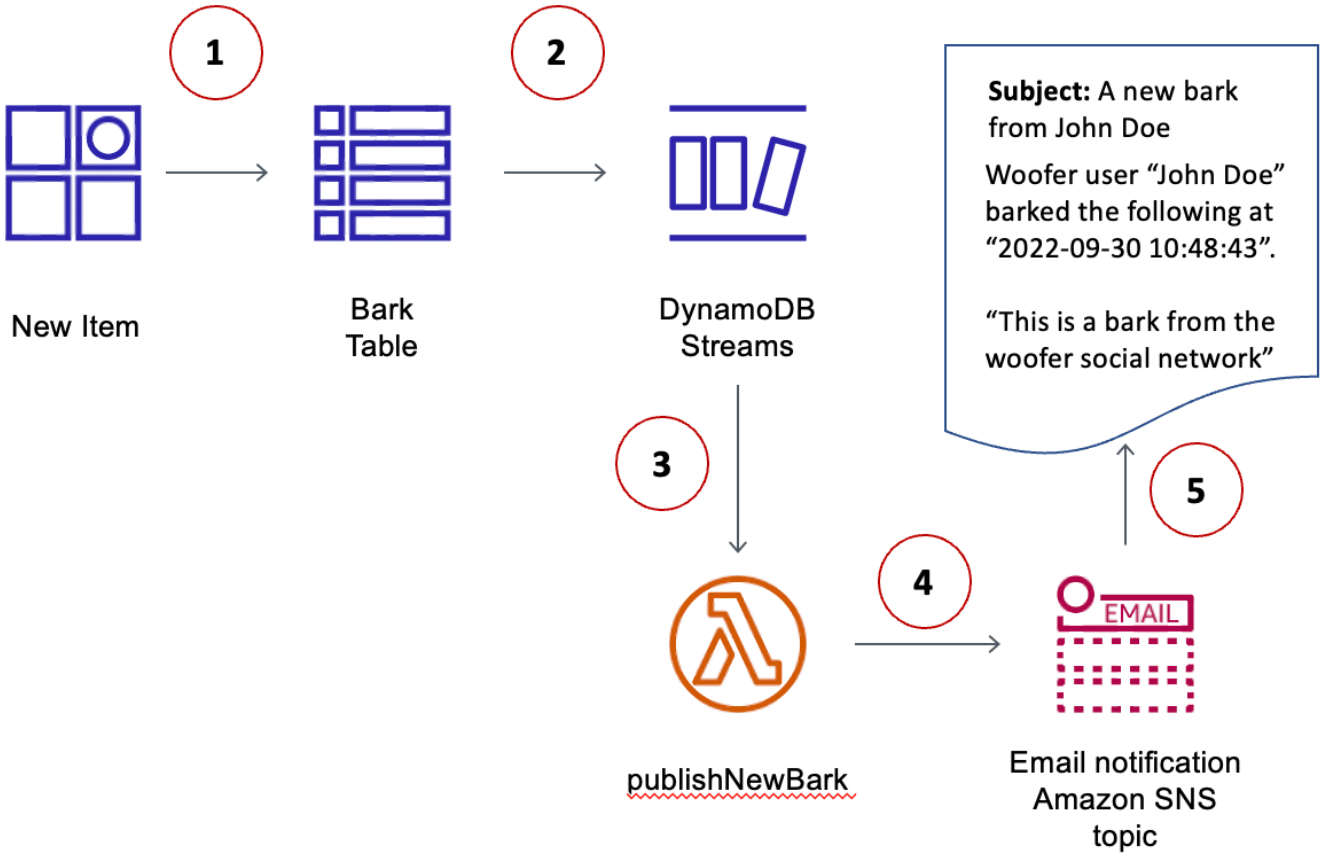
자습서 #1: 필터를 사용하여 AWS CLI로 AWS Lambda 및 Amazon DynamoDB에서 모든 이벤트 처리

이 자습서에서는 DynamoDB 테이블의 스트림을 처리하기 위해 AWS Lambda 트리거를 생성합니다.

주제

- 1단계: 스트림을 활성화하여 DynamoDB 테이블 생성
- 2단계: Lambda 실행 역할 생성
- 3단계: Amazon SNS 주제 생성
- 4단계: Lambda 함수 생성 및 테스트
- 5단계: 트리거 생성 및 테스트

이 자습서의 시나리오는 간단한 소셜 네트워크인 Woofer입니다. Woofer 사용자는 다른 Woofer 사용자에게 전달되는 바크(간단한 문자 메시지)를 이용해 의사소통합니다. 다음 다이어그램은 이 애플리케이션에 대한 구성 요소 및 워크플로우를 보여줍니다.




1. 사용자는 DynamoDB 테이블(`BarkTable`)에 항목을 씁니다. 테이블에서 각 항목은 바크를 나타냅니다.
2. 작성된 새 스트림 레코드는 새 항목이 `BarkTable`에 추가되었다는 것을 반영합니다.
3. 새 스트림 레코드에서 AWS Lambda 함수(`publishNewBark`)를 트리거합니다.
4. 스트림 레코드가 새 항목이 `BarkTable`에 추가되었음을 나타내는 경우 Lambda 함수는 스트림 레코드에서 데이터를 읽고 Amazon Simple Notification Service(Amazon SNS)의 주제에 메시지를 게시합니다.
5. Amazon SNS 주제의 구독자는 메시지를 수신합니다. (본 자습서에서 이메일 주소는 구독자에 해당됩니다.)

## 시작하기 전

이 자습서에서는 AWS Command Line Interface AWS CLI을 사용합니다. 아직 완료하지 않았다면 AWS Command Line Interface 사용 설명서의 지침에 따라 AWS CLI를 설치 및 구성하세요.

1단계: 스트림을 활성화하여 DynamoDB 테이블 생성

이 단계에서는 Woofer 사용자의 모든 바크를 저장할 DynamoDB 테이블(BarkTable)을 생성합니다. 기본 키는 Username(파티션 키)와 Timestamp(정렬 키)로 구성됩니다. 이 두 가지 속성 모두 문자열 유형입니다.

BarkTable에서는 스트림이 활성화되어 있습니다. 이 자습서에서 나중에 AWS Lambda 함수를 스트림에 연결하여 트리거를 생성할 수 있습니다.

1. 테이블을 생성하려면 다음 명령을 입력합니다.

```
aws dynamodb create-table \
    --table-name BarkTable \
    --attribute-definitions AttributeName=Username,AttributeType=S
 AttributeName=Timestamp,AttributeType=S \
    --key-schema AttributeName=Username,KeyType=HASH
 AttributeName=Timestamp,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

2. 출력에서 LatestStreamArn를 찾습니다.

```
...
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

*region* 및 *accountID*를 기록해 둡니다. 이 두 항목은 자습서의 다른 단계에서 필요합니다.

2단계: Lambda 실행 역할 생성

이 단계에서는 AWS Identity and Access Management(IAM) 역할(WooferLambdaRole_을 만들어 권한을 할당합니다. 이 역할은 4단계: Lambda 함수 생성 및 테스트에서 만드는 Lambda 함수에서 사용합니다.

또한 해당 역할에 대한 정책도 생성합니다. 이 정책에는 Lambda 함수에서 런타임에 필요로 하는 모든 권한이 포함되어 있습니다.

1. 다음 콘텐츠를 가진 `trust-relationship.json`이라는 파일을 생성합니다:

   JSON

   ```
   {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "lambda.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
   ```

2. 다음 명령을 입력하여 `WooferLambdaRole`을 생성합니다.

   ```
   aws iam create-role --role-name WooferLambdaRole \
       --path "/service-role/" \
       --assume-role-policy-document file://trust-relationship.json
   ```

3. 다음 콘텐츠를 가진 `role-policy.json`이라는 파일을 생성합니다: (*region* 및 *accountID*를 AWS 리전 및 계정 ID로 바꿉니다.)

   JSON

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "logs:CreateLogGroup",
                   "logs:CreateLogStream",
                   "logs:PutLogEvents"
               ],
               "Resource": "arn:aws:logs:us-east-1:111122223333:*"
           },
           {
   ```

```
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/
BarkTable/stream/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

정책은 WooferLambdaRole이 다음을 수행할 수 있도록 해주는 네 가지 문을 가지고 있습니다.

- Lambda 함수 실행(publishNewBark). 이 자습서에서는 나중에 함수를 생성합니다.
- Amazon CloudWatch Logs 액세스. Lambda 함수는 런타임에 CloudWatch Logs에 진단을 씁니다.
- BarkTable에 대한 DynamoDB 스트림에서 데이터 읽기
- Amazon SNS에 메시지 게시

4. 다음 명령을 입력하여 정책을 WooferLambdaRole에 연결합니다.

```
aws iam put-role-policy --role-name WooferLambdaRole \
    --policy-name WooferLambdaRolePolicy \
    --policy-document file://role-policy.json
```

### 3단계: Amazon SNS 주제 생성

이 단계에서는 Amazon SNS 주제(wooferTopic)를 생성하고 해당 주제에 대한 이메일 주소를 구독합니다. Lambda 함수에서는 이 주제를 사용하여 Woofer 사용자의 새 바크를 게시합니다.

1. 다음 명령을 입력하여 새 Amazon SNS 주제를 생성합니다.

```
aws sns create-topic --name wooferTopic
```

2. 다음 명령을 입력해 wooferTopic에 대한 이메일 주소를 구독합니다. (*region* 및 *accountID*를 AWS 리전 및 계정 ID로 바꾸고, *example@example.com*을 유효한 이메일 주소로 바꿉니다.)

```
aws sns subscribe \
    --topic-arn arn:aws:sns:region:accountID:wooferTopic \
    --protocol email \
    --notification-endpoint example@example.com
```

3. Amazon SNS에서 사용자 이메일 주소로 확인 메시지를 보냅니다. 구독 프로세스를 완료하는 메시지와 연결된 구독 확인 링크를 클릭합니다.

### 4단계: Lambda 함수 생성 및 테스트

이 단계에서는 AWS Lambda 함수(publishNewBark)를 만들어 BarkTable에서 스트림 레코드를 처리합니다.

publishNewBark 함수는 BarkTable의 새 항목에 해당하는 스트림 이벤트만을 처리합니다. 함수는 해당 이벤트에서 데이터를 읽은 다음 Amazon SNS를 호출하여 게시합니다.

1. 다음 콘텐츠를 가진 publishNewBark.js이라는 파일을 생성합니다: *region* 및 *accountID*를 AWS 리전 및 계정 ID로 바꿉니다.

```
'use strict';
var AWS = require("aws-sdk");
var sns = new AWS.SNS();

exports.handler = (event, context, callback) => {

    event.Records.forEach((record) => {
        console.log('Stream record: ', JSON.stringify(record, null, 2));

        if (record.eventName == 'INSERT') {
            var who = JSON.stringify(record.dynamodb.NewImage.Username.S);
            var when = JSON.stringify(record.dynamodb.NewImage.Timestamp.S);
            var what = JSON.stringify(record.dynamodb.NewImage.Message.S);
            var params = {
                Subject: 'A new bark from ' + who,
```

```
                Message: 'Woofer user ' + who + ' barked the following at ' + when
 + ':\n\n ' + what,
                TopicArn: 'arn:aws:sns:region:accountID:wooferTopic'
            };
            sns.publish(params, function(err, data) {
                if (err) {
                    console.error("Unable to send message. Error JSON:",
 JSON.stringify(err, null, 2));
                } else {
                    console.log("Results from sending message: ",
 JSON.stringify(data, null, 2));
                }
            });
        }
    });
    callback(null, `Successfully processed ${event.Records.length} records.`);
};
```

2. publishNewBark.js가 포함된 zip 파일을 만듭니다. zip 명령줄 유틸리티를 사용하는 경우 다음과 같은 명령을 입력하여 이를 수행할 수 있습니다.

```
zip publishNewBark.zip publishNewBark.js
```

3. Lambda 함수를 만들 때 2단계: Lambda 실행 역할 생성에서 생성한 WooferLambdaRole의 Amazon 리소스 이름(ARN)을 지정합니다. 이 ARN을 가져오려면 다음 명령을 입력합니다.

```
aws iam get-role --role-name WooferLambdaRole
```

출력에서 WooferLambdaRole의 ARN을 찾습니다.

```
...
"Arn": "arn:aws:iam::region:role/service-role/WooferLambdaRole"
...
```

Lambda 함수를 생성하려면 다음 명령을 입력합니다. *roleARN*를 WooferLambdaRole의 ARN으로 바꿉니다.

```
aws lambda create-function \
    --region region \
    --function-name publishNewBark \
```

```
    --zip-file fileb://publishNewBark.zip \
    --role roleARN \
    --handler publishNewBark.handler \
    --timeout 5 \
    --runtime nodejs16.x
```

4. 이제, publishNewBark을 테스트하여 제대로 작동하는지 확인합니다. 이렇게 하려면 DynamoDB Streams의 실제 레코드와 유사한 내용을 입력합니다.

   다음 콘텐츠를 가진 payload.json이라는 파일을 생성합니다: *region* 및 *accountID*를 AWS 리전 및 계정 ID로 바꿉니다.

```
{
    "Records": [
        {
            "eventID": "7de3041dd709b024af6f29e4fa13d34c",
            "eventName": "INSERT",
            "eventVersion": "1.1",
            "eventSource": "aws:dynamodb",
            "awsRegion": "region",
            "dynamodb": {
                "ApproximateCreationDateTime": 1479499740,
                "Keys": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "NewImage": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Message": {
                        "S": "This is a bark from the Woofer social network"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "SequenceNumber": "13021600000000001596893679",
                "SizeBytes": 112,
```

```
                "StreamViewType": "NEW_IMAGE"
            },
            "eventSourceARN": "arn:aws:dynamodb:region:account ID:table/BarkTable/
stream/2016-11-16T20:42:48.104"
        }
    ]
}
```

publishNewBark 함수를 테스트하려면 다음 명령을 입력합니다.

```
aws lambda invoke --function-name publishNewBark --payload file://payload.json --
cli-binary-format raw-in-base64-out output.txt
```

테스트가 성공적이면 다음 출력이 생성됩니다.

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

또한 output.txt 파일에는 다음 텍스트가 포함됩니다.

```
"Successfully processed 1 records."
```

몇 분 안에 새 이메일 메시지가 전송됩니다.

> **Note**
>
> AWS Lambda는 진단 정보를 Amazon CloudWatch Logs에 씁니다. Lambda 함수에서 오류가 발생하면 문제 해결을 위해 이 진단 정보를 사용할 수 있습니다.
>
> 1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
> 2. 탐색 창에서 로그를 선택합니다.
> 3. 다음과 같은 로그 그룹을 선택합니다./aws/lambda/publishNewBark
> 4. 최신 로그 스트림을 선택하여 함수의 출력(오류 사항 포함)을 봅니다.

### 5단계: 트리거 생성 및 테스트

4단계: Lambda 함수 생성 및 테스트에서 Lambda 함수를 테스트하여 제대로 실행되는지 확인합니다. 이 단계에서는 Lambda 함수(publishNewBark)를 이벤트 소스(BarkTable 스트림)과 연결하여 트리거를 생성합니다.

1. 트리거를 생성할 때 BarkTable 스트림의 ARN을 지정해야 합니다. 이 ARN을 가져오려면 다음 명령을 입력합니다.

   ```
   aws dynamodb describe-table --table-name BarkTable
   ```

   출력에서 LatestStreamArn를 찾습니다.

   ```
   ...
    "LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
   stream/timestamp
   ...
   ```

2. 다음 명령을 입력하여 트리거를 생성합니다. *streamARN*를 실제 스트림 ARN으로 바꿉니다.

   ```
   aws lambda create-event-source-mapping \
       --region region \
       --function-name publishNewBark \
       --event-source streamARN  \
       --batch-size 1 \
       --starting-position TRIM_HORIZON
   ```

3. 트리거를 테스트합니다. 다음 명령을 입력하여 BarkTable에 항목을 추가합니다.

   ```
   aws dynamodb put-item \
       --table-name BarkTable \
       --item Username={S="Jane
    Doe"},Timestamp={S="2016-11-18:14:32:17"},Message={S="Testing...1...2...3"}
   ```

   몇 분 이내에 새 이메일 메시지를 수신합니다.

4. DynamoDB 콘솔을 열어서 BarkTable에 몇 개 항목을 추가합니다. Username 및 Timestamp에 대한 속성값을 지정해야 합니다. (필수 사항은 아니지만 Message에 대한 값을 지정할 수도 있습니다.) 추가한 BarkTable의 각 항목에 대해 새 이메일 메시지를 수신하게 됩니다.

Lambda 함수는 BarkTable에 추가된 새 항목만을 처리합니다. 테이블에 업데이트되거나 삭제한 항목에 대해 이 함수는 별도의 작업을 하지 않습니다.

> **Note**
>
> AWS Lambda는 진단 정보를 Amazon CloudWatch Logs에 씁니다. Lambda 함수에서 오류가 발생하면 문제 해결을 위해 이 진단 정보를 사용할 수 있습니다.
>
> 1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
> 2. 탐색 창에서 로그를 선택합니다.
> 3. 다음과 같은 로그 그룹을 선택합니다./aws/lambda/publishNewBark
> 4. 최신 로그 스트림을 선택하여 함수의 출력(오류 사항 포함)을 봅니다.

자습서 #2: 필터를 사용하여 DynamoDB 및 Lambda에서 일부 이벤트 처리

이 자습서에서는 DynamoDB 테이블의 스트림에서 일부 이벤트만 처리하기 위해 AWS Lambda 트리거를 생성합니다.

주제

- 종합 - AWS CloudFormation
- 종합 - CDK

Lambda 이벤트 필터링을 사용하면 필터 표현식을 사용하여 Lambda가 어떤 이벤트를 처리를 위해 함수로 보내는지를 제어할 수 있습니다. DynamoDB Streams당 최대 5개의 서로 다른 필터를 구성할 수 있습니다. 일괄 처리 기간을 사용하는 경우 Lambda는 각 새 이벤트에 필터 기준을 적용하여 현재 배치에 포함되어야 하는지 확인합니다.

필터는 FilterCriteria라는 구조를 통해 적용됩니다. FilterCriteria의 3가지 주요 속성은 metadata properties, data properties 및 filter patterns입니다.

다음은 DynamoDB Streams 이벤트의 예제 구조입니다.

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
```

```
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
      "quantity": { "N": "50" },
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "700000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```


`metadata properties`는 이벤트 객체의 필드입니다. DynamoDB Streams의 경우 `metadata properties`은 dynamodb 또는 `eventName`과 같은 필드입니다.

`data properties`은 이벤트 본문의 필드입니다. `data properties`을 필터링하려면 적절한 키 내의 `FilterCriteria`에 해당 속성을 포함해야 합니다. DynamoDB 이벤트 소스의 경우 데이터 키는 `NewImage` 또는 `OldImage`입니다.

마지막으로 필터 규칙은 특정 속성에 적용할 필터 표현식을 정의합니다. 여기 몇 가지 예가 있습니다:

| 비교 연산자 | 예시 | 규칙 구문(부분적) |
| --- | --- | --- |
| Null | 제품 유형이 null입니다. | `{ "product_type": { "S": null } }` |

| 비교 연산자 | 예시 | 규칙 구문(부분적) |
| --- | --- | --- |
| 비어 있음 | 제품 이름이 비어 있습니다. | `{ "product_name": { "S": [ ""] } }` |
| 같음 | 주가 플로리다와 같습니다. | `{ "state": { "S": ["FL"] } }` |
| 및 | 제품 주는 플로리다와 같고 제품 범주는 초콜릿과 같습니다. | `{ "state": { "S": ["FL"] } , "category": { "S": [ "CHOCOLATE"] } }` |
| Or | 제품 주는 플로리다 또는 캘리포니아입니다. | `{ "state": { "S": ["FL","CA"] } }` |
| 아님 | 제품 주는 플로리다가 아닙니다. | `{"state": {"S": [{"anything-but": ["FL"]}]}}` |
| 존재함 | 홈메이드 제품이 있습니다. | `{"homemade": {"S": [{"exists": true}]}}` |
| 존재하지 않음 | 홈메이드 제품이 존재하지 않습니다 | `{"homemade": {"S": [{"exists": false}]}}` |
| 다음으로 시작 | PK는 COMPANY로 시작합니다. | `{"PK": {"S": [{"prefix": "COMPANY"}]}}` |

Lambda 함수에 대해 최대 5개의 이벤트 필터링 패턴을 지정할 수 있습니다. 5개의 이벤트 각각은 논리적 OR로 평가됩니다. 따라서 `Filter_One` 및 `Filter_Two`라는 두 개의 필터를 구성하면 Lambda 함수가 `Filter_One` 또는 `Filter_Two`를 실행합니다.

> **Note**
>
> Lambda 이벤트 필터링 페이지에는 숫자 값을 필터링하고 비교하는 몇 가지 옵션이 있지만 DynamoDB 필터 이벤트의 경우 DynamoDB의 숫자가 문자열로 저장되기 때문에 적용되지 않

습니다. 예를 들어 `"quantity": { "N": "50" }`의 경우, `"N"` 속성 때문에 숫자임을 알 수 있습니다.

### 종합 - AWS CloudFormation

이벤트 필터링 기능을 실제로 보여주기 위해 CloudFormation 템플릿 샘플을 소개합니다. 이 템플릿은 Amazon DynamoDB Streams가 활성화된 파티션 키 PK 및 정렬 키 SK가 있는 간단한 DynamoDB 테이블을 생성합니다. 그러면 Amazon Cloudwatch에 로그를 쓰고 Amazon DynamoDB 스트림에서 이벤트를 읽을 수 있는 람다 함수와 간단한 Lambda 실행 역할이 생성됩니다. 또한 DynamoDB Streams와 Lambda 함수 사이에 이벤트 소스 매핑을 추가하므로 Amazon DynamoDB Streams에 이벤트가 있을 때마다 함수를 실행할 수 있습니다.

```
AWSTemplateFormatVersion: "2010-09-09"

Description: Sample application that presents AWS Lambda event source filtering
with Amazon DynamoDB Streams.

Resources:
  StreamsSampleDDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: "PK"
          AttributeType: "S"
        - AttributeName: "SK"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "PK"
          KeyType: "HASH"
        - AttributeName: "SK"
          KeyType: "RANGE"
      StreamSpecification:
        StreamViewType: "NEW_AND_OLD_IMAGES"
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5

  LambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
```

```
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: "/"
      Policies:
        - PolicyName: root
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action:
                  - logs:CreateLogGroup
                  - logs:CreateLogStream
                  - logs:PutLogEvents
                Resource: arn:aws:logs:*:*:*
              - Effect: Allow
                Action:
                  - dynamodb:DescribeStream
                  - dynamodb:GetRecords
                  - dynamodb:GetShardIterator
                  - dynamodb:ListStreams
                Resource: !GetAtt StreamsSampleDDBTable.StreamArn

  EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST

  ProcessEventLambda:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python3.7
      Timeout: 300
      Handler: index.handler
      Role: !GetAtt LambdaExecutionRole.Arn
```

```
      Code:
        ZipFile: |
          import logging

          LOGGER = logging.getLogger()
          LOGGER.setLevel(logging.INFO)

          def handler(event, context):
            LOGGER.info('Received Event: %s', event)
            for rec in event['Records']:
              LOGGER.info('Record: %s', rec)

Outputs:
  StreamsSampleDDBTable:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.Arn
  StreamARN:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.StreamArn
```

이 클라우드 구성 템플릿을 배포한 후 다음과 같은 Amazon DynamoDB 항목을 삽입할 수 있습니다.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

이 클라우드 형성 템플릿에 인라인으로 포함된 간단한 람다 함수 덕분에 다음과 같이 람다 함수에 대한 Amazon CloudWatch 로그 그룹의 이벤트를 볼 수 있습니다.

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
```

```
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
      "quantity": { "N": "50" },
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "700000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

필터링 예제

- 지정된 주와 일치하는 제품만

이 예제에서는 플로리다(약어 'FL')에서 생산되는 모든 제품을 일치시키는 필터를 포함하도록 CloudFormation 템플릿을 수정합니다.

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
```

```
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

스택을 재배포하고 나면 다음 DynamoDB 항목을 테이블에 추가할 수 있습니다. 이 예제의 제품은 캘리포니아에서 만들어진 제품이므로 Lambda 함수 로그에는 표시되지 않습니다.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK#1000",
 "company_id": "1000",
 "fabric": "Florida Chocolates",
 "price": 15,
 "product_id": "1000",
 "quantity": 50,
 "state": "CA",
 "stores": 5,
 "type": ""
}
```

- PK 및 SK의 일부 값으로 시작하는 항목만

이 예에서는 다음 조건을 포함하도록 CloudFormation 템플릿을 수정합니다.

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix":
 "COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

AND 조건을 사용하려면 패턴 내부에 조건이 있어야 하며 이 경우 PK 및 SK 키가 쉼표로 구분되어 동일한 표현식 안에 있습니다.

PK 및 SK의 일부 값으로 시작하거나 특정 상태에서 시작합니다.

이 예에서는 다음 조건을 포함하도록 CloudFormation 템플릿을 수정합니다.

```
  EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix":
"COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
          - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

필터 섹션에 새 패턴을 도입하여 OR 조건이 추가되었음을 알 수 있습니다.

종합 - CDK

다음 샘플 CDK 프로젝트 구성 템플릿은 이벤트 필터링 기능을 안내합니다. 이 CDK 프로젝트로 작업하기 전에 준비 스크립트 실행을 포함한 전제 조건을 설치해야 합니다.

CDK 프로젝트 만들기

먼저 빈 디렉터리에서 `cdk init`를 호출하여 새 AWS CDK 프로젝트를 생성합니다.

```
mkdir ddb_filters
cd ddb_filters
cdk init app --language python
```

`cdk init` 명령은 프로젝트 폴더의 이름을 사용하여 클래스, 하위 폴더 및 파일을 포함한 프로젝트의 다양한 요소의 이름을 지정합니다. 폴더 이름의 하이픈은 밑줄로 변환됩니다. 그렇지 않으면 이름은 Python 식별자 형식을 따라야 합니다. 예를 들어 숫자로 시작하거나 공백을 포함해서는 안 됩니다.

새 프로젝트를 사용하려면 해당 가상 환경을 활성화하세요. 이렇게 하면 프로젝트의 종속성을 전역적으로 설치하는 대신 프로젝트 폴더에 로컬로 설치할 수 있습니다.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

> **Note**
>
> 이를 Mac/Linux 명령으로 인식하여 가상 환경을 활성화할 수 있습니다. Python 템플릿에는 Windows에서 동일한 명령을 사용할 수 있도록 하는 배치 파일인 `source.bat`가 포함되어 있습니다. 기존 Windows 명령 `.venv\Scripts\activate.bat`도 작동합니다. AWS CDK Toolkit v1.70.0 이하를 사용하여 AWS CDK 프로젝트를 초기화한 경우 가상 환경은 `.venv` 대신 `.env` 디렉터리에 있습니다.

기본 인프라

선호하는 텍스트 편집기로 `./ddb_filters/ddb_filters_stack.py` 파일을 엽니다. 이 파일은 AWS CDK 프로젝트를 생성할 때 자동으로 생성되었습니다.

다음으로 `_create_ddb_table` 및 `_set_ddb_trigger_function` 함수를 추가합니다. 이러한 함수는 프로비저닝 모드 온디맨드 모드에서 파티션 키 PK 및 정렬 키 SK가 있는 DynamoDB 테이블을 생성하며 Amazon DynamoDB Streams가 기본적으로 활성화되어 새 이미지와 이전 이미지를 표시합니다.

Lambda 함수는 `app.py` 파일 아래의 `lambda` 폴더에 저장됩니다. 이 파일은 나중에 생성됩니다. 여기에는 이 스택에서 생성된 Amazon DynamoDB 테이블의 이름이 될 환경 변수 `APP_TABLE_NAME`이 포함됩니다. 동일한 함수에서 Lambda 함수에 스트림 읽기 권한을 부여합니다. 마지막으로 DynamoDB Streams를 람다 함수의 이벤트 소스로 구독합니다.

`__init__` 메서드의 파일 끝에서 각 구성을 호출하여 스택에서 초기화합니다. 추가 구성 요소와 서비스가 필요한 더 큰 프로젝트의 경우 기본 스택 외부에서 이러한 구성을 정의하는 것이 가장 좋습니다.

```
import os
import json

import aws_cdk as cdk
from aws_cdk import (
    Stack,
    aws_lambda as _lambda,
    aws_dynamodb as dynamodb,
)
from constructs import Construct


class DdbFiltersStack(Stack):
```

```
    def _create_ddb_table(self):
        dynamodb_table = dynamodb.Table(
            self,
            "AppTable",
            partition_key=dynamodb.Attribute(
                name="PK", type=dynamodb.AttributeType.STRING
            ),
            sort_key=dynamodb.Attribute(
                name="SK", type=dynamodb.AttributeType.STRING),
            billing_mode=dynamodb.BillingMode.PAY_PER_REQUEST,
            stream=dynamodb.StreamViewType.NEW_AND_OLD_IMAGES,
            removal_policy=cdk.RemovalPolicy.DESTROY,
        )

        cdk.CfnOutput(self, "AppTableName", value=dynamodb_table.table_name)
        return dynamodb_table

    def _set_ddb_trigger_function(self, ddb_table):
        events_lambda = _lambda.Function(
            self,
            "LambdaHandler",
            runtime=_lambda.Runtime.PYTHON_3_9,
            code=_lambda.Code.from_asset("lambda"),
            handler="app.handler",
            environment={
                "APP_TABLE_NAME": ddb_table.table_name,
            },
        )

        ddb_table.grant_stream_read(events_lambda)

        event_subscription = _lambda.CfnEventSourceMapping(
            scope=self,
            id="companyInsertsOnlyEventSourceMapping",
            function_name=events_lambda.function_name,
            event_source_arn=ddb_table.table_stream_arn,
            maximum_batching_window_in_seconds=1,
            starting_position="LATEST",
            batch_size=1,
        )

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)
```

```
        ddb_table = self._create_ddb_table()
        self._set_ddb_trigger_function(ddb_table)
```

이제 Amazon CloudWatch에 로그를 출력하는 매우 간단한 람다 함수를 만들어 보겠습니다. 이를 위해 lambda라는 새 폴더를 만듭니다.

```
mkdir lambda
touch app.py
```

자주 사용하는 텍스트 편집기를 사용하여 다음 내용을 app.py 파일에 추가합니다.

```
import logging

LOGGER = logging.getLogger()
LOGGER.setLevel(logging.INFO)


def handler(event, context):
    LOGGER.info('Received Event: %s', event)
    for rec in event['Records']:
        LOGGER.info('Record: %s', rec)
```

/ddb_filters/ 폴더에 있는지 확인하고 다음 명령을 입력하여 샘플 애플리케이션을 생성합니다.

```
cdk deploy
```

어느 시점에서 솔루션 배포 여부를 확인하라는 메시지가 표시됩니다. Y를 입력하여 변경 내용을 적용합니다.

```
##########################################################################################
# + # ${LambdaHandler/ServiceRole} # arn:${AWS::Partition}:iam::aws:policy/service-
role/AWSLambdaBasicExecutionRole #
##########################################################################################

Do you wish to deploy these changes (y/n)? y

...

#  Deployment time: 67.73s
```

```
Outputs:
DdbFiltersStack.AppTableName = DdbFiltersStack-AppTable815C50BC-1M1W7209V5YPP
Stack ARN:
arn:aws:cloudformation:us-east-2:111122223333:stack/
DdbFiltersStack/66873140-40f3-11ed-8e93-0a74f296a8f6
```

변경 사항이 배포되면 AWS 콘솔을 열고 테이블에 항목 하나를 추가합니다.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

이제 CloudWatch 로그에 이 항목의 모든 정보가 포함되어야 합니다.

필터링 예제

- 지정된 주와 일치하는 제품만

ddb_filters/ddb_filters/ddb_filters_stack.py 파일을 열고 "FL"과 동일한 모든 제품과 일치하는 필터를 포함하도록 수정합니다. 이는 45행의 event_subscription 바로 아래에서 수정할 수 있습니다.

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
```

```
)
```

- PK 및 SK의 일부 값으로 시작하는 항목만

다음 조건을 포함하도록 Python 스크립트를 수정합니다.

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
        ]
    },
```

- PK 및 SK의 일부 값으로 시작하거나 특정 상태에서 시작합니다.

다음 조건을 포함하도록 Python 스크립트를 수정합니다.

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
                                "Keys": {
```

```
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

필터 배열에 더 많은 요소를 추가하면 OR 조건이 추가됩니다.

정리

작업 디렉터리의 베이스에서 필터 스택을 찾고 `cdk destroy`를 실행합니다. 리소스 삭제를 확인하라는 메시지가 표시됩니다.

```
cdk destroy
Are you sure you want to delete: DdbFiltersStack (y/n)? y
```

DynamoDB Streams를 Lambda와 함께 사용하는 모범 사례

AWS Lambda 함수는 다른 함수와 격리된 실행 환경인 컨테이너 내에서 실행됩니다. 함수를 처음 실행하면 AWS Lambda에서 컨테이너를 생성하고 함수 코드를 실행하기 시작합니다.

Lambda 함수에는 호출당 한 번 실행되는 핸들러가 있습니다. 핸들러에는 함수에 대한 주요 비즈니스 로직이 포함됩니다. 예를 들어, 4단계: Lambda 함수 생성 및 테스트에 표시된 Lambda 함수에는 DynamoDB 스트림의 레코드를 처리할 수 있는 핸들러가 있습니다.

또한 컨테이너가 생성되고 나서 AWS Lambda가 핸들러를 처음 실행하기 전에 한 번만 실행되는 초기화 코드도 제공할 수 있습니다. 4단계: Lambda 함수 생성 및 테스트에 표시된 Lambda 함수에는 SDK for JavaScript in Node.js를 가져오고 Amazon SNS에 대한 클라이언트를 생성하는 초기화 코드가 있습니다. 이 객체는 핸들러 외부에서 한 번만 정의할 수 있습니다.

함수를 실행한 후, AWS Lambda는 이후 함수 호출에 컨테이너를 재사용하도록 선택할 수 있습니다. 이 경우에, 함수 핸들러에서 초기화 코드에 정의된 리소스를 재사용할 수도 있습니다. (AWS Lambda에서는 컨테이너의 보유 기간 또는 컨테이너의 재사용 여부를 제어할 수 없습니다.)

AWS Lambda를 사용하는 DynamoDB 트리거의 경우 다음 사항이 권장됩니다.

- AWS 서비스 클라이언트는 핸들러가 아니라 초기화 코드에서 인스턴스화해야 합니다. 이렇게 하면 AWS Lambda에서 기존의 연결을 재사용해 컨테이너의 수명을 정할 수 있습니다.
- 일반적으로, 연결을 명시적으로 관리하거나 연결 풀링을 구현할 필요가 없습니다. 이 작업은 AWS Lambda에서 수행되기 때문입니다.

DynamoDB 스트림의 Lambda 소비자는 정확히 한 번의 전송을 보장하지 않으며 가끔 중복이 발생할 수 있습니다. 중복 처리로 인해 예상치 못한 문제가 발생하지 않도록 Lambda 함수 코드가 멱등성을 갖도록 하세요.

자세한 내용은 AWS Lambda 개발자 안내서의 AWS Lambda 함수 작업 모범 사례를 참조하세요.

## DynamoDB 스트림 및 Apache Flink

Apache Flink로 Amazon DynamoDB Streams 레코드를 사용할 수 있습니다. Amazon Managed Service for Apache Flink를 통해 Apache Flink를 사용하여 스트리밍 데이터를 실시간으로 변환하고 분석할 수 있습니다. Apache Flink는 실시간 데이터를 처리하기 위한 오픈 소스 스트림 처리 프레임워크입니다. Apache Flink용 Amazon DynamoDB Streams 커넥터는 Apache Flink 워크로드의 구축 및 관리를 간소화하고 애플리케이션과 기타 AWS 서비스의 통합을 가능하게 합니다.

Amazon Managed Service for Apache Flink를 사용하면 로그 분석, 클릭스트림 분석, 사물 인터넷(IoT), 광고 기술, 게임 등을 위한 엔드 투 엔드 스트림 처리 애플리케이션을 빠르게 구축할 수 있습니다. 가장 일반적인 4가지 사용 사례는 스트리밍 추출, 변환, 로드(ETL), 이벤트 중심 애플리케이션, 응답형 실시간 분석 및 데이터 스트림의 대화형 쿼리입니다. Amazon DynamoDB Streams에서 Apache Flink에 쓰는 방법에 대한 자세한 내용은 Amazon DynamoDB Streams Connector를 참조하세요.

Apache Flink용 DynamoDB Streams 커넥터와 함께 ShardFilter 파라미터 사용

Amazon DynamoDB Streams는 `DescribeStream` API에서 `ShardFilter` 파라미터를 지원하여 샤드의 하위 집합을 찾습니다. 요청에 상위 샤드를 지정하면 DynamoDB Streams가 바로 밑에 있는 하위 샤드를 반환합니다. `ShardFilter` 파라미터를 사용하면 상위 샤드가 닫힌 후 하위 샤드를 빠르게 검색하여 스트림 처리 애플리케이션의 응답성과 비용 효과를 높일 수 있습니다.

Amazon Managed Service for Apache Flink는 DynamoDB Streams에서 읽을 때 `ShardFilter` 파라미터를 지원합니다. 이 기능을 사용하려면 `ShardFilter` 지원이 포함된 Flink 커넥터의 특수 버전에

대한 액세스를 요청해야 합니다. 시작하려면 AWS Support Center에서 지원 사례를 열고 이러한 바이너리에 대한 액세스를 요청합니다. Apache Flink과 함께 DynamoDB Streams 사용에 대한 자세한 내용은 DynamoDB 개발자 설명서의 스트림 판독 및 처리 섹션을 참조하세요.

# DynamoDB Accelerator(DAX)를 통한 인 메모리 가속화

Amazon DynamoDB는 규모와 성능을 위해 설계되었습니다. 대부분의 경우 DynamoDB 응답 시간은 한 자릿수 밀리초 단위로 측정할 수 있습니다. 하지만 마이크로초 단위의 응답시간이 필요한 특정 사용 사례가 있습니다. 이러한 사용 사례의 경우 DynamoDB Accelerator(DAX)는 최종적 일관된 데이터를 액세스할 때 빠른 응답 시간을 제공합니다.

DAX는 DynamoDB와 호환되는 캐싱 서비스로 까다로운 애플리케이션에서 빠른 인 메모리 성능을 활용할 수 있다는 것이 장점입니다. DAX는 세 가지 주요 시나리오에 대응합니다.

1. DAX는 인 메모리 캐시로서 최종적 일관된 읽기 워크로드의 응답 시간을 한 자릿수 밀리초에서 마이크로초까지 대폭 줄여줍니다.
2. DAX는 DynamoDB와 API 호환되는 관리형 서비스를 제공하여 운영 및 애플리케이션 복잡성을 줄여줍니다. 따라서 기존 애플리케이션에서 사용하기 위한 최소한의 기능 변경만 필요합니다.
3. 읽기 중심적이거나 일정 시간에 사용량이 급증하는 워크로드의 경우 DAX는 읽기 용량 단위를 오버프로비저닝해야 하는 필요성을 줄여 높은 처리량을 제공하고 잠재적 운영 비용을 절감합니다. 이러한 이점은 특히 개별 키를 반복적으로 읽어야 하는 애플리케이션에 유용합니다.

DAX는 서버 측 암호화를 지원합니다. 유휴 시 암호화를 통해 DAX에 의해 디스크에 유지되는 데이터가 암호화됩니다. DAX는 기본 노드에서 읽기 전용 복제본으로 변경 사항을 전파하는 작업의 일환으로 디스크에 데이터를 작성합니다. 자세한 내용은 DAX 저장 데이터 암호화 섹션을 참조하세요.

DAX는 전송 중 암호화도 지원하여 애플리케이션과 클러스터 간의 모든 요청 및 응답이 TLS(전송 수준 보안)로 암호되고 클러스터에 대한 연결은 클러스터 x509 인증서 확인을 통해 인증될 수 있도록 합니다. 자세한 내용은 DAX 전송 중 데이터 암호화 섹션을 참조하세요.

주제


- DAX 사용 사례
- DAX 사용 참고 사항
- DAX: 작동 방식
- DAX 클러스터 구성 요소
- DAX 클러스터 생성
- DAX 및 DynamoDB 정합성 모델
- DynamoDB Accelerator(DAX) 클라이언트로 개발
- DAX 클러스터 관리

- Monitoring DynamoDB Accelerator
- DAX T3/T2 버스트 가능 인스턴스
- DAX 액세스 제어
- DAX 저장 데이터 암호화
- DAX 전송 중 데이터 암호화
- DAX에 대한 서비스 연결 IAM 역할 사용
- AWS 계정 간 DAX 액세스
- DAX 클러스터 크기 조정 안내서

# DAX 사용 사례

DAX는 DynamoDB 테이블의 최종적 일관된 데이터에 대한 액세스 권한을 대기 시간이 거의 없이 제공합니다. 다중 AZ DAX 클러스터는 초당 수백만 개의 요청을 처리할 수 있습니다.

DAX는 다음과 같은 유형의 애플리케이션에 적합합니다.

- 읽기에 가장 빠른 응답 시간을 요구하는 애플리케이션. 이에 대한 몇 가지 예로 실시간 입찰, 소셜 게임, 거래 애플리케이션 등을 들 수 있습니다. DAX는 이러한 사용 사례를 위한 빠른 인 메모리 읽기 성능을 제공합니다.
- 소수의 항목을 다른 항목보다 더 자주 읽는 애플리케이션. 예를 들면, 인기 많은 상품에 대해 하루 동안 할인 행사를 진행하는 전자 상거래 시스템을 들 수 있습니다. 할인 행사 동안 다른 모든 상품에 비해 해당 상품에 대한 수요(및 DynamoDB의 해당 데이터)가 급격하게 늘어납니다. '핫' 키 및 균일하지 않은 트래픽 분포의 영향을 완화하기 위해 하루 행사가 종료될 때까지 DAX 캐시에 대한 읽기 작업을 오프로드할 수 있습니다.
- 읽기 집약적이지만 비용에 민감한 애플리케이션. DynamoDB를 사용하여 애플리케이션에 필요한 초당 읽기 수를 프로비저닝합니다. 읽기 작업이 증가하면 테이블의 할당 읽기 처리량을 추가 비용으로 늘릴 수 있습니다. 또는 애플리케이션의 작업을 DAX 클러스터로 오프로드하고 그 밖에 구매해야 할 읽기 용량 단위 수를 줄일 수도 있습니다.
- 대량 데이터 집합에 대해 반복적인 읽기가 필요한 애플리케이션. 이러한 애플리케이션은 다른 애플리케이션에서 데이터베이스 리소스를 잠재적으로 전환시킬 수 있습니다. 예를 들어 지역 날씨 데이터를 장기적으로 분석하는데 DynamoDB 테이블에서 모든 읽기 용량이 일시적으로 소비될 수 있습니다. 이러한 상황은 동일한 데이터에 액세스해야 하는 다른 애플리케이션에 부정적인 영향을 미치게 됩니다. DAX를 사용하면 캐시된 데이터에 대한 날씨 분석을 대신 수행할 수 있습니다.

DAX는 다음과 같은 유형의 애플리케이션에 적합하지 않습니다.

- 강력한 일관된 읽기가 필요한(또는 최종적 일관된 읽기를 허용할 수 없는) 애플리케이션.
- 읽기 작업에 마이크로초 단위의 응답 시간이 요구되지 않거나, 반복되는 읽기 활동을 기본 테이블에서 제거하지 않아도 되는 애플리케이션.
- 쓰기 집약적인 애플리케이션으로, 쓰기 볼륨이 크면 클러스터의 DAX 노드 간 복제가 증가합니다. 이로 인해 리소스 소비가 증가하고 가용성 문제가 발생할 위험이 있습니다.
- 반복 읽기 횟수가 많지 않은 애플리케이션으로, DAX는 캐시 적중률이 90%를 초과할 때 가장 잘 작동합니다. 캐시 적중률이 낮을수록 캐시 누락이 증가하여 DAX 클러스터 전체에서 더 많은 리소스를 소비합니다.

# DAX 사용 참고 사항

- DAX를 사용할 수 있는 AWS 리전 목록은 Amazon DynamoDB 요금을 참조하세요.
- DAX는 이러한 프로그래밍 언어의 AWS 제공 클라이언트를 사용하여 Go, Java, Node.js, Python 및 .NET로 작성된 애플리케이션을 지원합니다.
- DAX는 EC2-VPC 플랫폼에서만 사용할 수 있습니다.
- DAX 클러스터 서비스 역할 정책은 DynamoDB 테이블에 대한 메타데이터를 유지하기 위해 `dynamodb:DescribeTable` 작업을 허용해야 합니다.
- DAX 클러스터는 저장한 항목의 속성 이름에 대한 메타데이터를 유지합니다. 이러한 메타데이터는 무기한 유지됩니다(해당 항목이 만료되거나 캐시에서 제거된 이후에도). 무한한 수의 속성 이름을 사용하는 애플리케이션은 시간이 지남에 따라 DAX 클러스터에서 메모리 부족을 야기할 수 있습니다. 이러한 제한은 최상위 속성 이름에만 적용되며 중첩 속성 이름에는 적용되지 않습니다. 문제가 있는 최상위 속성 이름의 예에는 타임스탬프, UUID 및 세션 ID가 포함됩니다.

  이러한 제한은 속성 이름에만 적용되며 해당 값에는 적용되지 않습니다. 다음과 같은 항목은 문제가 되지 않습니다.

```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "CreationDate": "2017-10-24T01:02:03+00:00"
}
```

  그러나 다음과 같은 항목은 충분한 수가 있으며 타임스탬프가 서로 다른 경우에 문제가 됩니다.

```
{
    "Id": 123,
```

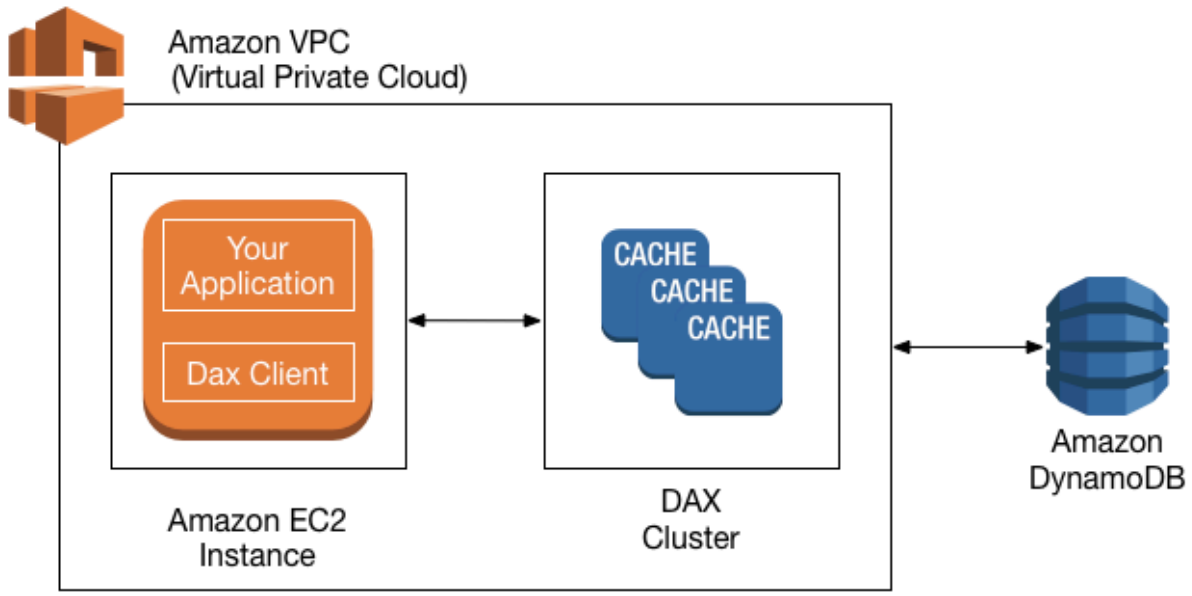
```
    "Title": "Bicycle 123",
    "2017-10-24T01:02:03+00:00": "created"
}
```

# DAX: 작동 방식

Amazon DynamoDB Accelerator(DAX)는 Amazon Virtual Private Cloud(Amazon VPC) 환경 내에서 실행되도록 설계되었습니다. Amazon VPC 서비스는 기존 데이터 센터와 매우 유사한 가상 네트워크를 정의합니다. VPC를 사용하면 IP 주소 범위, 서브넷, 라우팅 테이블, 네트워크 게이트웨이 및 보안 설정을 통제할 수 있습니다. Amazon VPC 보안 그룹을 사용하여 가상 네트워크에서 DAX 클러스터를 시작하고 클러스터에 대한 액세스를 제어할 수 있습니다.

> **Note**
>
> 2013년 12월 4일 이후 AWS 계정을 생성했다면 각 AWS 리전에 기본 VPC가 갖추어져 있습니다. 기본 VPC는 별도의 구성 단계를 수행할 필요 없이 즉시 사용할 수 있습니다.
> 자세한 내용은 Amazon VPC 사용 설명서의 기본 VPC 및 기본 서브넷을 참조하세요.

다음은 DAX에 대한 종합적인 개요를 보여주는 다이어그램입니다.



DAX 클러스터를 생성하려면 AWS Management Console을 사용합니다. 달리 지정하지 않으면 DAX 클러스터가 기본 VPC 내에서 실행됩니다. 애플리케이션을 실행하려면 Amazon EC2 인스턴스를 Amazon VPC에서 시작합니다. 그런 다음 DAX 클라이언트와 함께 애플리케이션을 EC2 인스턴스에 배포합니다.

런타임에 DAX 클라이언트는 모든 애플리케이션의 DynamoDB API 요청을 DAX 클러스터로 보냅니다. DAX는 이러한 API 결과 중 하나를 직접 처리할 수 있으면 그렇게 합니다. 그렇지 않은 경우에는 DynamoDB로 요청을 전달합니다.

마지막으로 DAX 클러스터가 결과를 애플리케이션에 반환합니다.

주제


- DAX에서 요청을 처리하는 방식
- 항목 캐시
- 쿼리 캐시


## DAX에서 요청을 처리하는 방식

DAX 클러스터는 하나 이상의 노드로 구성됩니다. 각 노드는 DAX 캐싱 소프트웨어의 자체 인스턴스를 실행합니다. 노드 중 하나는 클러스터의 기본 노드로 작동하며, 다른 노드(있는 경우)는 읽기 전용 복제본으로 작동합니다. 자세한 내용은 노드 단원을 참조하십시오.

애플리케이션은 DAX 클러스터의 엔드포인트를 지정하여 DAX에 액세스할 수 있습니다. DAX 클라이언트 소프트웨어는 클러스터 엔드포인트와 함께 작동하여 지능적인 로드 밸런싱 및 라우팅을 수행합니다.

### 읽기 작업

DAX는 다음 API 호출에 응답할 수 있습니다.

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

요청에서 최종적으로 일관된 읽기(기본 동작)를 지정하면 DAX에서 항목 읽기가 시도됩니다.

- DAX에 사용할 수 있는 항목이 있으면(캐시 적중), DAX가 DynamoDB에 액세스하지 않고 애플리케이션에 항목을 반환합니다.
- DAX에 사용할 수 있는 항목이 없으면(캐시 누락), DAX가 요청을 DynamoDB로 전달합니다. DAX는 DynamoDB에서 응답을 수신하면 애플리케이션에 결과를 반환합니다. 그러나 기본 노드의 캐시에도 그 결과를 기록합니다.

> **Note**
>
> 클러스터에 읽기 전용 복제본이 있으면 DAX가 자동으로 이 복제본을 프라이머리 노드와 동기화 상태로 유지합니다. 자세한 내용은 클러스터 단원을 참조하십시오.

요청에서 강력히 일관된 읽기를 지정하면 DAX가 해당 요청을 DynamoDB로 전달합니다. DynamoDB에서 나온 결과는 DAX에 캐시되지 않습니다. 대신에 애플리케이션에 반환되기만 합니다.

### 쓰기 작업

다음은 "라이트-스루(write-through)"로 간주되는 DAX API 작업입니다.

- `BatchWriteItem`
- `UpdateItem`
- `DeleteItem`
- `PutItem`

이 작업에서는 데이터가 먼저 DynamoDB 테이블에 기록된 다음 DAX 클러스터에 기록됩니다. 데이터가 테이블과 DAX 모두에 기록될 경우에만 작업이 성공합니다.

### 기타 작업

DAX는 모든 테이블 관리용 DynamoDB 작업을 인식하지 못합니다(예: `CreateTable`, `UpdateTable` 등). 애플리케이션에서 이러한 작업을 수행해야 하는 경우 DAX를 사용하지 말고 DynamoDB에 직접 액세스해야 합니다.

DAX 및 DynamoDB 일관성에 대한 자세한 내용은 DAX 및 DynamoDB 정합성 모델 단원을 참조하세요.

DAX에서 트랜잭션의 작동 방식에 대한 자세한 내용은 DynamoDB Accelerator(DAX)에서 트랜잭션 API 사용 단원을 참조하세요.

### 요청 속도 제한

DAX로 전송된 요청 수가 노드의 용량을 초과하는 경우 DAX는 ThrottlingException을 반환하여 추가 요청을 수락하는 속도를 제한합니다. DAX는 CPU 사용률을 지속적으로 평가하여 정상적인 클러스터 상태를 유지하면서 처리할 수 있는 요청 볼륨을 확인합니다.

DAX가 Amazon CloudWatch에 게시하는 ThrottledRequestCount 지표를 모니터링할 수 있습니다. 이러한 예외가 정기적으로 표시되는 경우 클러스터를 확장하는 것이 좋습니다.

## 항목 캐시

DAX는 `GetItem` 및 `BatchGetItem` 작업의 결과를 저장하기 위해 항목 캐시를 유지합니다. 캐시의 항목은 DynamoDB의 최종적 일관된 데이터를 나타내며 기본 키 값에 의해 저장됩니다.

애플리케이션에서 `GetItem` 또는 `BatchGetItem` 요청을 보내면, DAX가 지정된 키 값을 사용하여 항목 캐시로부터 직접 항목을 읽으려고 시도합니다. 항목이 검색되면(캐시 적중) DAX가 이를 애플리케이션에 즉시 반환합니다. 항목이 검색되지 않으면(캐시 누락) DAX가 요청을 DynamoDB에 보냅니다. DynamoDB는 최종적으로 일관된 읽기를 사용하여 요청을 처리하고 항목을 DAX에 반환합니다. DAX는 해당 항목을 항목 캐시에 저장한 다음 애플리케이션에 반환합니다.

항목 캐시에는 유지 시간(TTL) 설정이 있으며, 기본값은 5분입니다. DAX는 항목 캐시에 기록하는 모든 항목에 타임스탬프를 할당합니다. 항목이 TTL 설정보다 오래 캐시에 유지되면 항목이 만료됩니다. 만료된 항목에 대해 `GetItem` 요청을 실행하는 경우 캐시 누락으로 간주되어 DAX가 DynamoDB로 `GetItem` 요청을 보냅니다.

> **Note**
>
> 새 DAX 클러스터를 생성할 때 항목 캐시에 TTL 설정을 지정할 수 있습니다. 자세한 내용은 DAX 클러스터 관리 단원을 참조하십시오.

DAX는 항목 캐시에 대해 가장 오랫동안 사용되지 않음(LRU) 목록도 유지합니다. LRU 목록은 캐시에 항목이 처음 기록된 시간과 캐시로부터 항목이 마지막으로 읽혀진 시간을 추적합니다. 항목 캐시가 가득 차면 DAX는 새 항목을 위한 공간을 마련하기 위해 오래된 항목을 제거합니다(아직 만료되지 않은 항목 포함). LRU 알고리즘은 항목 캐시에 항상 활성화되어 있으며 사용자가 구성할 수 없습니다.

항목 캐시 TTL 설정으로 0을 지정하는 경우 항목 캐시의 항목이 LRU 제거 또는 '라이트-스루' 작업으로 인해 새로 고쳐집니다.

DAX에서 항목 캐시의 일관성에 대한 자세한 내용은 DAX 항목 캐시 동작 단원을 참조하세요.

## 쿼리 캐시

DAX는 `Query` 및 `Scan` 작업의 결과를 저장하기 위해 쿼리 캐시도 유지합니다. 이 캐시에 있는 항목은 DynamoDB 테이블에 대한 쿼리 및 스캔의 결과 집합을 나타냅니다. 이러한 결과 집합은 파라미터 값에 의해 저장됩니다.

애플리케이션에서 `Query` 또는 `Scan` 요청을 보내면, DAX가 지정된 파라미터 값을 사용하여 쿼리 캐시로부터 일치하는 결과 집합을 읽으려고 시도합니다. 결과 집합이 검색되면(캐시 적중) DAX가 이를 애플리케이션에 즉시 반환합니다. 결과 집합이 검색되지 않으면(캐시 누락) DAX가 요청을 DynamoDB에 보냅니다. DynamoDB는 최종적으로 일관된 읽기를 사용하여 요청을 처리하고 결과 집합을 DAX에 반환합니다. DAX는 결과 집합을 쿼리 캐시에 저장한 다음 애플리케이션에 반환합니다.

> **Note**
>
> 새 DAX 클러스터를 생성할 때 쿼리 캐시의 TTL 설정을 지정할 수 있습니다. 자세한 내용은 DAX 클러스터 관리 단원을 참조하십시오.

또한 DAX는 쿼리 캐시에 대한 LRU 목록을 유지합니다. 목록은 캐시에 결과 집합이 처음 기록된 시간과 캐시로부터 결과가 마지막으로 읽혀진 시간을 추적합니다. 쿼리 캐시가 가득 차면 DAX는 새 결과 집합을 위한 공간을 마련하기 위해 오래된 결과 집합을 제거합니다(아직 만료되지 않은 집합 포함). LRU 알고리즘은 쿼리 캐시에 항상 활성화되어 있으며 사용자가 구성할 수 없습니다.

쿼리 캐시 TTL 설정으로 0을 지정하면 쿼리 응답이 캐시되지 않습니다.

DAX에서 쿼리 캐시의 일관성에 대한 자세한 내용은 DAX 쿼리 캐시 동작 단원을 참조하세요.

# DAX 클러스터 구성 요소

Amazon DynamoDB Accelerator(DAX) 클러스터는 AWS 인프라 구성 요소로 이루어져 있습니다. 이 단원에서는 이러한 구성 요소 및 이들이 함께 작동하는 방법에 대해 설명합니다.

주제

- 노드
- 클러스터
- 리전 및 가용성 영역
- 파라미터 그룹
- 보안 그룹

- 클러스터 ARN
- 클러스터 엔드포인트
- 노드 엔드포인트
- 서브넷 그룹
- 이벤트
- 유지보수 윈도우

## 노드

노드는 DAX 클러스터의 가장 작은 빌딩 블록입니다. 각 노드는 DAX 소프트웨어의 인스턴스를 실행하고, 캐시된 데이터의 단일 복제본을 유지합니다.

DAX 클러스터는 다음 두 가지 방법 중 하나로 크기를 조정할 수 있습니다.

- 추가 노드를 클러스터에 추가하는 방법. 이 방법을 사용하면 클러스터의 전체 읽기 처리량이 증가합니다.
- 대형 노드 유형을 사용하는 방법. 대형 노드 유형은 용량이 더 크며 처리량이 늘어날 수 있습니다. (새로운 노드 유형으로 새 클러스터를 생성해야 한다는 점에 유의하세요.)

클러스터 내 모든 노드는 노드 유형이 동일하며, 동일한 DAX 캐싱 소프트웨어를 실행합니다. 이용 가능한 노드 유형의 목록은 Amazon DynamoDB 요금을 참조하세요.

## 클러스터

클러스터는 DAX가 단위로 관리하는 하나 이상 노드의 논리적 그룹입니다. 클러스터의 노드 중 하나는 기본 노드로 지정되며, 다른 노드들(있는 경우)은 읽기 전용 복제본이 됩니다.

기본 노드는 다음을 담당합니다.

- 캐시된 데이터에 대한 애플리케이션 요청 이행
- DynamoDB에 대한 쓰기 작업 처리
- 클러스터의 제거 정책에 따라 캐시에서 데이터 제거.

프라이머리 노드의 캐시된 데이터가 변경되면 DAX는 복제 로그를 사용하여 모든 읽기 전용 복제본 노드에 변경 내용을 전파합니다. DynamoDB는 모든 읽기 전용 복제본에서 확인을 받은 후 프라이머리 노드에서 복제 로그를 삭제합니다.

DAX 클러스터는 클러스터당 최대 11개까지 노드를 지원할 수 있습니다(프라이머리 노드와 최대 10개의 읽기 전용 복제본).

읽기 전용 복제본은 다음을 담당합니다.

- 캐시된 데이터에 대한 애플리케이션 요청 이행
- 클러스터의 제거 정책에 따라 캐시에서 데이터 제거.

하지만 프라이머리 노드와 달리 읽기 전용 복제본은 DynamoDB에 쓰지 않습니다.

읽기 전용 복제본은 다음과 같은 2가지 역할을 추가로 수행합니다.

- 확장성. 동시 액세스가 필요한 클라이언트가 많은 경우 읽기 확장을 위해 복제본을 더 추가할 수 있습니다. DAX는 클러스터의 모든 노드에 균일하게 로드를 분산합니다. (처리량을 늘리는 또 다른 방법은 더 큰 캐시 노드 유형을 사용하는 것입니다.)
- 고가용성. 프라이머리 노드에 장애가 발생하면 DAX가 자동으로 읽기 전용 복제본으로 장애 조치하고 이를 새로운 프라이머리 노드로 지정합니다. 복제본 노드가 실패하면 실패한 노드가 복구될 때까지 DAX 클러스터의 다른 노드가 계속 요청을 수행할 수 있습니다. 내결함성을 최대화하려면 별도의 가용 영역에 읽기 전용 복제본을 배포해야 합니다. 이렇게 구성하면 전체 가용 영역을 사용할 수 없게 되는 경우에도 DAX 클러스터가 계속 작동할 수 있습니다.

> **Important**
>
> 프로덕션 용도로는 각 노드가 서로 다른 가용 영역에 있는 최소 3개 노드로 구성된 DAX를 사용할 것을 적극 권장합니다. DAX 클러스터를 내결함성(fault-tolerant)으로 구성하려면 3개의 노드가 필요합니다.
> 개발 또는 테스트 워크로드를 위해 DAX 클러스터를 1개 또는 2개의 노드에 배포할 수 있습니다. 1개 및 2개 노드 클러스터는 내결함성 구성이 아니므로 프로덕션 용도로는 최소 3개 이상의 노드를 사용하세요. 1개 또는 2개 노드 클러스터에서 소프트웨어 오류나 하드웨어 오류가 발생할 경우, 클러스터를 사용할 수 없게 되거나 캐시된 데이터가 손실될 수 있습니다.

> **Important**
>
> DAX 클러스터는 최대 500개의 DynamoDB 테이블을 지원합니다. 테이블이 500개를 초과하면 클러스터의 가용성과 성능이 저하될 수 있습니다.

## 리전 및 가용성 영역

AWS 리전에 있는 DAX 클러스터는 동일한 리전에 있는 DynamoDB 테이블과만 상호 작용할 수 있습니다. 이러한 이유로 올바른 리전에서 DAX 클러스터를 시작하도록 주의해야 합니다. 다른 리전에 DynamoDB 테이블이 있는 경우에는 해당 리전에서도 DAX 클러스터를 시작해야 합니다.

각 리전은 다른 리전에서 완전히 격리되도록 설계되었습니다. 각 리전 안에는 가용 영역이 여러 개 있습니다. 서로 다른 가용 영역에서 노드를 시작하면 가능한 최고 수준의 내결함성을 갖출 수 있습니다.

> **Important**
>
> 하나의 가용 영역에 클러스터 노드를 모두 두지는 마세요. 이렇게 구성하면 가용 영역에 장애가 발생한 경우 DAX 클러스터를 사용할 수 없게 됩니다.
> 프로덕션 용도로는 각 노드가 서로 다른 가용 영역에 있는 최소 3개 노드로 구성된 DAX를 사용할 것을 적극 권장합니다. DAX 클러스터를 내결함성(fault-tolerant)으로 구성하려면 3개의 노드가 필요합니다.
> 개발 또는 테스트 워크로드를 위해 DAX 클러스터를 1개 또는 2개의 노드에 배포할 수 있습니다. 1개 및 2개 노드 클러스터는 내결함성 구성이 아니므로 프로덕션 용도로는 최소 3개 이상의 노드를 사용하세요. 1개 또는 2개 노드 클러스터에서 소프트웨어 오류나 하드웨어 오류가 발생할 경우, 클러스터를 사용할 수 없게 되거나 캐시된 데이터가 손실될 수 있습니다.

## 파라미터 그룹

파라미터 그룹은 DAX 클러스터의 런타임 설정을 관리하는 데 사용됩니다. DAX에는 성능을 최적화하는 데 사용할 수 있는 파라미터가 여러 개 있습니다(예: 캐시된 데이터에 대한 TTL 정책 정의). 파라미터 그룹이란 클러스터에 적용할 수 있는 이름이 지정된 파라미터 모음을 말합니다. 따라서 해당 클러스터에 있는 모든 노드가 정확히 동일한 방법으로 구성되게 할 수 있습니다.

## 보안 그룹

DAX 클러스터는 Amazon Virtual Private Cloud(Amazon VPC) 환경에서 실행됩니다. 이 환경은 해당 AWS 계정 전용이며 다른 VPC와 분리되는 가상 네트워크입니다. 보안 그룹은 인바운드 및 아웃바운드 네트워크 트래픽을 제어할 수 있는 VPC에 대한 가상 방화벽 역할을 합니다.

VPC에서 클러스터를 시작할 경우 보안 그룹에 수신 규칙을 추가하여 수신 네트워크 트래픽을 허용할 수 있습니다. 수신 규칙은 클러스터에 대해 프로토콜(TCP)과 포트 번호(8111)를 지정합니다. 이 수신 규칙을 보안 그룹에 추가하면 VPC 내에서 실행 중인 애플리케이션에서 DAX 클러스터에 액세스할 수 있습니다.

## 클러스터 ARN

모든 DAX 클러스터에는 Amazon 리소스 이름(ARN)이 할당됩니다. ARN 형식은 다음과 같습니다.

```
arn:aws:dax:region:accountID:cache/clusterName
```

IAM 정책의 클러스터 ARN을 사용하여 DAX API 작업에 대한 권한을 정의합니다. 자세한 내용은 DAX 액세스 제어 단원을 참조하십시오.

## 클러스터 엔드포인트

모든 DAX 클러스터는 애플리케이션에서 사용할 클러스터 엔드포인트를 제공합니다. 애플리케이션에서 해당 엔드포인트를 사용하여 클러스터에 액세스하면 클러스터에 있는 개별 노드의 호스트 이름과 포트 번호를 알 필요가 없습니다. 읽기 전용 복제본을 추가하거나 제거하더라도 애플리케이션은 클러스터에 있는 모든 노드를 자동으로 "알게" 됩니다.

다음은 전송 중 암호화를 사용하도록 구성되지 않은 us-east-1 리전의 클러스터 엔드포인트 예제입니다.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

다음은 전송 중 암호화를 사용하도록 구성된 동일한 리전의 클러스터 엔드포인트 예제입니다.

```
daxs://my-encrypted-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

## 노드 엔드포인트

DAX 클러스터의 각 개별 노드에는 자체 호스트 이름과 포트 번호가 있습니다. 다음은 노드 엔드포인트 예제입니다.

```
myDAXcluster-a.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111
```

애플리케이션은 엔드포인트를 사용하여 직접 노드에 액세스할 수 있습니다. 그러나, DAX 클러스터를 단일 단위로 간주한 후 클러스터 엔드포인트를 대신 사용하여 액세스하는 것이 좋습니다. 클러스터 엔드포인트를 사용하면 애플리케이션이 노드 목록을 유지할 필요가 없고, 클러스터에서 노드를 추가하거나 삭제할 때 이 목록을 최신 상태로 업데이트할 필요가 없습니다.

## 서브넷 그룹

DAX 클러스터 노드에 대한 액세스는 Amazon VPC 환경 내의 Amazon EC2 인스턴스에서 실행되는 애플리케이션으로만 제한됩니다. 서브넷 그룹을 사용하여 특정 서브넷에서 실행 중인 Amazon EC2 인스

턴스에서의 액세스 권한을 클러스터에 부여할 수 있습니다. 서브넷 그룹은 Amazon VPC 환경에서 실행 중인 클러스터에 대해 지정할 수 있는 서브넷(일반적으로 프라이빗 서브넷) 모음입니다.

DAX 클러스터를 생성할 때 서브넷 그룹을 지정해야 합니다. DAX는 해당 서브넷 그룹을 사용하여 노드에 연결된 서브넷 내의 서브넷 및 IP 주소를 선택합니다.

## 이벤트

DAX는 노드 추가 실패, 노드 추가 성공, 보안 그룹 변경과 같은 중요한 이벤트를 클러스터 내에 기록합니다. 주요 이벤트를 모니터링하면 클러스터의 현재 상태를 파악할 수 있으며, 이벤트에 따라 교정 작업을 수행할 수도 있습니다. DAX 관리 API의 AWS Management Console 또는 `DescribeEvents` 작업을 사용하여 이러한 이벤트에 액세스할 수 있습니다.

또한 해당 알림을 특정 Amazon Simple Notification Service(Amazon SNS) 주제에 전송하도록 요청할 수 있습니다. 이렇게 하면 DAX 클러스터에서 이벤트가 발생하는 경우 즉시 알 수 있습니다.

## 유지보수 윈도우

모든 클러스터에는 시스템 변경 사항을 적용할 수 있는 주 단위 유지 관리 기간이 있습니다. 변경 사항이 순차적으로 적용되면 기존 노드가 교체되고 변경 사항이 적용된 새 노드가 클러스터에 추가됩니다. 이 기간 동안 애플리케이션에서 일시적인 오류나 제한이 발생할 수 있습니다. 따라서 사용량이 가장 낮은 시간으로 유지 관리 기간을 예약하고 필요에 따라 이 일정을 주기적으로 조정하는 것이 좋습니다. 요청한 유지 관리 작업을 수행하는 기간을 최대 24시간까지 지정할 수 있습니다.

캐시 클러스터를 생성하거나 수정할 때 선호하는 유지 관리 기간을 지정하지 않으면 DAX는 평일에 60분 유지 관리 기간을 임의로 할당합니다. 이 60분 유지 관리 기간은 각 AWS 리전에 대해 8시간 블록에서 임의로 선택됩니다. 다음 표는 기본 유지 관리 기간이 할당된 각 리전의 시간 블록 목록입니다.

| 리전 코드 | 리전 이름 | 유지보수 윈도우 |
|---|---|---|
| ap-northeast-1 | 아시아 태평양(도쿄) 리전 | 13:00~21:00 UTC |
| ap-southeast-1 | 아시아 태평양(싱가포르) 리전 | 14:00~22:00 UTC |
| ap-southeast-2 | 아시아 태평양(시드니) 리전 | 12:00~20:00 UTC |
| ap-south-1 | Asia Pacific (Mumbai) Region | 17:30~1:30 UTC |
| cn-northwest-1 | 중국(닝샤) 리전 | 23:00~07:00 UTC |

| 리전 코드 | 리전 이름 | 유지보수 윈도우 |
| --- | --- | --- |
| cn-north-1 | 중국(베이징) 리전 | 14:00~22:00 UTC |
| eu-central-1 | Europe (Frankfurt) Region | 23:00~07:00 UTC |
| eu-north-1 | Europe (Stockholm) Region | 01:00~09:00 UTC |
| eu-south-2 | 유럽(스페인) 리전 | 21:00–05:00 UTC |
| eu-west-1 | Europe (Ireland) Region | 22:00~06:00 UTC |
| eu-west-2 | Europe (London) Region | 23:00~07:00 UTC |
| eu-west-3 | Europe (Paris) Region | 23:00~07:00 UTC |
| sa-east-1 | South America (São Paulo) Region | 01:00~09:00 UTC |
| us-east-1 | 미국 동부(버지니아 북부) 리전 | 03:00~11:00 UTC |
| us-east-2 | US East (Ohio) Region | 23:00~07:00 UTC |
| us-west-1 | US West (N. California) Region | 06:00~14:00 UTC |
| us-west-2 | US West (Oregon) Region | 06:00~14:00 UTC |

# DAX 클러스터 생성

이 단원에서는 기본 Amazon Virtual Private Cloud(Amazon VPC) 환경에서 Amazon DynamoDB Accelerator(DAX)를 처음으로 설정하고 사용하는 절차를 알아봅니다. AWS Command Line Interface(AWS CLI) 또는 AWS Management Console을 사용하여 처음으로 DAX 클러스터를 생성할 수 있습니다.

DAX 클러스터를 생성한 후에는 동일한 VPC에서 실행 중인 Amazon EC2 인스턴스에서 해당 클러스터에 액세스할 수 있습니다. 그러면 애플리케이션 프로그램에서 DAX 클러스터를 사용할 수 있습니다. 자세한 내용은 DynamoDB Accelerator(DAX) 클라이언트로 개발 섹션을 참조하세요.

주제

- DAX에 대한 IAM 서비스 역할을 생성하여 DynamoDB 액세스
- AWS CLI을 사용하여 DAX 클러스터 생성
- AWS Management Console을 사용하여 DAX 클러스터 생성

# DAX에 대한 IAM 서비스 역할을 생성하여 DynamoDB 액세스

DAX 클러스터가 사용자를 대신하여 DynamoDB 테이블에 액세스해야 할 경우 서비스 역할을 만들어야 합니다. 서비스 역할은 사용자를 대신하여 AWS 서비스를 수행하도록 허가하는 AWS Identity and Access Management(IAM) 역할입니다. 서비스 역할은 사용자가 테이블에 액세스하듯이 DAX가 DynamoDB 테이블에 액세스할 수 있도록 허용합니다. DAX 클러스터를 생성하려면 먼저 서비스 역할을 만들어야 합니다.

콘솔을 사용하는 경우, 클러스터를 생성하는 워크플로에서 기존 DAX 서비스 역할이 있는지 확인합니다. 서비스 연결이 없으면 콘솔이 새로운 서비스 역할을 생성합니다. 자세한 내용은 the section called “2단계: DAX 클러스터 생성” 섹션을 참조하세요.

AWS CLI를 사용하는 경우 이전에 생성한 DAX 서비스 역할을 지정해야 합니다. 그렇지 않으면 먼저 서비스 역할을 새로 만들어야 합니다. 자세한 내용은 1단계: AWS CLI을 사용하여 DAX에서 DynamoDB에 액세스하는 IAM 서비스 역할 생성 섹션을 참조하세요.

## 서비스 역할 생성에 필요한 권한

AWS 관리형 `AdministratorAccess` 정책은 DAX 클러스터 및 서비스 역할을 생성하는 데 필요한 모든 권한을 제공합니다. 사용자가 연결된 `AdministratorAccess`를 가지고 있으면 추가 작업이 필요하지 않습니다.

그렇지 않을 경우 사용자가 서비스 역할을 만들 수 있도록 다음 권한을 IAM 정책에 추가해야 합니다.

- `iam:CreateRole`
- `iam:CreatePolicy`
- `iam:AttachRolePolicy`
- `iam:PassRole`

작업을 수행하려 하는 사용자에게 이러한 권한을 연결합니다.

> **Note**
>
> `iam:CreateRole`, `iam:CreatePolicy`, `iam:AttachRolePolicy`, `iam:PassRole` 권한은 AWS에서 관리하는 DynamoDB용 정책에 포함되지 않습니다. 이러한 권한은 권한 승격 가능성, 즉 사용자가 이러한 권한을 사용하여 새로운 관리 정책을 만든 후 기존 역할에 해당 정책을 연결할 가능성이 있기 때문에 설계상 포함되지 않았습니다. 따라서 DAX 클러스터 관리자는 이러한 권한을 해당 정책에 명시적으로 추가해야 합니다.

## 문제 해결

사용자 정책에 `iam:CreateRole`, `iam:CreatePolicy` 및 `iam:AttachPolicy` 권한이 없는 경우 오류 메시지가 표시됩니다. 다음 표에는 이러한 메시지와 이 문제를 해결하는 방법이 나와 있습니다.

| 오류 메시지 | 다음을 수행합니다. |
| --- | --- |
| `User: arn:aws:iam::` *accountID* `:user/`*userName* `is not authorized to perform: iam:CreateRole on resource: arn:aws:iam::` *accountID* `:role/service-role/` *roleName* | 사용자 정책에 `iam:CreateRole` 을 추가합니다. |
| `User: arn:aws:iam::` *accountID* `:user/`*userName* `is not authorized to perform: iam:CreatePolicy on resource: policy` *policyName* | 사용자 정책에 `iam:CreatePolicy` 을 추가합니다. |
| `User: arn:aws:iam::` *accountID* `:user/`*userName* `is not authorized to perform: iam:AttachRolePolicy on resource: role` *daxServiceRole* | 사용자 정책에 `iam:AttachRolePolicy` 을 추가합니다. |

DAX 클러스터 관리에 필요한 IAM 정책에 대한 자세한 내용은 DAX 액세스 제어 단원을 참조하세요.

# AWS CLI을 사용하여 DAX 클러스터 생성

이 단원에서는 AWS Command Line Interface(AWS CLI)를 사용하여 Amazon DynamoDB Accelerator(DAX) 클러스터를 생성하는 방법을 설명합니다. AWS CLI를 아직 설치하지 않았다면 이를 설치하고 구성해야 합니다. 이를 위해 AWS Command Line Interface 사용 설명서에서 다음 지침을 참조하세요.

- AWS CLI 설치
- AWS CLI 구성

> **Important**
>
> AWS CLI를 사용하여 DAX 클러스터를 관리하려면 1.11.110 버전 이상을 설치하거나 업그레이드하세요.

모든 AWS CLI 예제는 `us-west-2` 리전과 가상의 계정 ID를 사용합니다.

주제


- 1단계: AWS CLI을 사용하여 DAX에서 DynamoDB에 액세스하는 IAM 서비스 역할 생성
- 2단계: 서브넷 그룹 생성
- 3단계: AWS CLI를 사용하여 DAX 클러스터 생성
- 4단계: AWS CLI을 사용하여 보안 그룹 인바운드 규칙 구성


## 1단계: AWS CLI을 사용하여 DAX에서 DynamoDB에 액세스하는 IAM 서비스 역할 생성

Amazon DynamoDB Accelerator(DAX) 클러스터를 생성하려면 먼저 이에 대한 서비스 역할을 생성해야 합니다. 서비스 역할은 사용자를 대신하여 AWS 서비스를 수행하도록 허가하는 AWS Identity and Access Management(IAM) 역할입니다. 사용자가 테이블에 액세스하듯이 서비스 역할을 통해 DAX가 DynamoDB 테이블에 액세스할 수 있습니다.

이 단계에서는 IAM 정책을 생성한 후 이 정책을 IAM 역할에 연결합니다. 이렇게 하면 사용자가 역할을 DAX 클러스터에 할당할 수 있으므로 클러스터가 사용자 대신 DynamoDB 작업을 수행할 수 있습니다.

DAX에 대한 IAM 서비스 역할을 생성하려면

1. 다음 콘텐츠를 가진 `service-trust-relationship.json`이라는 파일을 생성합니다:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. 서비스 역할을 생성합니다.

```
aws iam create-role \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --assume-role-policy-document file://service-trust-relationship.json
```

3. 다음 콘텐츠를 가진 `service-role-policy.json`이라는 파일을 생성합니다:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
```

```
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333:*"
            ]
        }
    ]
}
```

*accountID*를 해당 AWS 계정 ID로 바꿉니다. AWS 계정 ID를 찾으려면 콘솔의 상단 오른쪽 모서리에서 로그인 ID를선택합니다. 드롭다운 메뉴에 AWS 계정 ID가 표시됩니다.

이 예에서 Amazon 리소스 이름(ARN)의 *accountID*는 12자리여야 합니다. 하이픈이나 다른 구두점을 사용하지 마세요.

4. 서비스 역할에 대한 IAM 정책을 생성합니다.

```
aws iam create-policy \
    --policy-name DAXServicePolicyForDynamoDBAccess \
    --policy-document file://service-role-policy.json
```

다음 예제에서와 같이 출력에 표시된 생성 정책의 ARN을 기록해 둡니다.

```
arn:aws:iam::123456789012:policy/DAXServicePolicyForDynamoDBAccess
```

5. 이 정책을 서비스 역할에 연결합니다. 다음 코드의 *arn*을 전 단계에서 확인한 실제 역할 ARN으로 바꿉니다.

```
aws iam attach-role-policy \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --policy-arn arn
```

다음으로 기본 VPC의 서브넷 그룹을 지정합니다. 서브넷 그룹은 VPC 내에 있는 하나 이상의 서브넷 모음입니다. 2단계: 서브넷 그룹 생성을(를) 참조하세요.

## 2단계: 서브넷 그룹 생성

다음 절차를 따라 AWS Command Line Interface(AWS CLI)를 사용하여 Amazon DynamoDB Accelerator(DAX) 클러스터의 서브넷 그룹을 생성합니다.

> **Note**
>
> 기본 VPC에 대해 서브넷 그룹을 이미 생성했다면 이 단계를 생략할 수 있습니다.

DAX는 Amazon Virtual Private Cloud(Amazon VPC) 환경 내에서 실행되도록 설계되었습니다. 2013년 12월 4일 이후 AWS 계정을 생성했다면 각 AWS 리전에 기본 VPC가 갖추어져 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 기본 VPC 및 기본 서브넷을 참조하세요.

> **Note**
>
> 이 DAX 클러스터가 있는 VPC에는 다른 리소스는 물론 다른 서비스용 VPC 엔드포인트(ElastiCache용 VPC 엔드포인트 제외)가 포함될 수 있으며, 이로 인해 DAX 클러스터 작업에 오류가 발생할 수 있습니다.

서브넷 그룹 생성

1. 기본 VPC의 식별자를 확인하려면 다음 명령을 입력합니다.

   ```
   aws ec2 describe-vpcs
   ```

   다음 예제에서와 같이 출력에 표시된 기본 VPC의 식별자를 적어 둡니다.

   ```
   vpc-12345678
   ```

2. 기본 VPC와 연결된 서브넷 ID를 확인합니다. *vpcID*를 실제 VPC ID(예: vpc-12345678)로 바꿉니다.

   ```
   aws ec2 describe-subnets \
       --filters "Name=vpc-id,Values=vpcID" \
       --query "Subnets[*].SubnetId"
   ```

   출력에 표시된 서브넷 식별자(예: subnet-11111111)를 기록해 둡니다.

3. 서브넷 그룹을 생성합니다. --subnet-ids 파라미터에 서브넷 ID를 1개 이상 지정해야 합니다.

   ```
   aws dax create-subnet-group \
       --subnet-group-name my-subnet-group \
       --subnet-ids subnet-11111111 subnet-22222222 subnet-33333333 subnet-44444444
   ```

클러스터를 생성하는 방법은 3단계: AWS CLI를 사용하여 DAX 클러스터 생성 단원을 참조하세요.

## 3단계: AWS CLI를 사용하여 DAX 클러스터 생성

다음 절차를 따라 AWS Command Line Interface(AWS CLI)를 사용하여 기본 Amazon VPC에 Amazon DynamoDB Accelerator(DAX) 클러스터를 생성합니다.

DAX 클러스터를 생성하려면

1. 서비스 역할의 Amazon 리소스 이름(ARN)을 확인합니다.

```
aws iam get-role \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --query "Role.Arn" --output text
```

   다음 예제에서와 같이 출력에 표시된 서비스 역할 ARN을 적어 둡니다.

```
arn:aws:iam::123456789012:role/DAXServiceRoleForDynamoDBAccess
```

2. DAX 클러스터를 생성합니다. *roleARN*을 전 단계의 ARN으로 바꿉니다.

```
aws dax create-cluster \
    --cluster-name mydaxcluster \
    --node-type dax.r4.large \
    --replication-factor 3 \
    --iam-role-arn roleARN \
    --subnet-group my-subnet-group \
    --sse-specification Enabled=true \
    --region us-west-2
```

   클러스터의 모든 노드는 유형이 `dax.r4.large`(`--node-type`)입니다. 프라이머리 노드 1개와 복제본 2개, 총 3개의 노드(`--replication-factor`)가 생성됩니다.

> **Note**
>
> `sudo` 및 `grep`는 예약된 키워드이므로 DAX 클러스터를 생성할 때 이러한 단어를 클러스터 이름에 사용할 수 없습니다. 예를 들어 `sudo` 및 `sudocluster`는 잘못된 클러스터 이름입니다.

클러스터 상태를 보려면 다음 명령을 입력합니다.

```
aws dax describe-clusters
```

출력에 상태(예: "Status": "creating")가 표시됩니다.

> **Note**
>
> 클러스터를 생성하는 데는 몇 분 정도가 걸립니다. 클러스터가 준비되면 상태가 available로 변경됩니다. 그 동안 4단계: AWS CLI을 사용하여 보안 그룹 인바운드 규칙 구성 단원으로 이동하여 해당 지침을 따릅니다.

## 4단계: AWS CLI을 사용하여 보안 그룹 인바운드 규칙 구성

Amazon DynamoDB Accelerator(DAX) 클러스터의 노드는 Amazon VPC의 기본 보안 그룹을 사용합니다. 기본 보안 그룹을 사용하려면 암호화되지 않은 클러스터는 TCP 포트 8111에서, 암호화된 클러스터는 포트 9111에서 인바운드 트래픽을 허가해야 합니다. 이렇게 하면 Amazon VPC의 Amazon EC2 인스턴스가 DAX 클러스터에 액세스할 수 있습니다.

> **Note**
>
> DAX 클러스터를 default가 아닌 다른 보안 그룹으로 시작했다면 대신 해당 그룹에 대해 다음 절차를 수행해야 합니다.

보안 그룹 인바운드 규칙을 구성하려면

1. 기본 보안 그룹 식별자를 확인하려면 다음 명령을 입력합니다. *vpcID*를 실제 VPC ID(2단계: 서브넷 그룹 생성 참조)로 바꿉니다.

   ```
   aws ec2 describe-security-groups \
       --filters Name=vpc-id,Values=vpcID Name=group-name,Values=default \
       --query "SecurityGroups[*].{GroupName:GroupName,GroupId:GroupId}"
   ```

   출력에 표시된 보안 그룹 식별자(예: sg-01234567)를 메모해 둡니다.
2. 그리고 다음을 입력합니다. *sgID*를 실제 보안 그룹 식별자로 바꿉니다. 포트는 암호화되지 않은 클러스터에는 8111, 암호화된 클러스터에는 9111을 사용합니다.

   ```
   aws ec2 authorize-security-group-ingress \
   ```

```
    --group-id sgID --protocol tcp --port 8111
```

# AWS Management Console을 사용하여 DAX 클러스터 생성

이 단원에서는 AWS Management Console을 사용하여 Amazon DynamoDB Accelerator(DAX) 클러스터를 생성하는 방법을 설명합니다.

주제

- 1단계: AWS Management Console을 사용하여 서브넷 그룹 생성
- 2단계: AWS Management Console을 사용하여 DAX 클러스터 생성
- 3단계: AWS Management Console을 사용하여 보안 그룹 인바운드 규칙 구성

## 1단계: AWS Management Console을 사용하여 서브넷 그룹 생성

다음 절차를 따라 AWS Management Console을 사용하여 Amazon DynamoDB Accelerator(DAX) 클러스터의 서브넷 그룹을 생성합니다.

> **Note**
>
> 기본 VPC에 대해 서브넷 그룹을 이미 생성했다면 이 단계를 생략할 수 있습니다.

DAX는 Amazon Virtual Private Cloud(Amazon VPC) 환경 내에서 실행되도록 설계되었습니다. 2013년 12월 4일 이후 AWS 계정을 생성했다면 각 AWS 리전에 기본 VPC가 갖추어져 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 기본 VPC 및 기본 서브넷을 참조하세요.

DAX 클러스터 생성 프로세스 중에 사용자는 서브넷 그룹을 지정해야 합니다. 서브넷 그룹은 VPC 내에 있는 하나 이상의 서브넷 모음입니다. DAX 클러스터를 생성하면 노드가 서브넷 그룹 내의 서브넷에 배포됩니다.

> **Note**
>
> 이 DAX 클러스터가 있는 VPC에는 다른 리소스는 물론 다른 서비스용 VPC 엔드포인트(ElastiCache용 VPC 엔드포인트 제외)가 포함될 수 있으며, 이로 인해 DAX 클러스터 작업에 오류가 발생할 수 있습니다.

서브넷 그룹을 생성하려면

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 탐색 창의 DAX에서 서브넷 그룹(Subnet groups)을 선택합니다.
3. [서브넷 그룹 생성]을 선택합니다.
4. [Create subnet group] 창에서 다음을 수행하세요.

   a. 이름 - 서브넷 그룹의 짧은 이름을 입력합니다.
   b. 설명 - 서브넷 그룹에 대한 설명을 입력합니다.
   c. VPC ID - Amazon VPC 환경의 식별자를 선택합니다.
   d. 서브넷 - 목록에서 하나 이상의 서브넷을 선택합니다.

   > **Note**
   >
   > 서브넷은 여러 가용 영역에 분산되어 있습니다. 다중 노드의 DAX 클러스터(프라이머리 노드 한 개와 읽기 전용 복제본 여러 개)를 생성할 계획이라면 서브넷 ID를 여러 개 선택하는 것이 좋습니다. 그러면 DAX가 여러 가용 영역으로 클러스터 노드를 배포할 수 있습니다. 가용 영역이 사용할 수 없게 되면 DAX는 남아 있는 가용 영역으로 자동으로 장애 조치합니다. DAX 클러스터는 중단 없이 계속해서 작동합니다.

   원하는 대로 설정되었으면 서브넷 그룹 생성을 클릭합니다.

클러스터를 생성하는 방법은 2단계: AWS Management Console을 사용하여 DAX 클러스터 생성 단원을 참조하세요.

## 2단계: AWS Management Console을 사용하여 DAX 클러스터 생성

다음 절차를 따라 기본 Amazon VPC에 Amazon DynamoDB Accelerator(DAX) 클러스터를 생성합니다.

DAX 클러스터를 생성하려면

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 탐색 창의 DAX에서 클러스터를 선택합니다.
3. 클러스터 생성을 선택합니다.
4. [Create cluster] 창에서 다음을 수행하세요.

a. 클러스터 이름 - DAX 클러스터에 대한 간략한 이름을 입력합니다.

> **Note**
>
> sudo 및 grep는 예약된 키워드이므로 DAX 클러스터를 생성할 때 이러한 단어를 클러스터 이름에 사용할 수 없습니다. 예를 들어 sudo 및 sudocluster는 잘못된 클러스터 이름입니다.

b. 클러스터 설명 - 클러스터에 대한 설명을 입력합니다.

c. 노드 유형(Node types) - 클러스터에 있는 모든 노드의 노드 유형을 선택합니다.

d. 클러스터 크기(Cluster size) - 클러스터의 노드 수를 선택합니다. 클러스터는 기본 노드 한 개와 읽기 전용 복제본 최대 아홉 개로 구성됩니다.

> **Note**
>
> 단일 노드 클러스터를 생성하려면 [1]을 선택합니다. 그러면 기본 노드 한 개로 클러스터가 구성됩니다.
> 다중 노드 클러스터를 생성하려면 3(기본 노드 1개와 읽기 전용 복제본 2개)에서 10(기본 노드 1개와 읽기 전용 복제본 9개) 사이의 숫자를 선택합니다.
>
> > **Important**
> >
> > 프로덕션 용도로는 각 노드가 서로 다른 가용 영역에 있는 최소 3개 이상의 노드로 구성된 DAX를 사용할 것을 적극 권장합니다. DAX 클러스터를 내결함성(fault-tolerant)으로 구성하려면 3개의 노드가 필요합니다.
> > 개발 또는 테스트 워크로드를 위해 DAX 클러스터를 1개 또는 2개의 노드에 배포할 수 있습니다. 1개 및 2개 노드 클러스터는 내결함성 구성이 아니므로 프로덕션 용도로는 최소 3개 이상의 노드를 사용하세요. 1개 또는 2개 노드 클러스터에서 소프트웨어 오류나 하드웨어 오류가 발생할 경우, 클러스터를 사용할 수 없게 되거나 캐시된 데이터가 손실될 수 있습니다.

e. Next(다음)를 선택합니다.

f. 서브넷 그룹(Subnet group) - 기존 그룹 선택(Choose existing) 및 1단계: AWS Management Console을 사용하여 서브넷 그룹 생성에서 생성한 서브넷 그룹 선택을 선택합니다.

g. 액세스 제어(Access control) - 기본(Default) 보안 그룹을 선택합니다.

h. 가용 영역(AZ)(Availability Zones (AZ)) - 자동(Automatic)을 선택합니다.

i. 다음을 선택합니다.

j. IAM service role for DynamoDB access(DynamoDB 액세스를 위한 IAM 서비스 역할) - Create new(새로 생성)를 선택하고 다음 정보를 입력합니다.

- IAM 역할 이름 - 역할의 이름(예: `DAXServiceRole`)을 입력합니다. 그러면 콘솔에서 새 IAM 역할을 만들고, DAX 클러스터는 런타임에 이 역할을 수임합니다.
- 정책 생성(Create policy) 옆의 상자를 선택합니다.
- IAM 역할 정책 - 읽기/쓰기를 선택합니다. 그러면 DAX 클러스터가 DynamoDB에서 읽기 및 쓰기 작업을 수행할 수 있습니다.
- 새 IAM 정책 이름 - IAM 역할 이름을 입력하면 이 필드가 채워집니다. IAM 정책의 이름(예: `DAXServicePolicy`)을 입력할 수도 있습니다. 그러면 콘솔에서 새 IAM 정책을 만들고 이 정책을 IAM 역할에 연결합니다.
- DynamoDB 테이블에 대한 액세스 - 모든 테이블을 선택합니다.

k. 암호화 - 저장 중 암호화 켜기 및 전송 중 암호화 켜기를 선택합니다. 자세한 내용은 DAX 저장 데이터 암호화 및 DAX 전송 중 데이터 암호화 섹션을 참조하세요.

DAX가 Amazon EC2에 액세스하려면 별도의 서비스 역할도 필요하며 DAX는 이 서비스 역할을 자동으로 생성합니다. 자세한 내용은 DAX에 대한 서비스 연결 역할 사용을 참조하세요.

5. 원하는 대로 설정되었으면 [Next]를 선택합니다.
6. 파라미터 그룹 - 기존 항목 선택을 선택합니다.
7. 유지 관리 기간 - 소프트웨어 업그레이드가 적용될 때 기본 설정이 없는 경우 기본 설정 없음을 선택하고, 유지 관리 기간을 예약하려는 경우 기간 지정을 선택한 후 평일, 시간(UTC) 및 다음 시간 내에 시작(단위: 시간) 옵션을 지정합니다.
8. 태그 - 새 태그 추가를 선택하고 태그 지정을 위한 키-값 쌍을 입력합니다.
9. Next(다음)를 선택합니다.

검토 및 생성 화면에서 모든 설정을 검토할 수 있습니다. 클러스터를 만들 준비가 되면 클러스터 생성을 선택합니다.

Clusters(클러스터) 화면에 DAX 클러스터가 Creating(생성 중) 상태로 나열됩니다.

> **Note**
>
> 클러스터를 생성하는 데는 몇 분 정도가 걸립니다. 클러스터가 준비되면 상태가 사용 가능으로 변경됩니다.
> 그 동안 3단계: AWS Management Console을 사용하여 보안 그룹 인바운드 규칙 구성 단원으로 이동하여 해당 지침을 따릅니다.

## 3단계: AWS Management Console을 사용하여 보안 그룹 인바운드 규칙 구성

Amazon DynamoDB Accelerator(DAX) 클러스터는 통신을 위해 TCP 포트 8111(암호화되지 않은 클러스터의 경우) 또는 9111(암호화된 클러스터의 경우)을 사용하므로 해당 포트에서 인바운드 트래픽을 승인해야 합니다. 이렇게 하면 Amazon VPC의 Amazon EC2 인스턴스가 DAX 클러스터에 액세스할 수 있습니다.

> **Note**
>
> DAX 클러스터를 `default`가 아닌 다른 보안 그룹으로 시작했다면 대신 해당 그룹에 대해 다음 절차를 수행해야 합니다.

보안 그룹 인바운드 규칙을 구성하려면

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 탐색 창에서 보안 그룹(Security Groups)을 선택합니다.
3. [default] 보안 그룹을 선택합니다. 작업 메뉴에서 인바운드 규칙 편집을 선택합니다.
4. [Add Rule]을 선택하고 다음 정보를 입력합니다.
   - 포트 범위 - 8111(클러스터가 암호화되지 않은 경우) 또는 9111(클러스터가 암호화된 경우)
   - 소스 - 사용자 지정으로 유지하고 오른쪽에 있는 검색 필드를 선택합니다. 드롭다운 메뉴가 표시됩니다. 기본 보안 그룹에 대한 식별자를 선택합니다.
5. 규칙 저장을 선택하여 변경 사항을 저장합니다.
6. 콘솔에서 이름을 업데이트하려면 이름 속성으로 이동하고 화면에 표시되는 편집을 선택합니다.

# DAX 및 DynamoDB 정합성 모델

Amazon DynamoDB Accelerator(DAX)는 캐시를 DynamoDB 테이블에 추가하는 프로세스를 간소화하기 위해 설계된 라이트-스루(write-through) 캐싱 서비스입니다. DAX는 DynamoDB와 별개로 작동하므로 DAX와 DynamoDB 모두의 일관성 모델을 파악하여 애플리케이션이 예상대로 작동하는지 확인하는 것이 중요합니다.

많은 사용 사례를 보면, 애플리케이션이 DAX를 사용하는 방법에 따라 DAX 클러스터 내의 데이터 일관성뿐만 아니라 DAX와 DynamoDB 간 데이터 일관성도 영향을 받는 것으로 나타났습니다.

주제


- DAX 클러스터 노드 간 정합성
- DAX 항목 캐시 동작
- DAX 쿼리 캐시 동작
- 강력히 정합하는 트랜잭션 읽기
- 음성 캐싱
- 쓰기 전략


## DAX 클러스터 노드 간 정합성

애플리케이션의 고가용성을 얻으려면 최소 3개 이상의 노드가 있는 DAX 클러스터를 프로비저닝한 다음, 이러한 노드들을 한 리전 내에 여러 개의 가용 영역에 배포하는 것이 좋습니다.

DAX 클러스터가 실행 중인 경우 데이터를 클러스터의 모든 노드에 복제합니다(2개 이상의 노드를 프로비저닝한 경우). DAX를 사용하여 성공적인 `UpdateItem` 작업을 수행하는 애플리케이션을 생각해 봅니다. 이 작업으로 인해 기본 노드의 항목 캐시가 새 값으로 수정되는 경우가 발생할 수 있습니다. 이 값은 클러스터의 다른 모든 노드에 복제됩니다. 이 복제 작업은 최종적으로 일관성이 있는 작업이고, 완료되는 데 보통 1초 미만이 걸립니다.

이 시나리오에서는 두 개의 클라이언트가 동일한 DAX 클러스터에서 동일한 키를 읽지만, 각 클라이언트가 액세스한 노드에 따라 서로 다른 값을 수신할 수 있습니다. 이러한 노드는 클러스터의 모든 노드에 업데이트가 완전히 복제되면 전체 일관성이 달성됩니다. (이러한 동작은 DynamoDB의 특징인 최종적 일관성과 유사합니다.)

DAX를 사용하는 애플리케이션을 구축하고 있는 경우, 데이터가 최종적 일관성을 유지해도 괜찮은 방식으로 애플리케이션이 설계되어야 합니다.

# DAX 항목 캐시 동작

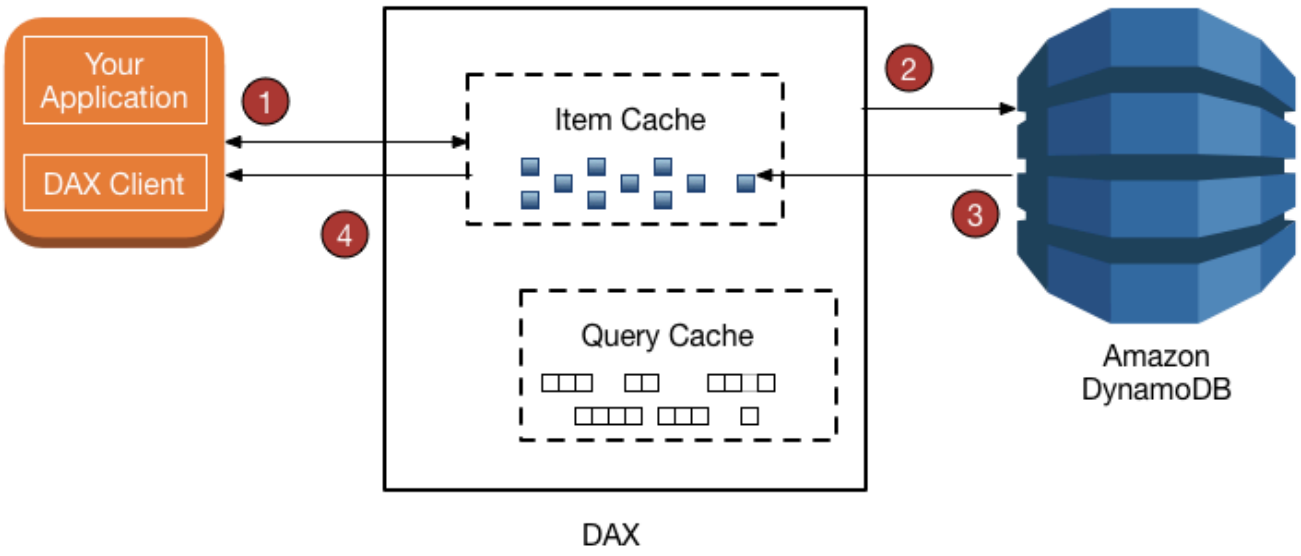
모든 DAX 클러스터에는 두 개의 개별 캐시인 항목 캐시와 쿼리 캐시가 있습니다. 자세한 내용은 DAX: 작동 방식 단원을 참조하십시오.

이 단원에서는 DAX 항목 캐시를 읽고 쓸 때 발생하는 일관성 영향에 대해 설명합니다.

## 읽기 정합성

DynamoDB를 사용하면 기본적으로 `GetItem` 작업이 최종적으로 일관된 읽기를 수행합니다. DynamoDB 클라이언트에서 `UpdateItem`을 사용한다고 가정합니다. 그 직후 동일한 항목을 읽으려고 하면 업데이트하기 전에 표시된 데이터가 그대로 표시될 수 있습니다. 이는 전체 DynamoDB 스토리지 위치에 걸쳐 전파 지연이 발생하기 때문입니다. 일관성은 일반적으로 몇 초 안에 이루어집니다. 따라서 읽기를 다시 시도하면 업데이트된 항목이 나타날 것입니다.

DAX 클라이언트에서 `GetItem`을 사용하는 경우에는 작업(이 경우 최종적으로 일관된 읽기)이 다음과 같이 수행됩니다.




1. DAX 클라이언트가 `GetItem` 요청을 실행합니다. DAX는 항목 캐시에서 요청된 항목을 읽으려고 시도합니다. 항목이 캐시에 있으면(캐시 적중), DAX가 이를 애플리케이션에 반환합니다.
2. 항목이 없으면(캐시 누락) DAX가 DynamoDB에 대해 최종적 일관된 `GetItem` 작업을 수행합니다.
3. DynamoDB는 요청된 항목을 반환하고, DAX는 이를 항목 캐시에 저장합니다.

4. DAX가 항목을 애플리케이션에 반환합니다.
5. (표시되지 않음) DAX 클러스터에 2개 이상의 노드가 있으면 항목이 클러스터의 다른 모든 노드에 복제됩니다.

캐시의 유지 시간(TTL) 설정 및 가장 오랫동안 사용되지 않음(LRU) 알고리즘에 따라 항목이 DAX 항목 캐시에 유지됩니다. 자세한 내용은 DAX: 작동 방식 단원을 참조하십시오.

단, 이 기간 동안에는 DAX가 DynamoDB에서 항목을 다시 읽지 않습니다. 다른 사용자가 DAX를 완전히 우회하는 방식으로 DynamoDB 클라이언트를 사용하여 항목을 업데이트하는 경우, DAX 클라이언트를 사용하는 `GetItem` 요청은 DynamoDB 클라이언트를 사용하는 동일한 `GetItem` 요청의 결과와 다릅니다. 이러한 경우 DAX 항목에 대한 TTL이 만료될 때까지 DAX 및 DynamoDB가 동일한 키에 대해 일관되지 않은 값을 갖고 있게 됩니다.

애플리케이션이 DAX를 우회하여 기본 DynamoDB 테이블에 있는 데이터를 수정하는 경우 애플리케이션은 이로 인해 발생할 수 있는 데이터 비일관성을 예상하고 허용할 수 있어야 합니다.

> **Note**
>
> `GetItem` 외에도 DAX 클라이언트는 `BatchGetItem` 요청도 지원합니다. `BatchGetItem`은 기본적으로 하나 이상의 `GetItem` 요청을 포함하는 래퍼이므로 DAX는 이러한 요청을 각각 개별 `GetItem` 작업으로 처리합니다.

## 쓰기 정합성

DAX는 동시 기록 캐시로, DAX 항목 캐시가 기본 DynamoDB 테이블과 일관성을 유지하도록 하는 프로세스를 간소화합니다.

DAX 클라이언트는 DynamoDB와 동일한 쓰기 API 작업을 지원합니다(`PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem` 및 `TransactWriteItems`). DAX 클라이언트를 사용하여 이러한 작업을 수행하면 DAX 및 DynamoDB 모두에서 항목이 수정됩니다. DAX는 이러한 항목의 TTL 값에 상관없이 항목 캐시에 있는 항목을 업데이트합니다.

예를 들어, DAX 클라이언트에서 `GetItem` 요청을 실행하여 `ProductCatalog` 테이블의 항목을 읽는다고 가정합니다. (파티션 키는 `Id`이고 정렬 키는 없습니다.) `Id`가 `101`인 항목을 검색합니다. 해당 항목의 `QuantityOnHand` 값은 `42`입니다. DAX는 특정 TTL을 가진 항목 캐시에 해당 항목을 저장합니다. 이 예제에서는 TTL이 10분이라고 가정합니다. 3분 후, 다른 애플리케이션이 DAX 클라이언트를 사용하여 동일한 항목을 업데이트합니다. 그러면 `QuantityOnHand` 값이 이제 `41`이 됩니다.

항목이 다시 업데이트되지 않는다고 가정하면, 다음 10분 동안 동일한 항목의 후속 읽기가 수행되면 QuantityOnHand에 대해 캐시된 값(41)이 반환됩니다.

### DAX에서 쓰기를 처리하는 방식

DAX는 고성능 읽기가 필요한 애플리케이션을 위해 만들어졌습니다. 동시 기록 캐시인 DAX는 동시 기록을 DynamoDB에 동기식으로 전달한 다음 결과 업데이트를 클러스터에 있는 모든 노드의 항목 캐시에 비동기식으로 자동 복제합니다. DAX에서 자동으로 처리해주므로 사용자가 캐시 무효화 로직을 관리할 필요가 없습니다.

DAX는 PutItem, UpdateItem, DeleteItem, BatchWriteItem, TransactWriteItems 등의 쓰기 작업을 지원합니다.

PutItem, UpdateItem, DeleteItem 또는 BatchWriteItem 요청을 DAX에 전송하면 다음 작업이 수행됩니다.

- DAX가 요청을 DynamoDB에 보냅니다.
- DynamoDB가 쓰기 성공을 확인하는 응답을 DAX에 보냅니다.
- DAX가 항목 캐시에 항목을 씁니다.
- DAX가 요청자에게 성공을 반환합니다.

TransactWriteItems 요청을 DAX에 전송하면 다음 작업이 수행됩니다.

- DAX가 요청을 DynamoDB에 보냅니다.
- DynamoDB가 트랜잭션 완료를 확인하는 응답을 DAX에 보냅니다.
- DAX가 요청자에게 성공을 반환합니다.
- 백그라운드에서 DAX는 TransactWriteItems 요청의 각 항목에 대해 TransactGetItems 요청을 수행하여 항목 캐시에 항목을 저장합니다. TransactGetItems는 직렬화 가능 격리를 유지하기 위해 사용됩니다.

제한을 비롯한 어떤 이유로든 DynamoDB에 쓰기가 실패하면 항목이 DAX에 캐시되지 않습니다. 이러한 실패에 대한 예외는 요청자에게 반환됩니다. 따라서 데이터가 먼저 DynamoDB에 성공적으로 쓰여질 때까지 DAX 캐시에 데이터가 쓰여지지 않습니다.

> **Note**
>
> DAX에 쓸 때마다 항목 캐시의 상태가 바뀝니다. 그러나 해당 항목 캐시에 대한 쓰기는 쿼리 캐시에 영향을 미치지 않습니다. (DAX 항목 캐시 및 쿼리 캐시는 용도가 서로 다르며, 서로 독립적으로 작동합니다.)

## DAX 쿼리 캐시 동작

DAX는 `Query` 및 `Scan` 요청의 결과를 쿼리 캐시에 캐시합니다. 그러나 이러한 결과는 항목 캐시에 전혀 영향을 미치지 않습니다. 애플리케이션이 DAX에서 `Query` 또는 `Scan` 요청을 실행하면 결과 집합이 항목 캐시가 아닌 쿼리 캐시에 저장됩니다. 항목 캐시와 쿼리 캐시는 별도의 엔터티이므로 `Scan` 작업을 수행하여 항목 캐시를 "워밍업"할 수 없습니다.

### 쿼리-업데이트-쿼리 정합성

항목 캐시나 기본 DynamoDB 테이블 업데이트 작업은 쿼리 캐시에 저장된 결과를 무효화하거나 수정하지 않습니다.

다음 시나리오를 예로 들어보겠습니다. 애플리케이션이 파티션 키가 `DocId`이고 정렬 키가 `RevisionNumber`인 `DocumentRevisions` 테이블에서 작동 중입니다.

1. 클라이언트는 `RevisionNumber`가 5보다 크거나 같은 모든 항목의 `DocId` `101`에 대해 `Query`를 실행합니다. DAX는 결과 집합을 쿼리 캐시에 저장하고 결과 집합을 사용자에게 반환합니다.
2. 클라이언트는 `20`의 값이 `RevisionNumber`인`DocId` `101`에 대해 `PutItem` 요청을 발행합니다.
3. 클라이언트가 1단계에서 설명한 것과 동일한 `Query`(`DocId` `101` 및 `RevisionNumber`가 `5` 이상인)를 발행합니다.

이 시나리오에서는 3단계에서 발행된 `Query`의 캐시된 결과 집합이 1단계에서 캐시된 결과 집합과 동일합니다. 이는 DAX가 개별 항목에 대한 업데이트를 기반으로 `Query` 또는 `Scan` 결과 집합을 무효화하지 않기 때문입니다. 2단계의 `PutItem` 작업은 `Query`의 TTL이 만료될 때만 DAX 쿼리 캐시에 반영됩니다.

사용자는 쿼리 캐시의 TTL 값과 더불어, 애플리케이션이 쿼리 캐시와 항목 캐시 간 비일관적인 결과를 허용할 수 있는 기간을 고려해야 합니다.

## 강력히 정합하는 트랜잭션 읽기

강력히 일관된 `GetItem`, `BatchGetItem`, `Query` 또는 `Scan` 요청을 수행하려면 `ConsistentRead` 파라미터를 true로 설정합니다. DAX는 강력히 일관된 읽기 요청을 DynamoDB에 전달합니다. DynamoDB에서 응답을 받으면 DAX는 결과를 클라이언트에 반환하지만 결과를 캐시하지 않습니다. DAX는 DynamoDB와 긴밀하게 결합되어 있지 않으므로 강력히 일관된 읽기를 자체적으로 수행할 수 없습니다. 이러한 이유로 DAX에서의 후속 읽기는 최종적으로 일관된 읽기가 되어야 합니다. 또한 이후의 모든 최종적으로 일관된 읽기는 DynamoDB으로 전달되어야 합니다.

DAX는 강력히 일관된 읽기를 처리하는 것과 동일한 방법으로 `TransactGetItems` 요청을 처리합니다. DAX는 모든 `TransactGetItems` 요청을 DynamoDB에 전달합니다. DynamoDB에서 응답을 받으면 DAX는 결과를 클라이언트에 반환하지만 결과를 캐시하지 않습니다.

## 음성 캐싱

DAX는 항목 캐시 및 쿼리 캐시 모두에서 음성 캐시 엔트리를 지원합니다. 음성 캐시 엔트리는 DAX가 기본 DynamoDB 테이블에서 요청한 항목을 찾을 수 없을 때 발생합니다. DAX는 오류를 생성하는 대신 빈 결과를 캐시하고 해당 결과를 사용자에게 반환합니다.

예를 들어, 애플리케이션이 `GetItem` 요청을 DAX 클러스터에 보내지만 DAX 항목 캐시에 일치하는 항목이 없다고 가정합니다. 이렇게 되면 DAX가 기본 DynamoDB 테이블에서 해당 항목을 읽게 됩니다. 항목이 DynamoDB에 없으면, DAX가 항목 캐시에 빈 항목을 저장한 다음, 빈 항목을 애플리케이션에 반환합니다. 이제 애플리케이션에서 동일한 항목에 대해 다른 `GetItem` 요청을 보낸다고 가정합니다. DAX는 항목 캐시에서 빈 항목을 찾고 이를 애플리케이션에 즉시 반환합니다. DynamoDB는 전혀 고려하지 않습니다.

음성 캐시 항목은 항목 TTL이 만료되거나, LRU가 호출되거나, 항목이 `PutItem`, `UpdateItem` 또는 `DeleteItem`을 통해 수정될 때까지 DAX 항목 캐시에 유지됩니다.

DAX 쿼리 캐시도 유사한 방식으로 음성 캐시 결과를 처리합니다. 애플리케이션이 `Query` 또는 `Scan`을 수행하고 DAX 쿼리 캐시에 캐시된 결과가 없는 경우 DAX가 요청을 DynamoDB로 보냅니다. 결과 집합에 일치하는 항목이 없으면, DAX가 쿼리 캐시에 빈 결과 집합을 저장한 다음, 빈 결과 집합을 애플리케이션에 반환합니다. 해당 결과 집합의 TTL이 만료될 때까지 후속 `Query` 또는 `Scan` 요청은 동일한(빈) 결과 집합을 산출합니다.

## 쓰기 전략

DAX의 동시 기록 동작은 많은 애플리케이션 패턴에 적합합니다. 그러나, 라이트-스루 모델이 적합하지 않은 일부 애플리케이션 패턴도 있습니다.

대기 시간에 민감한 애플리케이션의 경우, DAX를 통해 쓰기를 수행하면 추가 네트워크 홉이 발생합니다. 따라서 DAX에 쓰기를 하는 것이 DynamoDB에 직접 쓰기를 하는 것보다 속도가 약간 느립니다. 애플리케이션이 쓰기 지연 시간에 민감한 경우 대신 DynamoDB에 직접 쓰기를 수행함으로써 지연 시간을 줄일 수 있습니다. 자세한 내용은 라이트-어라운드(Write-Around) 단원을 참조하십시오.

쓰기 집약적인 애플리케이션(대량 데이터 로딩 작업을 수행하는 애플리케이션 등)의 경우 애플리케이션에서 매우 낮은 비율의 데이터만 읽을 수 있으므로 DAX를 통해 모든 데이터를 쓰는 것이 바람직하지 않을 수 있습니다. DAX를 통해 대량의 데이터를 쓰면 LRU 알고리즘이 호출되어 새로운 항목을 읽을 수 있도록 캐시에 공간이 마련됩니다. 이는 읽기 캐시로서의 DAX의 효율성을 떨어뜨립니다.

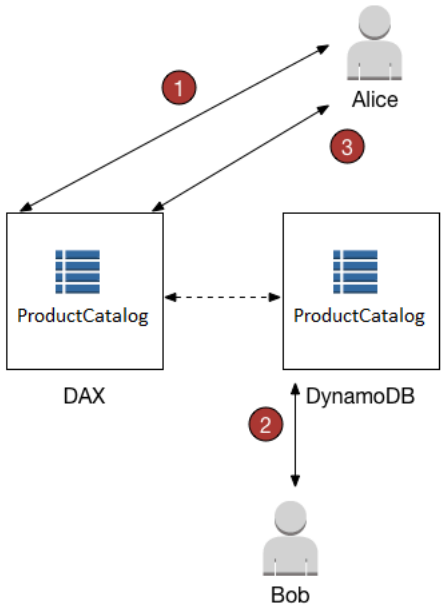
항목을 DAX에 쓰면, 새로운 항목을 수용하기 위해 항목 캐시 상태가 변경됩니다. (예를 들어, DAX는 항목 캐시에 있는 오래된 데이터를 제거하여 새로운 항목을 위한 공간을 마련해야 할 수 있습니다.) 캐시의 LRU 알고리즘 및 TTL 설정에 따라 새로운 항목이 항목 캐시에 유지됩니다. 항목이 항목 캐시에 유지되는 동안 DAX는 DynamoDB에서 항목을 다시 읽지 않습니다.

### 라이트-스루

DAX 항목 캐시는 라이트-스루 정책을 구현합니다. 자세한 내용은 DAX에서 쓰기를 처리하는 방식 단원을 참조하십시오.

항목을 쓰면, DAX가 캐시된 항목이 해당 항목(DynamoDB에 있는 경우)과 동기화되도록 합니다. 이러한 기능은 항목을 쓴 직후 다시 읽어야 하는 애플리케이션에 유용합니다. 그러나 다른 애플리케이션이 DynamoDB 테이블에 직접 쓰는 경우 DAX 항목 캐시의 항목이 더 이상 DynamoDB와 동기화 상태가 아니게 됩니다.

예를 들어, `ProductCatalog` 테이블을 사용하고 있는 두 명의 사용자(Alice와 Bob)를 가정해 보겠습니다. Alice는 DAX를 사용하여 테이블에 액세스하지만 Bob은 DAX를 우회하고 DynamoDB의 테이블에 직접 액세스합니다.




1. Alice는 `ProductCatalog` 테이블의 항목을 업데이트합니다 DAX는 요청을 DynamoDB에 전달하고 업데이트가 성공합니다. 그런 다음 DAX는 항목 캐시에 항목을 쓰고 성공한 응답을 Alice에 반환합니다. 이 시점에서 항목이 캐시에서 궁극적으로 제거될 때까지, DAX로부터 항목을 읽는 모든 사용자에게는 Alice가 업데이트한 항목이 표시됩니다.
2. 잠시 후 Bob이 Alice가 쓰기 작업을 한 것과 동일한 `ProductCatalog` 항목을 업데이트합니다. 그러나 Bob은 DynamoDB에서 직접 항목을 업데이트합니다. DAX는 DynamoDB를 통한 업데이트에 대한 응답으로 항목 캐시를 자동으로 새로 고치지 않습니다. 따라서 DAX 사용자는 Bob의 업데이트를 알 수 없습니다.
3. Alice가 DAX에서 항목을 다시 읽습니다. 항목이 항목 캐시에 있으므로 DAX가 DynamoDB 테이블에 액세스하지 않고 Alice에게 항목을 반환합니다.

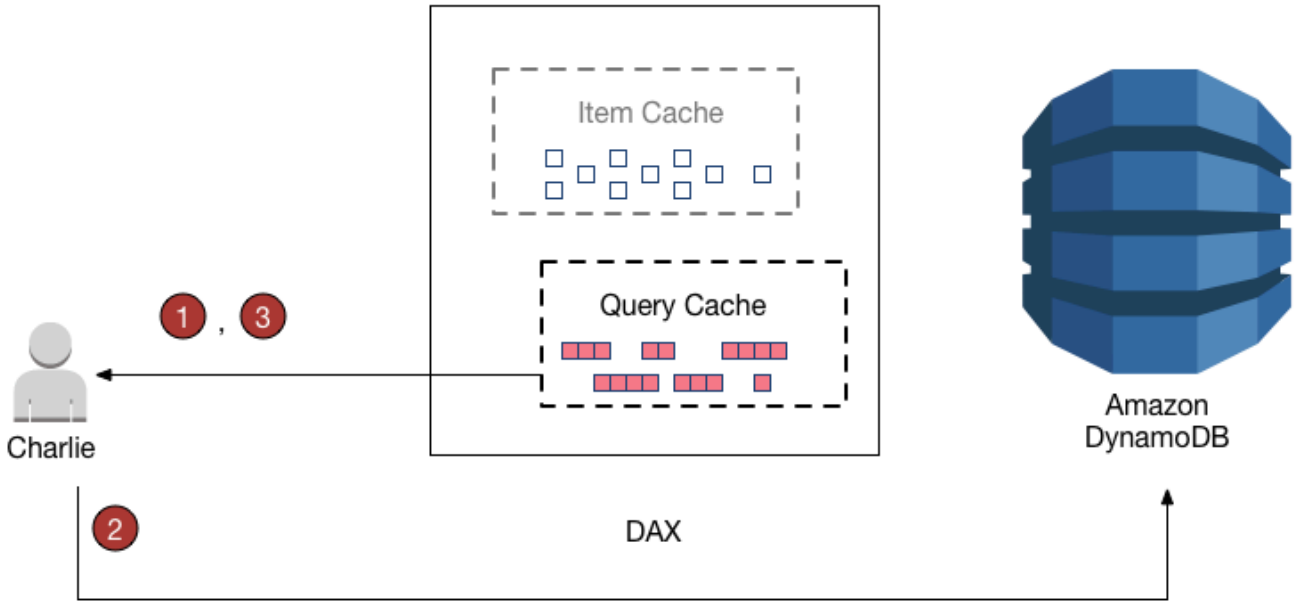
이 시나리오에서는 Alice와 Bob에게 동일한 `ProductCatalog` 항목의 다른 값이 표시됩니다. 이러한 현상은 DAX가 항목 캐시로부터 항목을 제거할 때까지, 또는 다른 사용자가 DAX를 사용하여 동일한 항목을 다시 업데이트할 때까지 계속됩니다.

## 라이트-어라운드(Write-Around)

애플리케이션에서 대량의 데이터를 써야 하는 경우(대량 데이터 로드 등), DAX를 우회하고 데이터를 DynamoDB에 직접 쓰는 것이 좋을 수 있습니다. 이러한 라이트-어라운드 전략은 쓰기 지연 시간을 줄여줍니다. 그러나 항목 캐시는 DynamoDB에서 데이터와 동기화된 상태가 유지되지 않습니다.

라이트-어라운드 전략을 사용하기로 결정했다면 애플리케이션이 DAX 클라이언트를 사용하여 데이터를 읽을 때마다 DAX가 항목 캐시를 채운다는 점을 명심하세요. 이러한 점은 가장 자주 읽는 데이터가 캐시된다는 점에서(가장 자주 쓰는 데이터와 달리) 어떤 경우에는 장점이 될 수 있습니다.

예를 들어, DAX를 사용하는 다른 테이블인 `GameScores` 테이블로 작업하려는 사용자(Charlie)를 고려해봅니다. `GameScores`의 파티션 키는 `UserId`이므로 Charlie의 모든 점수의 `UserId`는 동일합니다.




1. Charlie는 자신의 모든 점수를 검색하기 위해 DAX에 `Query`를 보냅니다. DAX은 이 쿼리가 이전에 실행된 적이 없다고 가정하고 처리를 위해 DynamoDB로 해당 쿼리를 전송합니다. 그리고 DAX 쿼리 캐시에 결과를 저장한 다음, Charlie에게 결과를 반환합니다. 결과 집합은 제거되기 전까지 쿼리 캐시에서 사용 가능한 상태로 유지됩니다.
2. 이제 Charlie가 Meteor Blasters 게임을 하여 높은 점수를 획득한다고 가정해 봅니다. Charlie가 `GameScores` 테이블의 항목을 수정하는 `UpdateItem` 요청을 DynamoDB에 보냅니다.
3. 마지막으로, Charlie는 이전에 수행한 `Query`를 다시 실행하여 `GameScores`로부터 자신의 모든 데이터를 검색하기로 합니다. Charlie는 결과에서 자신이 Meteor Blasters에서 얻은 높은 점수를 볼 수 없습니다. 쿼리 캐시에서 오는 쿼리 결과가 항목 캐시가 아니기 때문입니다. 두 캐시는 서로 독립적이므로 한 캐시가 변경되어도 다른 캐시에 영향을 주지 않습니다.

DAX는 DynamoDB의 가장 최신 데이터로 쿼리 캐시의 결과 집합을 새로 고치지 않습니다. 쿼리 캐시의 각 결과 집합은 `Query` 또는 `Scan` 작업이 수행되었던 당시를 기준으로 최신 상태입니다. 따라서

Charlie의 `Query` 결과에는 `PutItem` 작업이 반영되지 않습니다. 이러한 현상은 DAX가 쿼리 캐시에서 결과 집합을 제거할 때까지 계속됩니다.

# DynamoDB Accelerator(DAX) 클라이언트로 개발

애플리케이션에서 DAX를 사용하려면 해당 프로그래밍 언어를 위한 DAX 클라이언트를 사용하면 됩니다. DAX 클라이언트는 기존 Amazon DynamoDB 애플리케이션에 대한 중단 시간을 최소화할 수 있도록 설계되었습니다(코드를 약간만 수정하면 됨).

> **Note**
>
> 다양한 프로그래밍 언어의 DAX 클라이언트는 다음 URL에서 제공됩니다.
>
> - http://dax-sdk.s3-website-us-west-2.amazonaws.com

이 단원에서는 기본 Amazon VPC에서 Amazon EC2 인스턴스를 시작하고, 인스턴스에 연결하고, 샘플 애플리케이션을 실행하는 방법을 보여줍니다. 또한 DAX 클러스터를 활용할 수 있도록 기존 애플리케이션을 수정하는 방법에 대한 정보도 제공합니다.

주제

- 자습서: DynamoDB Accelerator(DAX)를 사용하여 샘플 애플리케이션 실행
- DAX를 사용하도록 기존 애플리케이션 수정

## 자습서: DynamoDB Accelerator(DAX)를 사용하여 샘플 애플리케이션 실행

이 자습서에서는 기본 Virtual Private Cloud(VPC)에서 Amazon EC2 인스턴스를 시작하고, 인스턴스에 연결하고, Amazon DynamoDB Accelerator(DAX)를 사용하는 샘플 애플리케이션을 실행하는 방법을 보여줍니다.

> **Note**
>
> 이 자습서를 수행하려면 기본 VPC에서 DAX 클러스터를 실행하고 있어야 합니다. DAX 클러스터를 생성하지 않았다면 DAX 클러스터 생성의 지침을 참조하세요.

주제

- 1단계: Amazon EC2 인스턴스 시작

- 2단계: 사용자 및 정책 생성
- 3단계: Amazon EC2 인스턴스 구성
- 4단계: 샘플 애플리케이션 실행


## 1단계: Amazon EC2 인스턴스 시작

Amazon DynamoDB Accelerator(DAX) 클러스터를 사용할 수 있는 경우 기본 Amazon Virtual Private Cloud(Amazon VPC)에서 Amazon EC2 인스턴스를 시작할 수 있습니다. 해당 인스턴스에서 DAX 클라이언트 소프트웨어를 설치 및 실행할 수 있습니다.

EC2 인스턴스 시작

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. Launch Instance(인스턴스 시작)를 선택하고 다음과 같이 합니다.

   1단계: Amazon 머신 이미지(AMI) 선택

   1. AMI 목록에서 Amazon Linux AMI를 찾은 후 선택을 선택합니다.

   2단계: 인스턴스 유형 선택

   1. 인스턴스 유형 목록에서 t2.micro를 선택합니다.
   2. [Next: Configure Instance Details]를 선택합니다.

   3단계: 인스턴스 세부 정보 구성

   1. 네트워크에서 기본 VPC를 선택합니다.
   2. 다음: 스토리지 추가를 선택합니다.

   4단계: 스토리지 추가

   1. Next: Add Tags(다음: 태그 추가)를 선택하여 이 단계를 건너뜁니다.

   5단계: 태그 추가

   1. [Next: Configure Security Group]을 선택하여 이 단계를 건너뜁니다.

6단계: 보안 그룹 구성

1. 기존 보안 그룹 선택을 선택합니다.
2. 보안 그룹 목록에서 기본값을 선택합니다. 이것이 VPC의 기본 보안 그룹입니다.
3. [Next: Review and Launch]를 선택합니다.

7단계: 인스턴스 시작 검토

1. 시작을 선택합니다.

3. [Select an existing key pair or create a new key pair] 창에서 다음 중 하나를 수행합니다.
   - Amazon EC2 키 페어가 없는 경우 Create a new key pair(새 키 페어 생성)를 선택하고 지침을 따릅니다. 프라이빗 키 파일(`.pem` 파일)을 다운로드하라는 메시지가 표시됩니다. 나중에 Amazon EC2 인스턴스에 로그인할 때 이 파일이 필요합니다.
   - Amazon EC2 키 페어가 이미 있으면 Select a key pair(키 페어 선택)로 이동하고 목록에서 해당 키 페어를 선택합니다. Amazon EC2 인스턴스에 로그인하려면 프라이빗 키 파일(`.pem` 파일)이 있어야 합니다.
4. 키 페어를 구성한 후 인스턴스 시작을 선택합니다.
5. 콘솔 탐색 창에서 EC2 대시보드를 선택한 후 시작한 인스턴스를 선택합니다. 하단 창의 설명 탭에서 인스턴스의 퍼블릭 DNS를 찾습니다(예: `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`). 이 퍼블릭 DNS 이름은 3단계: Amazon EC2 인스턴스 구성에서 필요하므로 기록해 둡니다.

> **Note**
>
> Amazon EC2 인스턴스가 사용 가능한 상태가 되는 데 몇 분 정도 걸립니다. 그 동안 2단계: 사용자 및 정책 생성 단원으로 이동하여 해당 지침을 따릅니다.

## 2단계: 사용자 및 정책 생성

이 단계에서는 AWS Identity and Access Management를 사용하여 Amazon DynamoDB Accelerator(DAX) 클러스터 및 DynamoDB에 대한 액세스 권한을 부여하는 정책으로 사용자를 생성합니다. 그러면 DAX 클러스터와 상호 작용하는 애플리케이션을 실행할 수 있습니다.

### AWS 계정에 등록

AWS 계정이 없는 경우 다음 절차에 따라 계정을 생성합니다.

#### AWS 계정에 가입

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   AWS 계정 루트 사용자에 가입하면 AWS 계정 루트 사용자가 만들어집니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS는 가입 절차 완료된 후 사용자에게 확인 이메일을 전송합니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

#### 관리자 액세스 권한이 있는 사용자 생성

AWS 계정에 가입하고 AWS 계정 루트 사용자에 보안 조치를 한 다음, AWS IAM Identity Center을 활성화하고 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

#### 귀하의 AWS 계정 루트 사용자 보호

1. 루트 사용자를 선택하고 AWS 계정이메일 주소를 입력하여 AWS Management Console에 계정 소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자용 가상 MFA 디바이스 활성화(콘솔)를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center 설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 기본 IAM Identity Center 디렉터리로 사용자 액세스 구성을 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자로 로그인하는 데 도움이 필요한 경우 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- AWS IAM Identity Center의 사용자 및 그룹:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.
- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.
- IAM 사용자:

- 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
- (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

JSON 정책 편집기를 사용하여 정책을 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 왼쪽의 탐색 창에서 정책을 선택합니다.

   정책을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작을 선택합니다.
3. 페이지 상단에서 정책 생성을 선택합니다.
4. 정책 편집기 섹션에서 JSON 옵션을 선택합니다.
5. JSON 정책 문서를 입력하거나 붙여 넣습니다. IAM 정책 언어에 대한 자세한 내용은 IAM JSON 정책 참조를 참조하세요.
6. 정책 검증 중에 생성되는 모든 보안 경고, 오류 또는 일반 경고를 해결하고 다음을 선택합니다.

   > **Note**
   >
   > 언제든지 시각적 편집기 옵션과 JSON 편집기 옵션을 서로 전환할 수 있습니다. 그러나 변경을 적용하거나 시각적 편집기에서 다음을 선택한 경우 IAM은 시각적 편집기에 최적화되도록 정책을 재구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 정책 재구성을 참조하세요.
7. (선택 사항) AWS Management Console에서 정책을 만들거나 편집할 때 AWS CloudFormation 템플릿에서 사용할 수 있는 JSON 또는 YAML 정책 템플릿을 생성할 수 있습니다.

   이렇게 하려면 정책 편집기에서 작업을 선택한 다음, CloudFormation 템플릿 생성을 선택합니다. AWS CloudFormation에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS Identity and Access Management 리소스 유형 참조를 참조하세요.
8. 정책에 권한 추가를 완료했으면 다음을 선택합니다.
9. 검토 및 생성 페이지에서 생성하는 정책의 정책 이름과 설명(선택 사항)을 입력합니다. 이 정책에 정의된 권한을 검토하여 정책이 부여한 권한을 확인합니다.

10. (선택 사항) 태그를 키 값 페어로 연결하여 메타데이터를 정책에 추가합니다. IAM에서 태그 사용에 대한 자세한 내용은 IAM 사용 설명서의 AWS Identity and Access Management 리소스용 태그를 참조하세요.
11. 정책 생성을 선택하고 새로운 정책을 저장합니다.

정책 문서 - 다음 문서를 복사해 붙여 넣어 JSON 정책을 생성합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "dynamodb:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

## 3단계: Amazon EC2 인스턴스 구성

Amazon EC2 인스턴스를 사용할 수 있는 경우 인스턴스에 로그인하고 사용 가능한 상태로 준비할 수 있습니다.

> **Note**
>
> 다음 단계에서는 사용자가 Linux를 실행하는 컴퓨터에서 Amazon EC2 인스턴스로 연결되어 있다고 가정합니다. 다른 연결 방법은 Amazon EC2 사용 설명서의 Linux 인스턴스에 연결을 참조하세요.

EC2 인스턴스를 구성하려면

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. ssh 명령을 사용하여 Amazon EC2 인스턴스에 로그인합니다(아래 예 참조).

   ```
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   프라이빗 키 파일(.pem 파일)과 인스턴스의 퍼블릭 DNS 이름을 지정해야 합니다. (1단계: Amazon EC2 인스턴스 시작를 참조하세요.)

   로그인 ID는 ec2-user입니다. 암호는 필요하지 않습니다.
3. EC2 인스턴스에 로그인한 후 다음과 같이 AWS 자격 증명을 구성합니다. AWS 액세스 키 ID와 비밀 키(2단계: 사용자 및 정책 생성 참조)를 입력하고 기본 리전 이름을 현재 리전으로 설정합니다. (다음 예제에서 기본 리전은 us-west-2입니다.)

   ```
   aws configure

   AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: us-west-2
   Default output format [None]:
   ```

Amazon EC2 인스턴스를 시작하고 구성한 후에는 사용 가능한 샘플 애플리케이션 중 하나를 사용하여 DAX 기능을 테스트할 수 있습니다. 자세한 내용은 4단계: 샘플 애플리케이션 실행 섹션을 참조하세요.

## 4단계: 샘플 애플리케이션 실행

Amazon DynamoDB Accelerator(DAX) 기능 테스트를 돕기 위해 Amazon EC2 인스턴스에서 사용할 수 있는 샘플 애플리케이션 중 하나를 실행할 수 있습니다.

주제

- Node.js 및 DAX
- DAX SDK for Go
- Java 및 DAX
- .NET 및 DAX
- Python 및 DAX

Node.js 및 DAX

Node.js에 대한 기본 클라이언트 구성

DAX JavaScript SDK 클라이언트를 구성할 때 다양한 파라미터를 사용자 지정하여 성능, 연결 처리 및 오류 복원력을 최적화할 수 있습니다. 다음 표에는 제한 시간 값, 재시도 메커니즘, 자격 증명 관리 및 상태 모니터링 옵션을 포함하여 클라이언트가 DAX 클러스터와 상호 작용하는 방식을 제어하는 기본 구성 설정이 요약되어 있습니다. 자세한 내용은 DynamoDBClient Operations를 참조하세요.

DAX JS SDK 클라이언트 기본값

| 파라미터 | 유형 | 설명 |
| --- | --- | --- |
| `region`<br><br>선택 사항 | `string` | DAX 클라이언트에 사용할 AWS 리전입니다(예: 'us-east-1'). 환경 변수를 통해 제공되지 않는 경우 필수 파라미터입니다. |
| `endpoint`<br><br>필수 | `string` | SDK가 연결되는 클러스터의 엔드포인트입니다.<br><br>예시:<br><br>암호화되지 않음 – dax-cluster-name.region.amazonaws.com<br><br>암호화됨 - daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com |

| 파라미터 | 유형 | 설명 |
| --- | --- | --- |
| `requestTimeout`<br><br>기본값 6,000ms | `number` | 이는 클라이언트가 DAX의 응답을 기다리는 최대 시간을 정의합니다. |
| `writeRetries`<br><br>기본값 1 | `number` | 실패한 쓰기 요청에 대한 재시도 횟수입니다. |
| `readRetries`<br><br>기본값 1 | `number` | 실패한 읽기 요청에 대한 재시도 횟수입니다. |
| `maxRetries`<br><br>기본값 1 | `number` | 실패한 요청에 대한 최대 재시도 횟수입니다.<br><br>readRetries/writeRetries가 설정된 경우 readRetries 및 writeRetries의 구성 세트가 maxRetries보다 우선합니다. |
| `connectTimeout`<br><br>기본값 10,000ms | `number` | 클러스터 노드에 대한 연결을 설정하기 위한 제한 시간(밀리초)입니다. |
| `maxConcurrentConnections`<br><br>기본값 100 | `number` | 클라이언트 인스턴스가 DAX 클러스터의 노드당 생성할 수 있는 총 동시 연결 수를 제한합니다. |

| 파라미터 | 유형 | 설명 |
| --- | --- | --- |
| maxRetryDelay<br><br>기본값 7,000ms | number | DAX 서버에서 waitForRecoveryBeforeRetrying 플래그를 true로 설정하여 복구가 필요하다고 알리면 클라이언트는 재시도 전에 일시 중지됩니다. 이러한 복구 기간 동안 maxRetryDelay 파라미터는 재시도 간의 최대 대기 시간을 결정합니다. 이 복구별 구성은 DAX 서버가 복구 모드인 경우에만 적용됩니다. 다른 모든 시나리오의 경우 재시도 동작은 재시도 횟수를 기반으로 하는 지수형 지연(writeRetries , readRetries 또는 maxRetries 파라미터에 의해 결정됨) 또는 예외 유형에 따른 즉시 재시도라는 두 가지 패턴 중 하나를 따릅니다. |
| credentials<br><br>선택 사항 | AwsCredentialIdentity \| AwsCredentialIdentityProvider | 요청을 인증하는 데 사용할 AWS 자격 증명입니다. 이는 AwsCredentialIdentity 또는 AwsCredentialIdentityProvider로 제공될 수 있습니다. 제공되지 않은 경우 AWS SDK는 기본 자격 증명 공급자 체인을 자동으로 사용합니다. 예: `{ accessKeyId: 'AKIA...', secretAccessKey: '...', sessionToken: '...' }` * @default는 AWS 자격 증명 공급자 체인을 사용합니다. |

| 파라미터 | 유형 | 설명 |
| --- | --- | --- |
| `healthCheckInterval`<br><br>기본값 5,000ms | `number` | 클러스터 상태 확인 간의 간격(밀리초)입니다. 간격이 작을수록 더 자주 확인합니다. |
| `healthCheckTimeout`<br><br>기본값 1,000ms | `number` | 상태 확인이 완료되기 위한 제한 시간(밀리초)입니다. |
| `skipHostnameVerification`<br><br>기본값 false | `boolean` | TLS 연결의 호스트 이름 확인을 건너뜁니다. 이는 암호화되지 않은 클러스터에는 영향을 주지 않습니다. 기본값은 호스트 이름 확인을 수행하는 것입니다. 이를 True로 설정하면 확인을 건너뜁니다. 이 설정을 끄면 연결 중인 클러스터를 인증할 수 없다는 점을 이해해야 합니다. |
| `unhealthyConsecutiveErrorCount`<br><br>기본값 5 | `number` | 상태 확인 간격 내에 노드 비정상 신호를 보내는 데 필요한 연속 오류 수를 설정합니다. |
| `clusterUpdateInterval`<br><br>기본값 4,000ms | `number` | 멤버십 변경에 대한 클러스터 멤버 폴링 간격을 반환합니다. |
| `clusterUpdateThreshold`<br><br>기본값 125 | `number` | 멤버십 변경에 대해 클러스터가 폴링되지 않는 임계값을 반환합니다. |

| 파라미터 | 유형 | 설명 |
| --- | --- | --- |
| credentailProvider<br><br>선택 사항 \| 기본값 Null | AwsCredentialIdentityProvider | DAX에 대한 요청을 인증하는 데 사용되는 AWS 자격 증명에 대한 사용자 정의 공급자입니다. |

DaxDocument에 대한 페이지 매김 구성

| 명칭 | 유형 | 세부 정보 |
| --- | --- | --- |
| client | DaxDocument | DaxDocument 유형의 인스턴스입니다. |
| pageSize | 숫자 | 페이지당 항목 수를 결정합니다. |
| startingToken<br><br>선택 사항 | any | 이전 응답의 LastEvaluatedKey를 후속 요청에 사용할 수 있습니다. |

페이지 매김 사용에 대한 내용은 the section called “TryDax.js” 섹션을 참조하세요.

DAX Node.js SDK V3로 마이그레이션

이 마이그레이션 가이드는 기존 DAX Node.js 애플리케이션을 전환하는 데 도움이 됩니다. 새로운 SDK에는 Node.js 18 이상이 필요하며 DynamoDB Accelerator 코드를 구성하는 방법에 몇 가지 중요한 변경 사항이 도입되었습니다. 이 가이드에서는 구문 변경, 새 가져오기 방법, 업데이트된 비동기 프로그래밍 패턴 등 주요 차이점을 안내합니다.

V2 Node.js DAX 사용량

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');

var region = "us-west-2";

AWS.config.update({
  region: region,
```

```
});

var client = new AWS.DynamoDB.DocumentClient();

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  client = new AWS.DynamoDB.DocumentClient({ service: dax });
}

// Make Get Call using Dax
var params = {
    TableName: 'TryDaxTable',
    pk: 1,
    sk: 1
}
client.get(params, function (err, data) {
    if (err) {
        console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
    } else {
        console.log(data);
    }
});
```

### V3 Node.js DAX 사용량

DAX Node.js 사용의 경우 V3 노드 버전 18 이상이 기본 버전입니다. 노드 18으로 이동하려면 다음을 사용합니다.

```
// Import AWS DAX V3
import { DaxDocument } from '@amazon-dax-sdk/lib-dax';

// Import AWS SDK V3 DynamoDBDocument ~ DocumentClient in V2
import { DynamoDBDocument } from '@aws-sdk/lib-dynamodb';
import { DynamoDBClient } from '@aws-sdk/client-dyanmodb';

// Create DynamoDBDocument
var client = DynamoDBDocument.from(new DynamoDB({region: 'us-west-2'});
```

```
// Override DynamoDBDocument Client with DaxDocument
if (process.argv.length > 2) {
  client = new DaxDocument({
    endpoints: [process.argv[2]],
    region: 'us-west-2',
  });
}

var params = {
    TableName: 'TryDaxTable',
    pk: 1,
    sk: 1
}
// Dax Shifted it's API Calls to await/promise
try {
  const results = await client.get(params);
  console.log(results);
} catch (err) {
  console.error(err)
}
```

Node.js v3.x용 DAX SDK는 Node.js v3.x용 AWS SDK와 호환됩니다. Node.js v3.x용 DAX SDK는 집계된 클라이언트 사용을 지원합니다. DAX는 기본 클라이언트 만들기를 지원하지 않습니다. 지원되지 않는 기능에 대한 자세한 내용은 the section called “AWS SDK V3와 동등하지 않은 기능” 섹션을 참조하세요.

다음 단계에 따라 Amazon EC2 인스턴스에서 Node.js 샘플 애플리케이션을 실행합니다.

DAX에 대한 Node.js 샘플을 실행하려면

1. 다음과 같이 Amazon EC2 인스턴스에서 Node.js를 설치합니다.

   a. 노드 버전 관리자(nvm)를 설치합니다.

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
   ```

   b. nvm을 사용하여 Node.js를 설치합니다.

   ```
   nvm install 18
   ```

   c. nvm을 사용하여 노드 18 사용

```
nvm use 18
```

d. Node.js가 설치되었고 올바르게 실행되는지 테스트합니다.

```
node -e "console.log('Running Node.js ' + process.version)"
```

다음 메시지가 표시되어야 합니다.

```
Running Node.js v18.x.x
```

2. 노드 패키지 관리자(npm)를 사용하여 DaxDocument Node.js 클라이언트를 설치합니다.

```
npm install @amazon-dax-sdk/lib-dax
```

TryDax 샘플 코드

DynamoDB Accelerator(DAX)의 성능 이점을 평가하려면 다음 단계에 따라 표준 DynamoDB와 DAX 클러스터 간의 읽기 작업 시간을 비교하는 샘플 테스트를 실행합니다.

1. 작업 영역을 설정하고 `lib-dax`를 종속 항목으로 설치한 후 the section called “TryDax.js”를 프로젝트에 복사합니다.
2. DAX 클러스터에 대해 프로그램을 실행합니다. DAX 클러스터의 엔드포인트를 정의하려면 다음 중 하나를 선택합니다.

   - Using the DynamoDB console(DynamoDB 콘솔 사용) - DAX 클러스터를 선택합니다. 다음 예제와 같이 클러스터 엔드포인트가 콘솔에 표시됩니다.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - AWS CLI 사용 - 다음 명령을 입력합니다.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     다음 예제와 같이 클러스터 엔드포인트가 출력에 표시됩니다.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     ```

```
}
```

3. 클러스터 엔드포인트를 명령줄 파라미터로 지정하여 프로그램을 실행합니다.

```
node TryDax.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

다음과 유사한 출력 화면이 표시되어야 합니다.

```
Attempting to create table; please wait...
Successfully created table. Table status: ACTIVE
Writing data to the table...
Writing 20 items for partition key:  1
Writing 20 items for partition key:  2
Writing 20 items for partition key:  3
...
Running GetItem Test
        Total time: 153555.10 µs - Avg time: 383.89 µs
        Total time: 44679.96 µs - Avg time: 111.70 µs
        Total time: 36885.86 µs - Avg time: 92.21 µs
        Total time: 32467.25 µs - Avg time: 81.17 µs
        Total time: 32202.60 µs - Avg time: 80.51 µs
Running Query Test
        Total time: 14869.25 µs - Avg time: 2973.85 µs
        Total time: 3036.31 µs - Avg time: 607.26 µs
        Total time: 2468.92 µs - Avg time: 493.78 µs
        Total time: 2062.53 µs - Avg time: 412.51 µs
        Total time: 2178.22 µs - Avg time: 435.64 µs
Running Scan Test
        Total time: 2395.88 µs - Avg time: 479.18 µs
        Total time: 2207.16 µs - Avg time: 441.43 µs
        Total time: 2443.14 µs - Avg time: 488.63 µs
        Total time: 2038.24 µs - Avg time: 407.65 µs
        Total time: 1972.17 µs - Avg time: 394.43 µs
Running Pagination Test
Scan Pagination
[
  { pk: 1, sk: 1, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 2, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 3, someData: 'XXXXXXXXXX' }
]
[
  { pk: 1, sk: 4, someData: 'XXXXXXXXXX' },
```

```
  { pk: 1, sk: 5, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 6, someData: 'XXXXXXXXXX' }
]
...
Query Pagination
[
  { pk: 1, sk: 1, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 2, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 3, someData: 'XXXXXXXXXX' }
]
[
  { pk: 1, sk: 4, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 5, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 6, someData: 'XXXXXXXXXX' }
]
...
Attempting to delete table; please wait...
Successfully deleted table.
```

타이밍 정보를 기록해 둡니다. `GetItem`, `Query`, `Scan` 테스트에 필요한 마이크로초의 수입니다.

4. 이 경우 DAX 클러스터에 대해 프로그램을 실행했습니다. 이제 프로그램을 다시 실행하는데, 이번에는 DynamoDB 클러스터에 대해 실행합니다.
5. 이제 프로그램을 다시 실행합니다. 이번에는 클러스터 엔드포인트 URL을 명령줄 파라미터로 지정합니다.

```
node TryDax.js
```

출력을 확인하여 타이밍 정보를 기록해 둡니다. `GetItem`, `Query` 및 `Scan`에 대한 경과 시간은 DAX가 DynamoDB보다 현저히 적어야 합니다.

### AWS SDK V3와 동등하지 않은 기능

- 기본 클라이언트 - Dax Node.js V3는 기본 클라이언트를 지원하지 않습니다.

```
const dynamoDBClient = new DynamoDBClient({ region: 'us-west-2' });
const regularParams = {
    TableName: 'TryDaxTable',
    Key: {
        pk: 1,
        sk: 1
```

```
    }
};
// The DynamoDB client supports the send operation.
const dynamoResult = await dynamoDBClient.send(new GetCommand(regularParams));

// However, the DaxDocument client does not support the send operation.
const daxClient = new DaxDocument({
    endpoints: ['your-dax-endpoint'],
    region: 'us-west-2',
});

const params = {
    TableName: 'TryDaxTable',
    Key: {
        pk: 1,
        sk: 1
    }
};

// This will throw an error - send operation is not supported for DAX. Please refer
 to documentation.
const result = await daxClient.send(new GetCommand(params));
console.log(result);
```

- 미들웨어 스택 - Dax Node.js V3는 미들웨어 함수 사용을 지원하지 않습니다.

```
const dynamoDBClient = new DynamoDBClient({ region: 'us-west-2' });
// The DynamoDB client supports the middlewareStack.
dynamoDBClient.middlewareStack.add(
  (next, context) =>> async (args) => {
    console.log("Before operation:", args);
    const result = await next(args);
    console.log("After operation:", result);
    return result;
  },
  {
    step: "initialize", // or "build", "finalizeRequest", "deserialize"
    name: "loggingMiddleware",
  }
);

// However, the DaxDocument client does not support the middlewareStack.
```

```
const daxClient = new DaxDocument({
    endpoints: ['your-dax-endpoint'],
    region: 'us-west-2',
});

// This will throw an error - custom middleware and middlewareStacks are not
 supported for DAX. Please refer to documentation.
daxClient.middlewareStack.add(
  (next, context) => async (args) => {
    console.log("Before operation:", args);
    const result = await next(args);
    console.log("After operation:", result);
    return result;
  },
  {
    step: "initialize", // or "build", "finalizeRequest", "deserialize"
    name: "loggingMiddleware",
  }
);
```

### TryDax.js

```
import { DynamoDB, waitUntilTableExists, waitUntilTableNotExists } from "@aws-sdk/
client-dynamodb";
import { DaxDocument, daxPaginateScan, daxPaginateQuery } from "@amazon-dax-sdk/lib-
dax";
import { DynamoDBDocument, paginateQuery, paginateScan } from "@aws-sdk/lib-dynamodb";

const region = "us-east-1";
const tableName = "TryDaxTable";

// Determine the client (DynamoDB or DAX)
let client = DynamoDBDocument.from(new DynamoDB({ region }));
if (process.argv.length > 2) {
  client = new DaxDocument({ region, endpoint: process.argv[2] });
}

// Function to create table
async function createTable() {
  const dynamodb = new DynamoDB({ region });
  const params = {
```

```
    TableName: tableName,
    KeySchema: [
      { AttributeName: "pk", KeyType: "HASH" },
      { AttributeName: "sk", KeyType: "RANGE" },
    ],
    AttributeDefinitions: [
      { AttributeName: "pk", AttributeType: "N" },
      { AttributeName: "sk", AttributeType: "N" },
    ],
    ProvisionedThroughput: { ReadCapacityUnits: 25, WriteCapacityUnits: 25 },
  };

  try {
    console.log("Attempting to create table; please wait...");
    await dynamodb.createTable(params);
    await waitUntilTableExists({ client: dynamodb, maxWaitTime: 30 }, { TableName:
 tableName });
    console.log("Successfully created table. Table status: ACTIVE");
  } catch (err) {
    console.error("Error in table creation:", err);
  }
}

// Function to insert data
async function writeData() {
  console.log("Writing data to the table...");
  const someData = "X".repeat(10);
  for (let ipk = 1; ipk <= 20; ipk++) {
    console.log("Writing 20 items for partition key: ", ipk)
    for (let isk = 1; isk <= 20; isk++) {
      try {
        await client.put({ TableName: tableName, Item: { pk: ipk, sk: isk,
 someData } });
      } catch (err) {
        console.error("Error inserting data:", err);
      }
    }
  }
}

// Function to test GetItem
async function getItemTest() {
  console.log("Running GetItem Test");
  for (let i = 0; i < 5; i++) {
```

```
    const startTime = performance.now();
    const promises = [];
    for (let ipk = 1; ipk <= 20; ipk++) {
      for (let isk = 1; isk <= 20; isk++) {
        promises.push(client.get({ TableName: tableName, Key: { pk: ipk, sk: isk } }));
      }
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 400).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Query
async function queryTest() {
  console.log("Running Query Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
    for (let pk = 1; pk <= 5; pk++) {
      const params = {
        TableName: tableName,
        KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
        ExpressionAttributeValues: { ":pkval": pk, ":skval1": 1, ":skval2": 2 },
      };
      promises.push(client.query(params));
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 5).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Scan
async function scanTest() {
  console.log("Running Scan Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
```

```
    const promises = [];
    for (let pk = 1; pk <= 5; pk++) {
      const params = {
        TableName: tableName,
        FilterExpression: "pk = :pkval and sk between :skval1 and :skval2",
        ExpressionAttributeValues: { ":pkval": pk, ":skval1": 1, ":skval2": 2 },
      };
      promises.push(client.scan(params));
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 5).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Pagination
async function paginationTest() {
  console.log("Running Pagination Test");
  console.log("Scan Pagination");
  const scanParams = { TableName: tableName };
  const paginator = process.argv.length > 2 ? daxPaginateScan : paginateScan;
  for await (const page of paginator({ client, pageSize: 3 }, scanParams)) {
    console.log(page.Items);
  }

  console.log("Query Pagination");
  const queryParams = {
    TableName: tableName,
    KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
    ExpressionAttributeValues: { ":pkval": 1, ":skval1": 1, ":skval2": 10 },
  };
  const queryPaginator = process.argv.length > 2 ? daxPaginateQuery : paginateQuery;
  for await (const page of queryPaginator({ client, pageSize: 3 }, queryParams)) {
    console.log(page.Items);
  }
}

// Function to delete the table
async function deleteTable() {
  const dynamodb = new DynamoDB({ region });
  console.log("Attempting to delete table; please wait...")
```

```
  try {
    await dynamodb.deleteTable({ TableName: tableName });
    await waitUntilTableNotExists({ client: dynamodb, maxWaitTime: 30 }, { TableName:
 tableName });
    console.log("Successfully deleted table.");
  } catch (err) {
    console.error("Error deleting table:", err);
  }
}

// Execute functions sequentially
(async function () {
  await createTable();
  await writeData();
  await getItemTest();
  await queryTest();
  await scanTest();
  await paginationTest();
  await deleteTable();
})();
```

DAX SDK for Go

다음 절차에 따라 Amazon EC2 인스턴스에서 Amazon DynamoDB Accelerator(DAX) SDK for Go 샘플 애플리케이션을 실행합니다.

DAX용 SDK for Go 샘플을 실행하려면

1. Amazon EC2 인스턴스에 SDK for Go를 설치합니다.

   a. Go 프로그래밍 언어(Golang)를 설치합니다.

   ```
   sudo yum install -y golang
   ```

   b. Golang가 설치되었고 올바르게 실행되는지 테스트합니다.

   ```
   go version
   ```

   다음과 같은 메시지가 나타납니다.

   ```
   go version go1.23.4 linux/amd64
   ```

2. 샘플 Golang 애플리케이션을 설치합니다.

```
go get github.com/aws-samples/sample-aws-dax-go-v2
```

3. 다음 Golang 프로그램을 실행합니다. 첫 프로그램은 `TryDaxGoTable`이라는 DynamoDB 테이블을 생성합니다. 두 번째 프로그램은 해당 테이블에 데이터를 씁니다.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command create-table
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command put-item
```

4. 다음 Golang 프로그램을 실행합니다.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command get-item
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command query
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command scan
```

`GetItem`, `Query` 및 `Scan` 테스트에 필요한 시간 정보(밀리초)를 기록해 둡니다.

5. 전 단계에서 DynamoDB 엔드포인트에 대해 프로그램을 실행했습니다. 이제 프로그램을 다시 실행하되, 이번에는 `GetItem`, `Query` 및 `Scan` 작업을 DAX 클러스터에서 처리합니다.

   DAX 클러스터의 엔드포인트를 정의하려면 다음 중 하나를 선택합니다.

   - Using the DynamoDB console(DynamoDB 콘솔 사용) - DAX 클러스터를 선택합니다. 다음 예제와 같이 클러스터 엔드포인트가 콘솔에 표시됩니다.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - AWS CLI 사용 - 다음 명령을 입력합니다.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

다음 예제와 같이 클러스터 엔드포인트가 출력에 표시됩니다.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

이제 프로그램을 다시 실행합니다. 이번에는 클러스터 엔드포인트를 명령줄 파라미터로 지정합니다.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command get-item -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command query -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-query -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-batch-get -endpoint my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com:8111
```

나머지 출력을 확인하여 시간 정보를 기록해 둡니다. `GetItem`, `Query`, `Scan`에 대한 경과 시간은 DAX가 DynamoDB보다 현저히 적어야 합니다.

6. 다음 Golang 프로그램을 실행하여 `TryDaxGoTable`을 삭제합니다.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command delete-table
```

AWS SDK for Go SDK V2와 동등하지 않은 기능

미들웨어 스택 - DAX Go V2는 APIoptions를 통한 미들웨어 스택 사용을 지원하지 않습니다. 자세한 내용은 Customizing the AWS SDK for Go v2 Client Requests with Middleware를 참조하세요.

예시:

```
// Custom middleware implementation
type customSerializeMiddleware struct{}
// ID returns the identifier for the middleware
func (m *customSerializeMiddleware) ID() string {
    return "CustomMiddleware"
}
// HandleSerialize implements the serialize middleware handler
func (m *customSerializeMiddleware) HandleSerialize(
    ctx context.Context,
    in middleware.SerializeInput,
    next middleware.SerializeHandler,
) (
    out middleware.SerializeOutput,
    metadata middleware.Metadata,
    err error,
) {
    // Add your custom logic here before the request is serialized
    fmt.Printf("Executing custom middleware for request: %v\n", in)
    // Call the next handler in the middleware chain
    return next.HandleSerialize(ctx, in)
}

func executeGetItem(ctx context.Context) error {
    client, err := initItemClient(ctx)
    if err != nil {
        os.Stderr.WriteString(fmt.Sprintf("failed to initialize client: %v\n", err))
        return err
```

```
    }

    st := time.Now()
    for c := 0; c < iterations; c++ {
        for i := 0; i < pkMax; i++ {
            for j := 0; j < skMax; j++ {
                // Create key using attributevalue.Marshal for type safety
                pk, err := attributevalue.Marshal(fmt.Sprintf("%s_%d", keyPrefix, i))
                if err != nil {
                    return fmt.Errorf("error marshaling pk: %v", err)
                }
                sk, err := attributevalue.Marshal(fmt.Sprintf("%d", j))
                if err != nil {
                    return fmt.Errorf("error marshaling sk: %v", err)
                }
                key := map[string]types.AttributeValue{
                    "pk": pk,
                    "sk": sk,
                }
                in := &dynamodb.GetItemInput{
                    TableName: aws.String(table),
                    Key:       key,
                }

                // Custom middleware option
                customMiddleware := func(o *dynamodb.Options) {
                    o.APIOptions = append(o.APIOptions, func(stack *middleware.Stack)
error {
                        // Add custom middleware to the stack
                        return stack.Serialize.Add(&customSerializeMiddleware{},
middleware.After)
                    })
                }

                // Apply options to the GetItem call
                out, err := client.GetItem(ctx, in, customMiddleware)
                if err != nil {
                    return err
                }
                writeVerbose(out)
            }
        }
    }
    d := time.Since(st)
```

```
    os.Stdout.WriteString(fmt.Sprintf("Total Time: %v, Avg Time: %v\n", d, d/
iterations))
    return nil
}
```

출력:

```
failed to execute command: custom middleware through APIOptions is not supported in DAX
 client
exit status 1
```

Go에 대한 기본 클라이언트 구성

이 가이드에서는 DAX 클라이언트의 성능, 연결 관리 및 로깅 동작을 미세 조정할 수 있는 구성 옵션을 안내합니다. 기본 설정과 이를 사용자 지정하는 방법을 이해하면 Go 애플리케이션과 DAX의 상호 작용을 최적화할 수 있습니다.

이 섹션의 내용

- DAX Go SDK 클라이언트 기본값
- 클라이언트 만들기

DAX Go SDK 클라이언트 기본값

| 파라미터 | 유형 | 설명 |
| --- | --- | --- |
| `Region`<br>필수 | `string` | DAX 클라이언트에 사용할 AWS 리전입니다(예: 'us-east-1'). 환경을 통해 제공되지 않는 경우 필수 파라미터입니다. |
| `HostPorts`<br>필수 | `[] string` | SDK가 연결하는 DAX 클러스터 엔드포인트 목록입니다.<br>예시:<br>암호화되지 않음 - dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com |

| 파라미터 | 유형 | 설명 |
| --- | --- | --- |
| | | 암호화됨 - daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com |
| `MaxPendingConnectionsPerHost`<br><br>기본값 10 | `number` | 동시 연결 시도 횟수입니다. (연결은 동시에 설정 중일 수 있습니다.) |
| `ClusterUpdateThreshold`<br><br>기본값 125 * time.Millisecond | `time.Duration` | 클러스터 새로 고침 사이에 경과해야 하는 최소 시간입니다. |
| `ClusterUpdateInterval`<br><br>기본값 4 * time.Second | `time.Duration` | 클라이언트가 DAX 클러스터 정보를 자동으로 새로 고치는 간격입니다. |
| `IdleConnectionsReapDelay`<br><br>기본값 30 * time.Second | `time.Duration` | 클라이언트가 DAX 클라이언트에서 유휴 연결을 닫는 간격입니다. |
| `ClientHealthCheckInterval`<br><br>기본값 5 * time.Second | `time.Duration` | 클라이언트가 DAX 클러스터 엔드포인트에서 상태 확인을 수행하는 간격입니다. |
| `Credentials`<br><br>기본값 | `aws.CredentialsProvider` | DAX 클라이언트가 DAX 서비스에 대한 요청을 인증하는 데 사용하는 AWS 자격 증명입니다. Credentials and Credential Providers를 참조하세요. |

| 파라미터 | 유형 | 설명 |
| --- | --- | --- |
| `DialContext`<br><br>기본값 | `func` | DAX 클라이언트가 DAX 클러스터에 대한 연결을 설정하는 데 사용하는 사용자 지정 함수입니다. |
| `SkipHostnameVerification`<br><br>기본값 false | bool | TLS 연결의 호스트 이름 확인을 건너뜁니다. 이 설정은 암호화된 클러스터에만 영향을 줍니다. True로 설정하면 호스트 이름 확인이 비활성화됩니다. 확인을 비활성화하면 연결 중인 클러스터의 ID를 인증할 수 없으므로 보안 위험이 있습니다. 기본적으로 호스트 이름 확인이 활성화됩니다. |
| `RouteManagerEnabled`<br><br>기본값 false | `bool` | 이 플래그는 네트워크 오류가 발생한 경로를 제거하는 데 사용됩니다. |
| `RequestTimeout`<br><br>기본값 60 * time.Second | `time.Duration` | 이는 클라이언트가 DAX의 응답을 기다리는 최대 시간을 정의합니다.<br><br>우선순위: 컨텍스트 제한 시간 (설정된 경우) > `RequestTimmeout` (설정된 경우) > 기본값 60초 `RequestTimeout` . |
| `WriteRetries`<br><br>기본값 2 | `number` | 실패한 쓰기 요청에 대한 재시도 횟수입니다. |
| `ReadRetries`<br><br>기본값 2 | `number` | 실패한 읽기 요청에 대한 재시도 횟수입니다. |

| 파라미터 | 유형 | 설명 |
| --- | --- | --- |
| `RetryDelay`<br><br>기본값 0 | `time.Duration` | 요청 실패 시 스로틀링되지 않은 오류에 대한 재시도 지연 시간(초)입니다. |
| `Logger`<br><br>선택 사항 | `logging.Logger` | Logger는 특정 분류에서 항목을 로깅하기 위한 인터페이스입니다. |
| `LogLevel`<br><br>기본값 utils.LogOff | `number` | 이 로그 수준은 DAX에 대해서만 정의됩니다. github.com/aws/aws-dax-go-v2/tree/main/dax/utils를 사용하여 가져올 수 있습니다.<br><br>`const ( LogOff LogLevelType = 0 LogDebug LogLevelType = 1 LogDebugWithRequestRetries LogLevelType = 2 )` |

> **Note**
>
> `time.Duration`의 경우 기본 단위는 나노초입니다. 파라미터에 대한 단위를 지정하지 않으면 나노초로 간주됩니다. `daxCfg.ClusterUpdateInterval = 10`은 10나노초를 의미합니다(`daxCfg.ClusterUpdateInterval = 10 * time.Millisecond`는 10밀리초를 의미함).

클라이언트 만들기

DAX 클러스터를 만들려면:

- DAX 구성을 만든 다음 DAX 구성을 사용하여 DAX 클라이언트를 만듭니다. 이를 사용하여 필요한 경우 DAX 구성을 덮어쓸 수 있습니다.

```
import (
"github.com/aws/aws-dax-go-v2/dax/utils"
"github.com/aws/aws-dax-go-v2/dax"
)

// Non - Encrypted : 'dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com'.
// Encrypted : daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com'.

config := dax.DefaultConfig()
config.HostPorts = []string{endpoint}
config.Region = region
config.LogLevel = utils.LogDebug
daxClient, err := dax.New(config)
```

DAX Go SDK V2로 마이그레이션

이 마이그레이션 가이드는 기존 DAX Go 애플리케이션을 전환하는 데 도움이 됩니다.

V1 DAX Go SDK 사용

```
package main

import (
    "fmt"
    "os"

    "github.com/aws/aws-dax-go/dax"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"
)

func main() {
    region := "us-west-2"
    endpoint := "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
```

```
    // Create session
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String(region),
    })
    if err != nil {
        fmt.Printf("Failed to create session: %v\n", err)
        os.Exit(1)
    }

    // Configure DAX client
    cfg := dax.DefaultConfig()
    cfg.HostPorts = []string{endpoint}
    cfg.Region = region

    // Create DAX client
    daxClient, err := dax.New(cfg)
    if err != nil {
        fmt.Printf("Failed to create DAX client: %v\n", err)
        os.Exit(1)
    }
    defer daxClient.Close() // Don't forget to close the client

    // Create GetItem input
    input := &dynamodb.GetItemInput{
        TableName: aws.String("TryDaxTable"),
        Key: map[string]*dynamodb.AttributeValue{
            "pk": {
                N: aws.String("1"),
            },
            "sk": {
                N: aws.String("1"),
            },
        },
    }

    // Make the GetItem call
    result, err := daxClient.GetItem(input)
    if err != nil {
        fmt.Printf("Failed to get item: %v\n", err)
        os.Exit(1)
    }

    // Print the result
```

```
    fmt.Printf("GetItem succeeded: %+v\n", result)
}
```

### V2 DAX Go SDK 사용

```
package main

import (
    "context"
    "fmt"
    "os"

    "github.com/aws/aws-dax-go-v2/dax"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/aws"
)

func main() {
    ctx := context.Background()
    region := "us-west-2"
    endpoint := "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"

    // Create DAX config
    config := dax.DefaultConfig()
    // Specify Endpoint and Region
    config.HostPorts = []string{endpoint}
    config.Region = region
    // Enabling logging
    config.LogLevel = utils.LogDebug
    // Create DAX client
    daxClient, err := dax.New(config)
    if err != nil {
        fmt.Printf("Failed to create DAX client: %v\n", err)
        os.Exit(1)
    }
    defer daxClient.Close() // Don't forget to close the client

    // Create key using attributevalue.Marshal for type safety
    pk, err := attributevalue.Marshal(fmt.Sprintf("%s_%d", keyPrefix, i))
    if err != nil {
```

```
        return fmt.Errorf("error marshaling pk: %v", err)
    }
    sk, err := attributevalue.Marshal(fmt.Sprintf("%d", j))
    if err != nil {
        return fmt.Errorf("error marshaling sk: %v", err)
    }

    // Create GetItem input
    input := &dynamodb.GetItemInput{
        TableName: aws.String("TryDaxTable"),
        Key: map[string]types.AttributeValue{
             "pk": pk,
             "sk": sk,
        },
    }

    // Make the GetItem call
    result, err := daxClient.GetItem(ctx, input)
    if err != nil {
        fmt.Printf("Failed to get item: %v\n", err)
        os.Exit(1)
    }

    // Print the result
    fmt.Printf("GetItem succeeded: %+v\n", result)
}
```

자세한 내용은 AWS samples를 참조하세요.

Java 및 DAX

DAX SDK for Java 2.x는 AWS SDK for Java 2.x와 호환됩니다. Java 8+에 토대를 두고 있으며, 비차단형 I/O 지원을 포함합니다. AWS SDK for Java 1.x와 함께 DAX를 사용하는 방법에 대한 자세한 내용은 AWS SDK for Java 1.x와 함께 DAX 사용 섹션을 참조하세요.

클라이언트를 Maven 종속 항목으로 사용

애플리케이션에서 DAX SDK for Java용 클라이언트를 종속 항목으로 사용하려면 다음 단계를 따릅니다.

1. Apache Maven을 다운로드하고 설치합니다. 자세한 내용은 Downloading Apache Maven 및 Installing Apache Maven을 참조하세요.

2. 애플리케이션의 POM(Project Object Model) 파일에 클라이언트 Maven 종속 항목을 추가합니다. 이 예제에서는 *x.x.x*를 클라이언트의 실제 버전 번호로 바꿉니다.

```
<!--Dependency:-->
<dependencies>
    <dependency>
        <groupId>software.amazon.dax</groupId>
        <artifactId>amazon-dax-client</artifactId>
        <version>x.x.x</version>
    </dependency>
</dependencies>
```

### TryDax 샘플 코드

작업 영역을 설정하고 DAX SDK를 종속 항목으로 추가한 후 TryDax.java를 프로젝트에 복사합니다.

다음 명령을 사용하여 코드를 실행합니다.

```
java -cp classpath TryDax
```

다음과 유사한 출력 화면이 표시되어야 합니다.

```
Creating a DynamoDB client

Attempting to create table; please wait...
Successfully created table.  Table status: ACTIVE
Writing data to the table...
Writing 10 items for partition key: 1
Writing 10 items for partition key: 2
Writing 10 items for partition key: 3
...

Running GetItem and Query tests...
First iteration of each test will result in cache misses
Next iterations are cache hits

GetItem test - partition key 1-100 and sort keys 1-10
  Total time: 4390.240 ms - Avg time: 4.390 ms
  Total time: 3097.089 ms - Avg time: 3.097 ms
  Total time: 3273.463 ms - Avg time: 3.273 ms
  Total time: 3353.739 ms - Avg time: 3.354 ms
  Total time: 3533.314 ms - Avg time: 3.533 ms
```

```
Query test - partition key 1-100 and sort keys between 2 and 9
  Total time: 475.868 ms - Avg time: 4.759 ms
  Total time: 423.333 ms - Avg time: 4.233 ms
  Total time: 460.271 ms - Avg time: 4.603 ms
  Total time: 397.859 ms - Avg time: 3.979 ms
  Total time: 466.644 ms - Avg time: 4.666 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

타이밍 정보, 즉 `GetItem` 및 `Query` 테스트에 필요한 시간(밀리초)를 기록해 둡니다. 여기서는 DynamoDB 엔드포인트에 대해 프로그램을 실행했습니다. 이제 프로그램을 다시 실행하는데, 이번에는 DAX 클러스터에 대해 실행합니다.

DAX 클러스터의 엔드포인트를 결정하려면 다음 중 하나를 선택합니다.

- DynamoDB 콘솔에서 DAX 클러스터를 선택합니다. 다음 예제와 같이 클러스터 엔드포인트가 콘솔에 표시됩니다.

  ```
  dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
  ```

- AWS CLI를 사용하여 다음 명령을 입력합니다.

  ```
  aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
  ```

  다음 예제와 같이 클러스터 엔드포인트 주소, 포트, URL이 출력에 표시됩니다.

  ```
  {
      "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
      "Port": 8111,
      "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
  }
  ```

이제 프로그램을 다시 실행합니다. 이번에는 클러스터 엔드포인트 URL을 명령줄 파라미터로 지정합니다.

```
java -cp classpath TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

출력을 확인하여 타이밍 정보를 기록해 둡니다. `GetItem` 및 `Query`에 대한 경과 시간은 DAX가 DynamoDB보다 현저히 적어야 합니다.

### SDK 지표

DAX SDK for Java 2.x를 사용하면 애플리케이션의 서비스 클라이언트에 대한 지표를 수집하고 Amazon CloudWatch에서 출력을 분석할 수 있습니다. 자세한 내용은 SDK 지표 활성화를 참조하세요.

> **Note**
>
> DAX SDK for Java는 `ApiCallSuccessful` 및 `ApiCallDuration` 지표만 수집합니다.

### TryDax.java

```
import java.util.Map;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.dax.ClusterDaxAsyncClient;
import software.amazon.dax.Configuration;

public class TryDax {
    public static void main(String[] args) throws Exception {
        DynamoDbAsyncClient ddbClient = DynamoDbAsyncClient.builder()
                .build();

        DynamoDbAsyncClient daxClient = null;
        if (args.length >= 1) {
            daxClient = ClusterDaxAsyncClient.builder()
                    .overrideConfiguration(Configuration.builder()
                            .url(args[0]) // e.g. dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com
                            .build())
                    .build();
```

```
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        createTable(tableName, ddbClient);

        System.out.println("Populating table...");
        writeData(tableName, ddbClient, 100, 10);

        DynamoDbAsyncClient testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
        getItemTest(tableName, testClient, 100, 10, 5);

        // Query
        queryTest(tableName, testClient, 100, 2, 9, 5);

        System.out.println("Deleting table...");
        deleteTable(tableName, ddbClient);
    }

    private static void createTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("Attempting to create table; please wait...");

            client.createTable(CreateTableRequest.builder()
                    .tableName(tableName)
                    .keySchema(KeySchemaElement.builder()
                            .keyType(KeyType.HASH)
                            .attributeName("pk")
                            .build(), KeySchemaElement.builder()
                            .keyType(KeyType.RANGE)
                            .attributeName("sk")
                            .build())
```

```
                    .attributeDefinitions(AttributeDefinition.builder()
                            .attributeName("pk")
                            .attributeType(ScalarAttributeType.N)
                            .build(), AttributeDefinition.builder()
                            .attributeName("sk")
                            .attributeType(ScalarAttributeType.N)
                            .build())
                    .billingMode(BillingMode.PAY_PER_REQUEST)
                    .build()).get();
            client.waiter().waitUntilTableExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully created table.");

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    private static void deleteTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            client.deleteTable(DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            client.waiter().waitUntilTableNotExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
        }
    }

    private static void writeData(String tableName, DynamoDbAsyncClient client, int
pkmax, int skmax) {
        System.out.println("Writing data to the table...");

        int stringSize = 1000;
        StringBuilder sb = new StringBuilder(stringSize);
        for (int i = 0; i < stringSize; i++) {
```

```
            sb.append('X');
        }
        String someData = sb.toString();

        try {
            for (int ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
                for (int isk = 1; isk <= skmax; isk++) {
                    client.putItem(PutItemRequest.builder()
                            .tableName(tableName)
                            .item(Map.of("pk", attr(ipk), "sk", attr(isk), "someData",
attr(someData)))
                            .build()).get();
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
    }

    private static AttributeValue attr(int n) {
        return AttributeValue.builder().n(String.valueOf(n)).build();
    }

    private static AttributeValue attr(String s) {
        return AttributeValue.builder().s(s).build();
    }

    private static void getItemTest(String tableName, DynamoDbAsyncClient client, int
pk, int sk, int iterations) {
        long startTime, endTime;
        System.out.println("GetItem test - partition key 1-" + pk + " and sort keys 1-"
+ sk);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            try {
                for (int ipk = 1; ipk <= pk; ipk++) {
                    for (int isk = 1; isk <= sk; isk++) {
                        client.getItem(GetItemRequest.builder()
                                .tableName(tableName)
                                .key(Map.of("pk", attr(ipk), "sk", attr(isk)))
```

```
                                .build()).get();
                    }
                }
            } catch (Exception e) {
                System.err.println("Unable to get item:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk * sk);
        }
    }

    private static void queryTest(String tableName, DynamoDbAsyncClient client, int pk,
int sk1, int sk2, int iterations) {
        long startTime, endTime;
        System.out.println("Query test - partition key 1-" + pk + " and sort keys
between " + sk1 + " and " + sk2);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            for (int ipk = 1; ipk <= pk; ipk++) {
                try {
                    // Pagination API for Query.
                    client.queryPaginator(QueryRequest.builder()
                            .tableName(tableName)
                            .keyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
                            .expressionAttributeValues(Map.of(":pkval", attr(ipk),
":skval1", attr(sk1), ":skval2", attr(sk2)))
                            .build()).items().subscribe((item) -> {
                    }).get();
                } catch (Exception e) {
                    System.err.println("Unable to query table:");
                    e.printStackTrace();
                }
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk);
        }
    }

    private static void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
(1000000.0));
```

```
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
 1000000.0));
    }
}
```

.NET 및 DAX

Amazon EC2 인스턴스에서 .NET 샘플을 실행하려면 다음 단계를 따릅니다.

> **Note**
>
> 이 자습서에서는 .NET 6 SDK를 사용하지만 .NET Core SDK도 사용할 수 있습니다. 기본 Amazon VPC에서 프로그램을 실행하여 Amazon DynamoDB Accelerator(DAX) 클러스터에 액세스하는 방법을 보여줍니다. 원하는 경우 AWS Toolkit for Visual Studio를 사용하여 .NET 애플리케이션을 작성하고 VPC에 배포할 수 있습니다.
> 자세한 내용은 AWS Elastic Beanstalk 개발자 설명서에서 AWS Toolkit for Visual Studio를 사용한 .NET 내 Elastic Beanstalk 애플리케이션 생성 및 배포를 참조하세요.

DAX에 대한 .NET 샘플을 실행하려면

1. Microsoft 다운로드 페이지로 이동하여 최신 버전의 .NET 6(또는 .NET Core) SDK for Linux를 다운로드합니다. 다운로드된 파일은 `dotnet-sdk-N.N.N-linux-x64.tar.gz`입니다.
2. SDK 파일의 압축을 풉니다.

   ```
   mkdir dotnet
   tar zxvf dotnet-sdk-N.N.N-linux-x64.tar.gz -C dotnet
   ```

   *N.N.N*을 .NET SDK의 실제 버전 번호로 바꿉니다.(예: `6.0.100`).
3. 설치를 확인합니다.

   ```
   alias dotnet=$HOME/dotnet/dotnet
   dotnet --version
   ```

   .NET SDK의 버전 번호를 인쇄합니다.

   > **Note**
   >
   > 버전 번호 대신 다음 오류를 수신할 수 있습니다.

error: libunwind.so.8: cannot open shared object file: No such file or directory
오류를 해결하려면 `libunwind` 패키지를 설치합니다.

```
sudo yum install -y libunwind
```

이를 완료한 후에는 오류 없이 `dotnet --version` 명령을 실행할 수 있습니다.

4. 새로운 .NET 프로젝트 생성.

```
dotnet new console -o myApp
```

일회성 설정을 수행하는 데 몇 분이 걸릴 수 있습니다. 완료되고 나면 샘플 프로젝트를 실행합니다.

```
dotnet run --project myApp
```

다음과 같은 메시지를 받게 됩니다. `Hello World!`

5. `myApp/myApp.csproj` 파일에는 프로젝트 관련 메타데이터가 포함되어 있습니다. 애플리케이션에서 DAX 클라이언트를 사용하려면 파일을 다음과 같이 수정해야 합니다.

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net6.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="AWSSDK.DAX.Client" Version="*" />
    </ItemGroup>
</Project>
```

6. 샘플 프로그램 소스 코드(`.zip` 파일)를 다운로드합니다.

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

다운로드가 완료되면 소스 파일의 압축을 풉니다.

```
unzip TryDax.zip
```

7. 이제 한 번에 하나씩 샘플 프로그램을 실행합니다. 각 프로그램에서 내용을 myApp/Program.cs로 복사한 다음 MyApp 프로젝트를 실행합니다.

   다음 .NET 프로그램을 실행합니다. 첫 프로그램은 TryDaxTable이라는 DynamoDB 테이블을 생성합니다. 두 번째 프로그램은 해당 테이블에 데이터를 씁니다.

   ```
   cp TryDax/dotNet/01-CreateTable.cs myApp/Program.cs
   dotnet run --project myApp

   cp TryDax/dotNet/02-Write-Data.cs myApp/Program.cs
   dotnet run --project myApp
   ```

8. 다음으로 일부 프로그램을 실행하여 DAX 클러스터에서 GetItem, Query 및 Scan 작업을 수행합니다. DAX 클러스터의 엔드포인트를 정의하려면 다음 중 하나를 선택합니다.

   - Using the DynamoDB console(DynamoDB 콘솔 사용) - DAX 클러스터를 선택합니다. 다음 예제와 같이 클러스터 엔드포인트가 콘솔에 표시됩니다.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - AWS CLI 사용 - 다음 명령을 입력합니다.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     다음 예제와 같이 클러스터 엔드포인트가 출력에 표시됩니다.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   다음 프로그램을 실행하여 클러스터 엔드포인트를 명령줄 파라미터로 지정합니다. (샘플 엔드포인트를 실제 DAX 클러스터 엔드포인트로 바꿉니다.)

   ```
   cp TryDax/dotNet/03-GetItem-Test.cs myApp/Program.cs
   dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com

   cp TryDax/dotNet/04-Query-Test.cs myApp/Program.cs
   ```

```
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com

cp TryDax/dotNet/05-Scan-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
```

`GetItem`, `Query` 및 `Scan` 테스트에 필요한 시간 정보(밀리초)를 기록해 둡니다.

9. 다음 .NET 프로그램을 실행하여 `TryDaxTable`를 삭제합니다.

```
cp TryDax/dotNet/06-DeleteTable.cs myApp/Program.cs
dotnet run --project myApp
```

이러한 프로그램에 대한 자세한 내용은 다음 단원을 참조하세요.

- 01-CreateTable.cs
- 02-Write-Data.cs
- 03-GetItem-Test.cs
- 04-Query-Test.cs
- 05-Scan-Test.cs
- 06-DeleteTable.cs

### 01-CreateTable.cs

`01-CreateTable.cs` 프로그램은 테이블(`TryDaxTable`)을 만듭니다. 이 단원의 나머지 .NET 프로그램은 이 테이블을 사용합니다.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
```

```
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            var request = new CreateTableRequest()
            {
                TableName = tableName,
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement{ AttributeName = "pk",KeyType = "HASH"},
                    new KeySchemaElement{ AttributeName = "sk",KeyType = "RANGE"}
                },
                AttributeDefinitions = new List<AttributeDefinition>() {
                    new AttributeDefinition{ AttributeName = "pk",AttributeType = "N"},
                    new AttributeDefinition{ AttributeName = "sk",AttributeType  = "N"}
                },
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 10
                }
            };

            var response = await client.CreateTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

### 02-Write-Data.cs

`02-Write-Data.cs` 프로그램은 테스트 데이터를 `TryDaxTable`에 씁니다.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
```

```
namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            string someData = new string('X', 1000);
            var pkmax = 10;
            var skmax = 10;

            for (var ipk = 1; ipk <= pkmax; ipk++)
            {
                Console.WriteLine($"Writing {skmax} items for partition key: {ipk}");
                for (var isk = 1; isk <= skmax; isk++)
                {
                    var request = new PutItemRequest()
                    {
                        TableName = tableName,
                        Item = new Dictionary<string, AttributeValue>()
                       {
                            { "pk", new AttributeValue{N = ipk.ToString() } },
                            { "sk", new AttributeValue{N = isk.ToString() } },
                            { "someData", new AttributeValue{S = someData } }
                       }
                    };

                    var response = await client.PutItemAsync(request);
                }
            }

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

### 03-GetItem-Test.cs

`03-GetItem-Test.cs` 프로그램은 `GetItem`에서 `TryDaxTable` 작업을 수행합니다.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var pk = 1;
            var sk = 10;
            var iterations = 5;

            var startTime = System.DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                for (var ipk = 1; ipk <= pk; ipk++)
                {
                    for (var isk = 1; isk <= sk; isk++)
                    {
                        var request = new GetItemRequest()
                        {
                            TableName = tableName,
```

```
                            Key = new Dictionary<string, AttributeValue>() {
                            {"pk", new AttributeValue {N = ipk.ToString()} },
                            {"sk", new AttributeValue {N = isk.ToString() } }
                        }
                        };
                        var response = await client.GetItemAsync(request);
                        Console.WriteLine($"GetItem succeeded for pk: {ipk},sk:
 {isk}");
                    }
                }
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

04-Query-Test.cs

04-Query-Test.cs 프로그램은 Query에서 TryDaxTable 작업을 수행합니다.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");
```

```
            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var pk = 5;
            var sk1 = 2;
            var sk2 = 9;
            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                var request = new QueryRequest()
                {
                    TableName = tableName,
                    KeyConditionExpression = "pk = :pkval and sk between :skval1
and :skval2",
                    ExpressionAttributeValues = new Dictionary<string,
AttributeValue>() {
                            {":pkval", new AttributeValue {N = pk.ToString()} },
                            {":skval1", new AttributeValue {N = sk1.ToString()} },
                            {":skval2", new AttributeValue {N = sk2.ToString()} }
                    }
                };
                var response = await client.QueryAsync(request);
                Console.WriteLine($"{i}: Query succeeded");

            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
```

```
    }
}
```

### 05-Scan-Test.cs

05-Scan-Test.cs 프로그램은 Scan에서 TryDaxTable 작업을 수행합니다.

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                var request = new ScanRequest()
                {
                    TableName = tableName
                };
                var response = await client.ScanAsync(request);
                Console.WriteLine($"{i}: Scan succeeded");
```

```
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

### 06-DeleteTable.cs

`06-DeleteTable.cs` 프로그램은 `TryDaxTable`을 삭제합니다. 테스트를 완료한 후 이 프로그램을 실행합니다.

```
using System;
using System.Threading.Tasks;
using Amazon.DynamoDBv2.Model;
using Amazon.DynamoDBv2;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            var request = new DeleteTableRequest()
            {
                TableName = tableName
            };

            var response = await client.DeleteTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
```

```
            Console.ReadLine();
        }
    }
}
```

Python 및 DAX

다음 절차를 따라 Amazon EC2 인스턴스에서 Python 샘플 애플리케이션을 실행합니다.

DAX에 대한 Python 샘플을 실행하려면

1. pip 유틸리티를 사용하여 DAX Python 클라이언트를 설치합니다.

   ```
   pip install amazon-dax-client
   ```

2. 샘플 프로그램 소스 코드(.zip 파일)를 다운로드합니다.

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   다운로드가 완료되면 소스 파일의 압축을 풉니다.

   ```
   unzip TryDax.zip
   ```

3. 다음 Python 프로그램을 실행합니다. 첫 프로그램은 TryDaxTable이라는 Amazon DynamoDB 테이블을 생성합니다. 두 번째 프로그램은 해당 테이블에 데이터를 씁니다.

   ```
   python 01-create-table.py
   python 02-write-data.py
   ```

4. 다음 Python 프로그램을 실행합니다.

   ```
   python 03-getitem-test.py
   python 04-query-test.py
   python 05-scan-test.py
   ```

   GetItem, Query 및 Scan 테스트에 필요한 시간 정보(밀리초)를 기록해 둡니다.

5. 전 단계에서 DynamoDB 엔드포인트에 대해 프로그램을 실행했습니다. 이제 프로그램을 다시 실행하되, 이번에는 GetItem, Query 및 Scan 작업을 DAX 클러스터에서 처리합니다.

DAX 클러스터의 엔드포인트를 정의하려면 다음 중 하나를 선택합니다.

- Using the DynamoDB console(DynamoDB 콘솔 사용) - DAX 클러스터를 선택합니다. 다음 예제와 같이 클러스터 엔드포인트가 콘솔에 표시됩니다.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- AWS CLI 사용 - 다음 명령을 입력합니다.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

다음 예제와 같이 클러스터 엔드포인트가 출력에 표시됩니다.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

프로그램을 다시 실행합니다. 이번에는 클러스터 엔드포인트를 명령줄 파라미터로 지정합니다.

```
python 03-getitem-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
python 04-query-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
python 05-scan-test.py dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

나머지 출력을 확인하여 시간 정보를 기록해 둡니다. `GetItem`, `Query`, `Scan`에 대한 경과 시간은 DAX가 DynamoDB보다 현저히 적어야 합니다.

6. 다음 Python 프로그램을 실행하여 `TryDaxTable`을 삭제합니다.

```
python 06-delete-table.py
```

이러한 프로그램에 대한 자세한 내용은 다음 단원을 참조하세요.

- 01-create-table.py
- 02-write-data.py

- 03-getitem-test.py
- 04-query-test.py
- 05-scan-test.py
- 06-delete-table.py

### 01-create-table.py

`01-create-table.py` 프로그램은 테이블(`TryDaxTable`)을 만듭니다. 이 단원의 나머지 Python 프로그램은 이 테이블을 사용합니다.

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
        "KeySchema": [
            {"AttributeName": "partition_key", "KeyType": "HASH"},
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "BillingMode": "PAY_PER_REQUEST",
    }
    table = dyn_resource.create_table(**params)
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table
```

```
if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

02-write-data.py

`02-write-data.py` 프로그램은 테스트 데이터를 `TryDaxTable`에 씁니다.

```
import boto3


def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
            table.put_item(
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")


if __name__ == "__main__":
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
```

```
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

03-getitem-test.py

`03-getitem-test.py` 프로그램은 `GetItem`에서 `TryDaxTable` 작업을 수행합니다. 이 예제는 eu-west-1 리전용입니다.

```
import argparse
import sys
import time
import amazondax
import boto3


def get_item_test(key_count, iterations, dyn_resource=None):
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource('dynamodb')

    table = dyn_resource.Table('TryDaxTable')
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(Key={
                    'partition_key': partition_key,
                    'sort_key': sort_key
                })
                print('.', end='')
                sys.stdout.flush()
    print()
```

```
    end = time.perf_counter()
    return start, end


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        'endpoint_url', nargs='?',
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.")
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url,
 region_name='eu-west-1') as dax:
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax)
    else:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the Boto3 client.")
        test_start, test_end = get_item_test(
            test_key_count, test_iterations)
    print(f"Total time: {test_end - test_start:.4f} sec. Average time: "
          f"{(test_end - test_start)/ test_iterations}.")
```

### 04-query-test.py

`04-query-test.py` 프로그램은 `Query`에서 `TryDaxTable` 작업을 수행합니다.

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key


def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
```

```
    Queries the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this value.
    :param sort_keys: The range of sort key values for the query. The query returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
    ).between(*sort_keys)

    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()

    test_partition_key = 5
    test_sort_keys = (2, 9)
    test_iterations = 100
    if args.endpoint_url:
```

```
        print(f"Querying the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations, dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )

    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

05-scan-test.py

05-scan-test.py 프로그램은 Scan에서 TryDaxTable 작업을 수행합니다.

```
import argparse
import time
import sys
import amazondax
import boto3


def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
```

```
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()

    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

06-delete-table.py

`06-delete-table.py` 프로그램은 `TryDaxTable`을 삭제합니다. Amazon DynamoDB Accelerator(DAX) 기능 테스트를 완료한 후 이 프로그램을 실행합니다.

```
import boto3


def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.
```

```
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()

    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
    delete_dax_table()
    print("Table deleted!")
```

## DAX를 사용하도록 기존 애플리케이션 수정

Amazon DynamoDB를 사용하는 Java 애플리케이션이 이미 있는 경우 DynamoDB Accelerator(DAX) 클러스터에 액세스할 수 있도록 애플리케이션을 수정해야 합니다. DAX Java 클라이언트는 AWS SDK for Java 2.x에 포함되어 있는 DynamoDB 하위 수준 클라이언트와 매우 유사하므로 전체 애플리케이션을 다시 작성할 필요가 없습니다. 자세한 내용은 DynamoDB의 항목 작업을 참조하세요.

> **Note**
>
> 이 예제에서는 AWS SDK for Java 2.x를 사용합니다. 레거시 SDK for Java 1.x 버전의 경우 DAX를 사용하도록 기존 SDK for Java 1.x 애플리케이션 수정을 참조하세요.

프로그램을 수정하기 위해 DynamoDB 클라이언트를 DAX 클라이언트로 바꿉니다.

```
Region region = Region.US_EAST_1;

// Create an asynchronous DynamoDB client
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
                .region(region)
                .build();

// Create an asynchronous DAX client
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
                .overrideConfiguration(Configuration.builder()
```

```
                    .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
                    .region(region)
                    .addMetricPublisher(cloudWatchMetricsPub) // optionally enable SDK
 metric collection
                    .build())
                .build();
```

또한 AWS SDK for Java 2.x에 포함된 상위 수준 라이브러리를 사용하여 DynamoDB 클라이언트를 DAX 클라이언트로 바꿀 수 있습니다.

```
Region region = Region.US_EAST_1;
DynamoDbAsyncClient dax = ClusterDaxAsyncClient.builder()
        .overrideConfiguration(Configuration.builder()
            .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
            .region(region)
            .build())
        .build();

DynamoDbEnhancedAsyncClient enhancedClient = DynamoDbEnhancedAsyncClient.builder()
        .dynamoDbClient(dax)
        .build();
```

자세한 내용은 DynamoDB 테이블의 항목 매핑 단원을 참조하세요.

# DAX 클러스터 관리

이 단원에서는 Amazon DynamoDB Accelerator(DAX) 클러스터에 대한 몇 가지 일반 관리 작업에 대해 설명합니다.

주제

- DAX 클러스터 관리를 위한 IAM 권한
- DAX 클러스터 크기 조정
- DAX 클러스터 설정 사용자 지정
- TTL 설정 구성
- DAX에 대한 태그 지정 지원
- AWS CloudTrail 통합
- DAX 클러스터 삭제

# DAX 클러스터 관리를 위한 IAM 권한

AWS Management Console 또는 AWS Command Line Interface(AWS CLI)를 사용하여 DAX 클러스터를 관리할 경우 사용자가 수행할 수 있는 작업의 범위를 제한하는 것이 좋습니다. 이렇게 하면 최소한의 권한 원칙을 따르면서 위험을 최소화할 수 있습니다.

다음 설명은 DAX 관리 API에 대한 액세스 제어에 초점을 맞춥니다. 자세한 내용은 Amazon DynamoDB API 참조의 Amazon DynamoDB Accelerator를 참조하세요.

> **Note**
>
> AWS Identity and Access Management(IAM) 권한 관리에 대한 자세한 내용은 다음을 참조하세요.
>
> - IAM 및 DAX 클러스터 생성: DAX 클러스터 생성.
> - IAM 및 DAX 데이터 영역 작업: DAX 액세스 제어.

DAX 관리 API의 경우 특정 리소스로 API 작업의 범위를 지정할 수 없습니다. `Resource` 요소를 `"*"`로 설정해야 합니다. 이는 `GetItem`, `Query`, `Scan` 등의 DAX 데이터 영역 API 작업과 다릅니다. 데이터 영역 작업은 DAX 클라이언트를 통해 노출되며, 그러한 작업은 특정 리소스로 범위를 지정할 수 있습니다.

다음과 같은 IAM 정책 문서를 예로 들어 보겠습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
```

```
}
```

이 정책의 의도는 DAXCluster01 클러스터에 대해서만 DAX 관리 API 호출을 허용하는 것이라고 간주합니다.

이제 사용자가 다음 AWS CLI 명령을 발행한다고 가정해 보겠습니다.

```
aws dax describe-clusters
```

이 명령은 권한 없음 예외로 실패합니다. 기본 DescribeClusters API 호출을 특정 클러스터로 범위를 지정할 수 없기 때문입니다. 이 정책은 구문상으로는 유효하지만 Resource 요소를 "*"로 설정해야 하기 때문에 실패합니다. 하지만 사용자가 DAX 데이터 영역 호출(예: GetItem 또는 Query)을 DAXCluster01로 보내는 프로그램을 실행할 경우 이 호출은 성공합니다. DAX 데이터 영역 API의 범위를 특정 리소스(이 경우 DAXCluster01)로 지정할 수 있기 때문입니다.

DAX 관리 API와 DAX 데이터 영역 API를 모두 포괄하는 단일 IAM 정책을 작성하려면 정책 문서에 두 가지 별개의 명령문을 포함하는 것이 좋습니다. 이러한 명령문 중 하나는 DAX 데이터 영역 API를 다뤄야 하고, 다른 명령문은 DAX 관리 API를 다뤄야 합니다.

다음 예제 정책은 이를 보여 줍니다. DAXDataAPIs 구문은 DAXCluster01 리소스로 범위가 지정되지만 DAXManagementAPIs의 리소스는 "*"가 되어야 합니다. 각 구문에 나온 작업은 예시용입니다. 애플리케이션에서 필요한 만큼 리소스를 사용자 지정할 수 있습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DAXDataAPIs",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem"
            ],
```

```
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]},
            {
            "Sid": "DAXManagementAPIs",
            "Action": [
                "dax:CreateParameterGroup",
                "dax:CreateSubnetGroup",
                "dax:DecreaseReplicationFactor",
                "dax:DeleteCluster",
                "dax:DeleteParameterGroup",
                "dax:DeleteSubnetGroup",
                "dax:DescribeClusters",
                "dax:DescribeDefaultParameters",
                "dax:DescribeEvents",
                "dax:DescribeParameterGroups",
                "dax:DescribeParameters",
                "dax:DescribeSubnetGroups",
                "dax:IncreaseReplicationFactor",
                "dax:ListTags",
                "dax:RebootNode",
                "dax:TagResource",
                "dax:UntagResource",
                "dax:UpdateCluster",
                "dax:UpdateParameterGroup",
                "dax:UpdateSubnetGroup"
                ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

## DAX 클러스터 크기 조정

DAX 클러스터 크기를 조정할 때 2가지 옵션을 사용할 수 있습니다. 첫 번째 옵션은 읽기 전용 복제본을 클러스터에 추가하는 수평적 조정입니다. 두 번째 옵션은 서로 다른 노드 유형을 선택하는 수직적 조정입니다. 애플리케이션에 적합한 클러스터 크기 및 노드 유형을 선택하는 방법에 대한 조언은 DAX 클러스터 크기 조정 안내서 단원을 참조하세요.

## 수평 크기 조정

수평적 조정을 사용하면 클러스터에 더 많은 읽기 전용 복제본을 추가하여 읽기 작업의 처리량을 개선할 수 있습니다. 단일 DAX 클러스터는 최대 10개의 읽기 전용 복제본을 지원하며, 클러스터가 실행 중인 동안 복제본을 추가하거나 제거할 수 있습니다.

새 노드를 추가할 때는 피어 노드의 캐시 데이터를 동기화해야 합니다. 따라서 추가 시간은 캐시 크기와 애플리케이션 워크로드에 따라 달라집니다. 예상되는 트래픽 최고치를 충족하려면 클러스터 규모를 미리 조정하는 것이 좋습니다. 적절한 크기 조정 지침과 모니터링 권장 사항에 대한 자세한 내용은 DAX 클러스터 크기 조정 안내서 섹션을 참조하세요.

다음은 노드 수를 늘리거나 줄이는 방법을 보여 주는 AWS CLI 예입니다. `--new-replication-factor` 인수는 클러스터의 총 노수 수를 지정합니다. 노드 중 하나는 기본 노드이고, 나머지 노드들은 읽기 전용 복제본입니다.

```
aws dax increase-replication-factor \
    --cluster-name MyNewCluster  \
    --new-replication-factor 5
```

```
aws dax decrease-replication-factor \
    --cluster-name MyNewCluster  \
    --new-replication-factor 3
```

> **Note**
>
> 복제 인수를 수정하면 클러스터 상태가 `modifying`으로 변경됩니다. 수정이 완료되면 상태가 `available`로 변경됩니다.

## 세로 크기 조정

데이터 작업 집합이 큰 경우 대형 노드 유형을 사용하면 애플리케이션에 도움이 될 수 있습니다. 대형 노드는 클러스터가 메모리에 더 많은 데이터를 저장하도록 해주어 캐시 누락을 감소시키고 애플리케이션의 전체 애플리케이션 성능을 개선할 수 있습니다. (DAX 클러스터에 있는 모든 노드는 유형이 동일해야 합니다.)

DAX 클러스터가 쓰기 작업 속도가 높거나 캐시 누락이 있는 경우 애플리케이션은 더 큰 노드 유형을 사용하는 것이 좋습니다. 쓰기 작업 및 캐시 누락은 클러스터의 기본 노드에서 리소스를 사용합니다.

따라서 더 큰 노드 유형을 사용하면 기본 노드의 성능이 향상되어 이러한 유형의 작업에 대한 처리량이 높아질 수 있습니다.

실행 중인 DAX 클러스터에서는 노드 유형을 수정할 수 없습니다. 대신, 원하는 노드 유형으로 새 클러스터를 생성해야 합니다. 지원되는 노드 유형의 전체 목록은 노드 단원을 참조하세요.

AWS Management Console, AWS CloudFormation, AWS CLI, AWS SDK 등을 사용하여 새 DAX 클러스터를 생성할 수 있습니다. (AWS CLI의 경우 `--node-type` 파라미터를 사용하여 노드 유형을 지정하세요.)

## DAX 클러스터 설정 사용자 지정

DAX 클러스터를 생성하는 경우 다음 기본 설정이 사용됩니다.

- 유지 시간(TTL)을 5분으로 설정하여 자동 캐시 제거 활성화
- 가용 영역에 대한 기본 설정 없음
- 유지 관리 기간에 대한 기본 설정 없음
- 알림 비활성화됨

새로운 클러스터의 경우 생성 시에 설정을 사용자 지정할 수 있습니다. AWS Management Console에서 사용자 지정하려면 기본 설정 사용을 지우고 다음 설정을 수정합니다.

- 네트워크 및 보안 - 현재 AWS 리전 내에 있는 다른 가용 영역에서 개별 DAX 클러스터 노드의 실행을 허용합니다. 기본 설정 없음를 선택하는 경우 노드가 AZ 사이에 자동으로 배분됩니다.
- 파라미터 그룹 - 클러스터의 모든 노드에 적용되는 명명된 파라미터 집합입니다. 파라미터 그룹을 사용하면 TTL 동작을 지정할 수 있습니다. 언제든지 파라미터 그룹(기본 파라미터 그룹 `default.dax.1.0` 제외) 내에서 지정된 파라미터 값을 변경할 수 있습니다.
- 유지 관리 기간 - 소프트웨어 업그레이드 및 패치가 클러스터의 노드에 적용되는 기간(주 단위)입니다. 유지 관리 기간의 시작 날짜, 시작 시간 및 기간을 선택할 수 있습니다. 기본 설정 없음을 선택하면 리전별로 8시간 블록 시간 중에서 임의로 유지 관리 기간이 선택됩니다. 자세한 내용은 유지보수 윈도우 섹션을 참조하세요.

> **Note**
>
> 파라미터 그룹 및 유지 관리 기간도 실행 중인 클러스터에서 언제든지 변경할 수도 있습니다.

유지 관리 이벤트가 발생하면 DAX가 Amazon Simple Notification Service(Amazon SNS)를 사용하여 알릴 수 있습니다. 알림을 구성하려면 [Topic for SNS notification] 선택기에서 옵션을 선택합니다. 새로운 Amazon SNS 주제를 생성하거나 기존 주제를 사용할 수 있습니다.

Amazon SNS 주제 설정 및 구독에 대한 자세한 내용은 Amazon Simple Notification Service 개발자 가이드의 Amazon SNS 시작하기를 참조하세요.

## TTL 설정 구성

DAX는 DynamoDB에서 읽어 들인 데이터를 위해 다음과 같은 두 개의 캐시를 유지합니다.

- 항목 캐시 - `GetItem` 또는 `BatchGetItem`을 사용하여 검색된 항목용입니다.
- 쿼리 캐시 - `Query` 또는 `Scan`을 사용하여 검색된 결과 집합용입니다.

자세한 내용은 항목 캐시 및 쿼리 캐시 단원을 참조하세요.

각 캐시의 기본 TTL은 5분입니다. 다른 TTL 설정을 사용하려면 사용자 지정 파라미터 그룹을 사용하여 DAX 클러스터를 시작하면 됩니다. 이러한 작업을 콘솔에서 수행하려면 탐색 창에서 DAX | Parameter groups(DAX | 파라미터 그룹)를 선택합니다.

AWS CLI를 사용하여 이러한 작업을 수행할 수도 있습니다. 다음은 사용자 지정 파라미터 그룹을 사용하여 새로운 DAX 클러스터를 시작하는 방법을 보여 주는 예입니다. 이 예에서는 항목 캐시 TTL이 10분으로 설정되어 있으며, 쿼리 캐시 TTL은 3분으로 설정되어 있습니다.

1. 새 파라미터 그룹을 생성해야 합니다.

```
aws dax create-parameter-group \
    --parameter-group-name custom-ttl
```

2. 항목 캐시 TTL을 10분(600000밀리초)으로 설정합니다.

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=record-ttl-millis,ParameterValue=600000"
```

3. 쿼리 캐시 TTL을 3분(180000밀리초)으로 설정합니다.

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=query-ttl-millis,ParameterValue=180000"
```

4. 파라미터를 올바르게 설정했는지 확인합니다.

```
aws dax describe-parameters --parameter-group-name custom-ttl \
    --query "Parameters[*].[ParameterName,Description,ParameterValue]"
```

이제는 이러한 파라미터 그룹을 사용하여 새 DAX 클러스터를 시작할 수 있습니다.

```
aws dax create-cluster \
    --cluster-name MyNewCluster \
    --node-type dax.r3.large \
    --replication-factor 3 \
    --iam-role-arn arn:aws:iam::123456789012:role/DAXServiceRole \
    --parameter-group custom-ttl
```

> **Note**
>
> 실행 중인 DAX 인스턴스에서 사용 중인 파라미터 그룹은 수정할 수 없습니다.

## DAX에 대한 태그 지정 지원

DynamoDB를 비롯한 여러 AWS 서비스에서 리소스에 사용자 정의 이름으로 레이블을 지정하는 기능인 태그 지정을 지원합니다. DAX 클러스터에 태그를 지정하여, 동일한 태그를 가진 모든 AWS 리소스를 빠르게 식별하거나, 지정한 태그별로 AWS 청구서를 분류할 수 있습니다.

자세한 내용은 DynamoDB의 리소스에 태그 및 레이블 추가 섹션을 참조하세요.

### AWS Management Console 사용

DAX 클러스터 태그를 관리하려면

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 탐색 창의 DAX에서 클러스터를 선택합니다.
3. 사용하려는 클러스터를 선택합니다.
4. 태그 탭을 선택합니다. 여기서 태그를 추가하거나, 태그 목록을 보거나, 편집 또는 삭제할 수 있습니다.

   원하는 대로 설정이 되었으면 Apply Changes를 선택합니다.

### AWS CLI 사용

AWS CLI를 사용하여 DAX 클러스터 태그를 관리할 경우 먼저 클러스터의 Amazon 리소스 이름(ARN)을 확인해야 합니다. 다음 예는 `MyDAXCluster`라는 클러스터의 ARN을 확인하는 방법을 보여 줍니다.

```
aws dax describe-clusters \
    --cluster-name MyDAXCluster \
    --query "Clusters[*].ClusterArn"
```

출력에 `arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster`와 같은 ARN이 표시될 것입니다.

다음 예는 클러스터에 태그를 지정하는 방법을 보여 줍니다.

```
aws dax tag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tags="Key=ClusterUsage,Value=prod"
```

클러스터의 모든 태그 목록을 나열합니다.

```
aws dax list-tags \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

태그를 제거하려면 해당 키를 지정합니다.

```
aws dax untag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tag-keys ClusterUsage
```

## AWS CloudTrail 통합

DAX가 AWS CloudTrail과 통합되어 DAX 클러스터 작업을 감사할 수 있게 되었습니다. CloudTrail 로그를 사용하여 클러스터 수준에서 수행된 모든 변경을 볼 수 있습니다. 또한 노드, 서브넷 그룹 및 파라미터 그룹 같은 클러스터 구성 요소에 대한 변경을 볼 수 있습니다. 자세한 내용은 AWS CloudTrail을 사용하여 DynamoDB 작업 로깅 섹션을 참조하세요.

## DAX 클러스터 삭제

DAX 클러스터를 더 이상 사용하지 않는 경우에는 사용하지 않은 리소스에 대해 비용이 청구되지 않도록 해당 클러스터를 삭제해야 합니다.

콘솔 또는 AWS CLI를 사용하여 DAX 클러스터를 삭제할 수 있습니다. 다음은 예입니다.

```
aws dax delete-cluster --cluster-name mydaxcluster
```

# Monitoring DynamoDB Accelerator

모니터링은 Amazon DynamoDB Accelerator(DAX)와 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. 발생하는 다중 지점 실패를 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집해야 합니다.

DAX 모니터링을 시작하기 전에 다음 질문에 대한 답변을 포함하는 모니터링 계획을 작성해야 합니다

- 모니터링의 목표
- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

주제

- DynamoDB Accelerator용 모니터링 도구
- Amazon CloudWatch를 사용한 모니터링
- AWS CloudTrail을 사용하여 DAX 작업 로깅

## DynamoDB Accelerator용 모니터링 도구

AWS는 Amazon DynamoDB Accelerator(DAX)를 모니터링하는 데 사용할 수 있는 도구를 제공합니다. 이러한 도구 중에는 모니터링을 자동으로 수행하도록 구성할 수 있는 도구가 있으며, 수동 작업이 필요한 몇 가지 도구도 있습니다. 모니터링 작업은 최대한 자동화하는 것이 좋습니다.

주제

- 자동 모니터링 도구
- 수동 모니터링 도구

## 자동 모니터링 도구

다음과 같은 자동 모니터링 도구를 사용하여 DAX를 관찰하고 문제 발생 시 보고할 수 있습니다.

- Amazon CloudWatch 경보 – 지정한 기간 동안 단일 지표를 감시하고, 여러 기간에 대해 지정된 임계값과 관련하여 지표 값을 기준으로 하나 이상의 작업을 수행합니다. 이 작업은 Amazon Simple Notification Service(Amazon SNS) 주제 또는 Amazon EC2 Auto Scaling 정책에 전송되는 알림입니다. CloudWatch 경보는 특정 상태에 있다는 이유만으로는 작업을 호출하지 않습니다. 상태가 변경되고 지정한 기간 동안 유지되어야 합니다. 자세한 내용은 Amazon CloudWatch로 DynamoDB에서 지표 모니터링 단원을 참조하십시오.
- Amazon CloudWatch Logs – AWS CloudTrail 또는 기타 소스의 로그 파일을 모니터링, 저장 및 액세스합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 로그 파일 모니터링을 참조하세요.
- Amazon CloudWatch Events – 이벤트를 일치시키고 하나 이상의 대상 함수 또는 스트림으로 라우팅하여 값을 변경하거나 상태 정보를 캡처하거나 수정 작업을 수행합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch Events란 무엇인가요를 참조하세요.
- AWS CloudTrail 로그 모니터링 - 계정 간에 로그 파일을 공유하고, CloudTrail 로그 파일을 CloudWatch Logs에 전송하여 실시간으로 모니터링하며, Java에서 로그 처리 애플리케이션을 작성하고, CloudTrail에서 전송한 후 로그 파일이 변경되지 않았는지 확인합니다. 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 로그 파일 작업을 참조하세요.

## 수동 모니터링 도구

DAX 모니터링의 또 한 가지 중요한 부분은 CloudWatch 경보에 포함되지 않는 항목을 수동으로 모니터링해야 한다는 점입니다. DAX, CloudWatch, Trusted Advisor 및 기타 AWS Management Console 대시보드에서는 AWS 환경의 상태를 한눈에 볼 수 있습니다. 또한 DAX에서 로그 파일을 확인하는 것이 좋습니다.

- DAX 대시보드는 다음을 보여줍니다.
  - 서비스 상태
- CloudWatch 홈 페이지에 다음이 표시됩니다:
  - 현재 경보 및 상태
  - 경보 및 리소스 그래프
  - 서비스 상태

  또한 CloudWatch를 사용하여 다음을 수행할 수 있습니다.
  - 사용자 정의 대시보드를 생성하여 관심 있는 서비스를 모니터링

- 지표 데이터를 그래프로 작성하여 문제를 해결하고 추세 파악
- 모든 AWS 리소스 지표를 검색하고 찾아보기
- 문제에 대해 알려주는 경보 생성 및 편집

## Amazon CloudWatch를 사용한 모니터링

DAX에서 원시 데이터를 수집하여 읽기 가능하며 실시간에 가까운 지표로 처리하는 Amazon CloudWatch를 통해 DynamoDB Accelerator(DAX)를 모니터링할 수 있습니다. 이러한 통계는 2주 동안 기록됩니다. 그러면 기록 정보에 액세스하고 웹 애플리케이션 또는 서비스가 어떻게 실행되고 있는지 전체적으로 더 잘 파악할 수 있습니다. 기본적으로 DAX 지표 데이터는 CloudWatch에 자동으로 전송됩니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch란 무엇입니까?를 참조하세요.

주제

- DAX 지표 사용 방법
- DAX 지표 및 차원 보기
- DAX를 모니터링하는 CloudWatch 경보 생성
- 프로덕션 모니터링

### DAX 지표 사용 방법

DAX에서 보고하는 지표는 다양한 방법으로 분석이 기능한 정보를 제공합니다. 다음 목록은 몇 가지 일반적인 지표 사용 사례를 보여 줍니다. 모든 사용 사례를 망라한 것은 아니지만 시작하는 데 참고가 될 것입니다.

| 사용 방법 | 관련 지표 |
| --- | --- |
| 시스템 오류가 발생했는지 여부 확인 | `FaultRequestCount` 를 모니터링하여 HTTP 500(서버 오류) 코드가 발생한 요청이 있는지 확인합니다. 이는 기본 테이블의 SystemErrors 지표에서 DAX 내부 서비스 오류 또는 HTTP 500을 나타낼 수 있습니다. |
| 사용자 오류가 발생했는지 여부 확인 | `ErrorRequestCount` 를 모니터링하여 HTTP 400(클라이언트 오류) 코드가 발생한 요청이 있는지 확인합니다. 오류 개수가 증 |

| 사용 방법 | 관련 지표 |
| --- | --- |
| | 가하면 이를 조사하여 올바른 클라이언트 요청을 전송하는지 확인해야 합니다. |
| 캐시 누락이 발생했는지 여부 확인 | `ItemCacheMisses` 를 모니터링하여 캐시에서 항목을 찾지 못한 횟수를 확인하고 `QueryCacheMisses` 및 `ScanCacheMisses` 를 모니터링하여 캐시에서 쿼리 또는 스캔 결과를 찾지 못한 횟수를 확인합니다. |
| 모니터 캐시 적중률 | CloudWatch 지표 수식으로 수학 표현식을 사용하여 캐시 적중률 지표를 정의합니다.<br><br>예를 들어, 항목 캐시의 경우 표현식 m1/SUM([m1, m2])*100을 사용할 수 있습니다. 여기서 m1은 `ItemCacheHits` 지표이고 m2는 클러스터의 `ItemCacheMisses` 지표입니다. 쿼리 및 스캔 캐시의 경우 해당하는 쿼리 및 스캔 캐시 지표를 사용하여 동일한 패턴을 따를 수 있습니다. |

## DAX 지표 및 차원 보기

Amazon DynamoDB는 사용자와 상호 작용할 때 다음 지표와 차원을 Amazon CloudWatch로 전송합니다. 다음 절차에 따라 DynamoDB Accelerator(DAX)에 대한 지표를 볼 수 있습니다.

지표를 보려면(콘솔)

지표는 먼저 서비스 네임스페이스별로 그룹화된 다음, 각 네임스페이스 내에서 다양한 차원 조합별로 그룹화됩니다.

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 지표를 선택합니다.
3. DAX 네임스페이스를 선택합니다.

지표()를 보는 방법AWS CLI

- 명령 프롬프트에서 다음 명령을 사용합니다.

```
aws cloudwatch list-metrics --namespace "AWS/DAX"
```

## DAX 지표 및 차원

다음 단원에서는 DAX에서 CloudWatch로 전송하는 지표와 차원을 설명합니다.

### DAX 지표

DAX에서 사용할 수 있는 지표는 아래와 같습니다. 단, DAX는 값이 0이 아닌 경우에 한해 지표를 CloudWatch에 전송합니다.

> **Note**
>
> CloudWatch에서는 1분 간격으로 다음의 DAX 지표를 집계합니다.
>
> - `CPUUtilization`
> - `CacheMemoryUtilization`
> - `NetworkBytesIn`
> - `NetworkBytesOut`
> - `BaselineNetworkBytesInUtilization`
> - `BaselineNetworkBytesOutUtilization`
> - `NetworkPacketsIn`
> - `NetworkPacketsOut`
> - `GetItemRequestCount`
> - `BatchGetItemRequestCount`
> - `BatchWriteItemRequestCount`
> - `DeleteItemRequestCount`
> - `PutItemRequestCount`
> - `UpdateItemRequestCount`
> - `TransactWriteItemsCount`
> - `TransactGetItemsCount`
> - `ItemCacheHits`
> - `ItemCacheMisses`
> - `QueryCacheHits`
> - `QueryCacheMisses`
> - `ScanCacheHits`

- ScanCacheMisses
- TotalRequestCount
- ErrorRequestCount
- FaultRequestCount
- FailedRequestCount
- QueryRequestCount
- ScanRequestCount
- ClientConnections
- EstimatedDbSize
- EvictedSize
- CPUCreditUsage
- CPUCreditBalance
- CPUSurplusCreditBalance
- CPUSurplusCreditsCharged

Average 또는 Sum과 같은 모든 지표에 적용되지 않는 통계도 있습니다. 하지만 이 값은 모두 DAX 콘솔, CloudWatch 콘솔, AWS CLI 또는 AWS SDK(모든 지표에 대해)를 통해 사용할 수 있습니다. 다음 테이블에는 각 지표마다 적용되는 유효한 통계 목록이 있습니다.

| 지표 | 설명 |
| --- | --- |
| CPUUtilization | 노드 또는 클러스터의 CPU 사용률의 백분율입니다.<br><br>단위: Percent<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average |
| CacheMemoryUtilization | 사용 가능한 캐시 메모리 중 노드 또는 클러스터의 항목 캐시 및 쿼리 캐시에서 사용 중인 메모리의 비율입니다. 메모리 사용률이 100%에 도달하기 전에 캐시된 데이터가 제거되기 시작합니 |

| 지표 | 설명 |
| --- | --- |
| | 다(EvictedSize 지표 참조). 어느 노드에서든 CacheMemoryUtilization 이 100%에 도달하면 쓰기 요청이 제한되며 노드 유형이 더 큰 클러스터로 전환하는 것이 좋습니다.<br><br>단위: Percent<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average |
| NetworkBytesIn | 노드 또는 클러스터가 모든 네트워크 인터페이스에서 받은 바이트 수입니다.<br><br>단위: Bytes<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average |
| NetworkBytesOut | 노드 또는 클러스터가 모든 네트워크 인터페이스에서 보낸 바이트 수입니다. 이 지표는 단일 노드 또는 클러스터에서 발신 트래픽의 볼륨을 바이트 수 기준으로 식별합니다.<br><br>단위: Bytes<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average |

| 지표 | 설명 |
| --- | --- |
| BaselineNetworkBytesInUtilization | 특정 시간에 수신 트래픽에 소비된 기준 네트워크 대역폭의 백분율입니다. 참고로 50%는 수신 트래픽에 사용 가능한 네트워크 대역폭의 절반이 사용되고 있음을 의미합니다.<br><br>단위: Percent<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| BaselineNetworkBytesOutUtilization | 특정 시간에 송신 트래픽에 소비된 기준 네트워크 대역폭의 백분율입니다. 참고로 50%는 송신 트래픽에 사용 가능한 네트워크 대역폭의 절반이 사용되고 있음을 의미합니다.<br><br>단위: Percent<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| NetworkPacketsIn | 노드 또는 클러스터가 모든 네트워크 인터페이스에서 받은 패킷 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average |
| NetworkPacketsOut | 노드 또는 클러스터가 모든 네트워크 인터페이스에서 보낸 패킷 수입니다. 이 지표는 단일 노드 또는 클러스터에서 발신 트래픽의 볼륨을 패킷 수 기준으로 식별합니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average |
| GetItemRequestCount | 노드 또는 클러스터가 처리하는 GetItem 요청의 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| BatchGetItemRequestCount | 노드 또는 클러스터가 처리하는 BatchGetItem 요청의 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| BatchWriteItemRequestCount | 노드 또는 클러스터가 처리하는 BatchWriteItem 요청의 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| DeleteItemRequestCount | 노드 또는 클러스터가 처리하는 DeleteItem 요청의 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| PutItemRequestCount | 노드 또는 클러스터가 처리하는 PutItem 요청의 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| UpdateItemRequestCount | 노드 또는 클러스터가 처리하는 UpdateItem 요청의 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| TransactWriteItemsCount | 노드 또는 클러스터가 처리하는 TransactWriteItems 요청의 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| TransactGetItemsCount | 노드 또는 클러스터가 처리하는 TransactGetItems 요청의 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| ItemCacheHits | 항목이 노드 또는 클러스터에 의해 캐시에서 반환된 횟수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| ItemCacheMisses | 항목이 노드 또는 클러스터 캐시에 없어서 DynamoDB에서 검색해야 했던 횟수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| QueryCacheHits | 쿼리 결과가 노드 또는 클러스터 캐시에서 반환된 횟수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| QueryCacheMisses | 쿼리 결과가 노드 또는 클러스터 캐시에 없어서 DynamoDB에서 검색해야 했던 횟수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| ScanCacheHits | 스캔 결과가 노드 또는 클러스터 캐시에서 반환된 횟수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| ScanCacheMisses | 스캔 결과가 노드 또는 클러스터 캐시에 없어서 DynamoDB에서 검색해야 했던 횟수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| TotalRequestCount | 노드 또는 클러스터가 처리하는 총 요청의 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| ErrorRequestCount | 노드 또는 클러스터에서 사용자 오류를 보고한 총 요청 수입니다. 노드 또는 클러스터에서 제한한 요청이 포함됩니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| ThrottledRequestCount | 노드 또는 클러스터에서 제한한 총 요청의 수입니다. DynamoDB에서 제한한 요청은 DynamoDB Metrics를 사용하여 모니터링할 수 있습니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| FaultRequestCount | 노드 또는 클러스터에서 내부 오류를 보고한 총 요청 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| FailedRequestCount | 노드 또는 클러스터에서 오류를 보고한 총 요청 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| QueryRequestCount | 노드 또는 클러스터가 처리하는 쿼리 요청의 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| ScanRequestCount | 노드 또는 클러스터가 처리하는 스캔 요청의 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| ClientConnections | 클라이언트가 노드 또는 클러스터에 연결한 동시 연결 수입니다.<br><br>단위: Count<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| EstimatedDbSize | 노드 또는 클러스터에 의해 항목 캐시 및 쿼리 캐시에 캐시된 데이터 양(근사치)입니다.<br><br>단위: Bytes<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average |

| 지표 | 설명 |
| --- | --- |
| EvictedSize | 새로 요청된 데이터를 위한 공간을 마련하기 위해 노드 또는 클러스터가 제거한 데이터 양입니다. 누락률이 상승하고 이 지표도 증가하면 아마도 작업 집합이 증가한 것입니다. 더 큰 노드 유형이 있는 클러스터로 전환하는 것을 고려해야 합니다.<br><br>단위: Bytes<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• Sum |
| CPUCreditUsage | CPU 사용률을 위해 노드에서 소비되는 CPU 크레딧의 수입니다. CPU 크레딧 하나는 1분 동안 100%의 사용률로 실행되는 vCPU 1개 또는 이와 동등한 vCPU, 사용률 및 시간의 조합과 동일합니다(예를 들어 2분 동안 50%의 사용률로 실행되는 vCPU 1개 또는 2분 동안 25%의 사용률로 실행되는 vCPU 2개).<br><br>CPU 크레딧 지표는 5분 간격으로만 제공됩니다. 5분 이상의 시간을 지정할 경우 Average 대신 Sum 통계를 사용하세요.<br><br>단위: Credits (vCPU-minutes)<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| CPUCreditBalance | 시작 이후 노드가 누적한 획득 CPU 크레딧 수입니다.<br><br>크레딧은 획득 이후에 크레딧 밸런스에 누적되고, 소비 시 크레딧 밸런스에서 소멸됩니다. 크레딧 밸런스는 최대 한도(DAX 노드 크기에 따라 결정)가 있습니다. 한도에 도달하면 새로 획득한 크레딧이 모두 삭제됩니다.<br><br>CPUCreditBalance 의 크레딧은 노드가 기준 CPU 사용률 이상으로 늘리는 데 소비할 수 있습니다.<br><br>단위: Credits (vCPU-minutes)<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

| 지표 | 설명 |
| --- | --- |
| CPUSurplusCreditBalance | CPUCreditBalance 값이 0일 때 DAX 노드에서 소비된 잉여 크레딧의 수 입니다.<br><br>획득한 CPU 크레딧에 따라 CPUSurplusCreditBalance 값이 청산됩니다. 잉여 크레딧의 수가 노드가 24시간 동안 획득할 수 있는 최대 크레딧 수를 초과한 경우 최대 값 이상으로 소비된 잉여 크레딧은 추가 요금으로 부과됩니다.<br><br>단위: Credits (vCPU-minutes)<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |
| CPUSurplusCreditsCharged | 획득한 CPU 크레딧으로 청산되지 않는 소비 잉여 크레딧의 수로, 추가 요금으로 부과됩니다.<br><br>소비한 잉여 크레딧이 노드가 24시간 동안 획득할 수 있는 최대 크레딧 수를 초과하는 경우 소비한 잉여 크레딧에 요금이 부과됩니다. 해당 시간이 끝나거나 노드가 종료될 때 최대 값 이상으로 소비한 잉여 크레딧에 요금이 부과됩니다.<br><br>단위: Credits (vCPU-minutes)<br><br>유효한 통계:<br><br>• Minimum<br>• Maximum<br>• Average<br>• SampleCount<br>• Sum |

> **Note**
>
> `CPUCreditUsage`, `CPUCreditBalance`, `CPUSurplusCreditBalance` 및 `CPUSurplusCreditsCharged` 지표는 T3 노드에만 사용할 수 있습니다.

DAX 지표의 차원

DAX의 지표는 계정, 클러스터 ID 또는 클러스터 ID 및 노드 ID 조합의 값으로 한정됩니다. CloudWatch 콘솔을 사용하면 다음 표의 어떤 차원이든지 함께 DAX 데이터를 가져올 수 있습니다.

| 측정기준 | 설명 |
| --- | --- |
| `Account` | 계정의 모든 노드에 대해 집계된 통계를 제공합니다. |
| `ClusterId` | 데이터를 클러스터로 제한합니다. |
| `ClusterId, NodeId` | 데이터를 클러스터 내 노드로 제한합니다. |

## DAX를 모니터링하는 CloudWatch 경보 생성

경보 때문에 상태가 변경되면 Amazon Simple Notification Service(Amazon SNS) 메시지를 보내는 Amazon CloudWatch 경보를 생성할 수 있습니다. 경보는 지정한 기간 동안 단일 지표를 감시합니다. 기간 수에 대한 주어진 임계값과 지표 값을 비교하여 하나 이상의 작업을 수행합니다. 이 작업은

Amazon SNS 주제 또는 Auto Scaling 정책으로 전송되는 알림입니다. 경보는 지속적인 상태 변경에 대해서만 작업을 호출합니다. CloudWatch 경보는 단순히 특정 상태에 있다고 해서 작업을 호출하지 않습니다. 상태가 변경되어 지정된 기간 수 동안 유지되어야 합니다.

쿼리 캐시 누락 알림을 받는 방법

1. Amazon SNS 주제 `arn:aws:sns:us-west-2:522194210714:QueryMissAlarm`을 생성합니다.

   자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon Simple Notification Service 설정을 참조하세요 .

2. 경보를 만듭니다.

```
aws cloudwatch put-metric-alarm \
    --alarm-name QueryCacheMissesAlarm  \
    --alarm-description "Alarm over query cache misses" \
    --namespace AWS/DAX \
    --metric-name QueryCacheMisses  \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 8 \
    --comparison-operator GreaterThanOrEqualToThreshold \
    --period 60 \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-west-2:522194210714:QueryMissAlarm
```

3. 경보를 테스트합니다.

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value ALARM
```

> **Note**
>
> 애플리케이션에 적절하게 임계값을 높이거나 줄일 수 있습니다. 또한 CloudWatch 지표 수식을 사용하여 캐시 누락 비율 지표를 정의하고 해당 지표에 대해 경보를 설정할 수 있습니다.

요청으로 인해 클러스터에서 내부 오류가 발생하는 경우 알림을 받는 방법

1. Amazon SNS 주제 `arn:aws:sns:us-west-2:123456789012:notify-on-system-errors`을 생성합니다.

   자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon Simple Notification Service 설정을 참조하세요 .

2. 경보를 만듭니다.

```
aws cloudwatch put-metric-alarm \
    --alarm-name FaultRequestCountAlarm \
    --alarm-description "Alarm when a request causes an internal error" \
    --namespace AWS/DAX \
    --metric-name FaultRequestCount \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. 경보를 테스트합니다.

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value ALARM
```

## 프로덕션 모니터링

다양한 시간과 다양한 부하 조건에서 성능을 측정하여 환경에서 일반 DAX 성능의 기준을 설정해야 합니다. DAX를 모니터링할 때 과거 모니터링 데이터를 저장할 것을 고려해야 합니다. 이 저장된 데이터는 현재 성능 데이터와 비교하고, 일반 성능 패턴과 성능 이상을 식별하며 문제 해결 방법을 제안하는 기준이 됩니다.

기준을 설정하려면 로드 테스트 중에 프로덕션 환경에서 최소한 다음 항목을 모니터링해야 합니다.

- CPU 사용률 및 조정된 요청 - 클러스터에서 더 큰 노드 유형을 사용해야 하는지 여부를 확인할 수 있습니다. 클러스터의 CPU 사용률은 `CPUUtilization` CloudWatch 지표를 통해 제공됩니다. 이 지표의 평균 통계는 클러스터의 모든 노드에 대한 평균 CPU 사용률 보기를 제공합니다. 클러스터 규모를 조정할 때는 모든 노드의 최대 사용률인 최대 통계를 고려하는 것이 좋습니다.

  > **Note**
  >
  > AWS는 `CPUUtilization` 지표의 세분성을 개선했습니다. 2024년 5월 17일부터 2024년 6월 22일까지 지표가 변경됩니다.

- 작업 지연 시간(클라이언트 측에서 측정)은 애플리케이션의 지연 시간 요구 사항 내에서 일관되게 유지되어야 합니다.
- 오류 발생률은 `ErrorRequestCount`, `FaultRequestCount` 및 `FailedRequestCount` CloudWatch 지표에서 볼 수 있는 것처럼 낮게 유지되어야 합니다.
- 네트워크 바이트 사용량: 클러스터에서 더 많은 노드를 사용해야 하는지 아니면 더 큰 노드 유형을 사용해야 하는지 결정할 수 있습니다. 사용량을 모니터링하기 위해 CloudWatch에서 사용할 수 있는 `BaselineNetworkBytesInUtilization` 및 `BaselineNetworkBytesOutUtilization` 지표에 대한 알림을 설정할 수 있습니다. 이 지표는 인스턴스 유형에 따라 수신 및 송신 트래픽에 각각 사용 가능한 네트워크 대역폭의 사용률을 나타냅니다.
- 캐시 메모리 사용률 및 제거된 크기. 클러스터의 노드 유형에 작업 세트를 보유할 만큼 충분한 메모리가 있는지 확인하고, 그렇지 않은 경우 더 큰 노드 유형으로 전환할 수 있습니다.

  > **Note**
  >
  > 캐시 누락 및 쓰기 횟수가 많은 경우 캐시 메모리 사용률이 최대 100%까지 증가할 수 있으며 이로 인해 가용성 차질이 발생할 수 있습니다.

- 클라이언트 연결 - 클러스터에 대한 연결에 설명되지 않은 스파이크를 모니터링할 수 있습니다.

## AWS CloudTrail을 사용하여 DAX 작업 로깅

Amazon DynamoDB Accelerator(DAX)는 DAX에서 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다.

DAX 및 CloudTrail에 대한 자세한 내용은 AWS CloudTrail을 사용하여 DynamoDB 작업 로깅의 DynamoDB Accelerator(DAX) 단원을 참조하세요.

# DAX T3/T2 버스트 가능 인스턴스

DAX를 사용하면 고정 성능 인스턴스(R4, R5, R7 등)와 성능 버스트 가능 인스턴스(T2, T3 등) 중에서 선택할 수 있습니다. 버스트 가능 성능 인스턴스는 기본 수준의 CPU 성능 외에 필요할 경우 기준 이상으로 높일 수 있는 기능을 제공합니다.

기본 성능과 기준 이상으로 높일 수 있는 기능은 CPU 크레딧에 의해 좌우됩니다. 버스트 가능 성능 인스턴스는 워크로드가 기준 임계값보다 낮을 때 인스턴스 크기에 따라 결정되는 속도로 CPU 크레딧을 지속적으로 누적합니다. 이러한 크레딧은 워크로드가 증가하면 사용할 수 있습니다. CPU 크레딧은 1분 동안 CPU 코어의 전체 성능을 제공합니다.

많은 워크로드에서 지속적으로 높은 수준의 CPU를 필요로 하지 않지만 필요할 때 매우 빠른 CPU에 완전히 액세스할 수 있으면 상당한 이점을 얻을 수 있습니다. 버스트 가능 성능 인스턴스는 이러한 사용 사례를 위해 특별히 설계되었습니다. 데이터베이스에 지속적으로 높은 CPU 성능이 필요한 경우에는 고정 성능 인스턴스를 사용하는 것이 좋습니다.

## DAX T2 인스턴스 패밀리

DAX T2 인스턴스는 버스트 가능 범용 성능 인스턴스로, 기본 수준의 CPU 성능 외에 필요할 경우 기준 이상으로 높일 수 있는 기능을 제공합니다. T2 인스턴스는 가격 예측 가능성이 필요한 테스트 및 개발 워크로드에 적합합니다. DAX T2 인스턴스는 스탠더드 모드로 구성됩니다. 즉, 인스턴스의 누적된 크레딧이 부족해지면 CPU 사용률이 점차 기본 수준으로 떨어집니다. 스탠더드 모드에 대한 자세한 내용은 Amazon EC2 사용 설명서에서 성능 버스트 가능 인스턴스의 스탠더드 모드를 참조하세요.

## DAX T3 인스턴스 패밀리

DAX T3 인스턴스는 차세대 버스트 가능 범용 인스턴스 유형으로서, 기본 수준의 CPU 성능을 제공하면서도 필요에 따라 언제든지 CPU 사용량을 높일 수 있습니다. T3 인스턴스는 컴퓨팅, 메모리 및 네트워크 리소스를 균형 있게 제공하고 CUP 사용량이 중간 정도이며 사용량이 일시적으로 급증하는 워크로드에 적합합니다. DAX T3 인스턴스는 무제한 모드로 구성되어 있습니다. 즉, 추가 요금을 지불하면 24시간 동안 기준 이상으로 높일 수 있습니다. 무제한 모드에 대한 자세한 내용은 Amazon EC2 사용 설명서에서 성능 버스트 가능 인스턴스의 무제한 모드를 참조하세요.

DAX T3 인스턴스는 워크로드에 필요한 동안 높은 CPU 성능을 유지할 수 있습니다. 대부분의 범용 워크로드에서 T3 인스턴스는 추가 요금 없이 충분한 성능을 제공합니다. 24시간 동안 T3 인스턴스의 평균 CPU 사용률이 기준 이하인 경우에 중간에 발생하는 모든 사용량 급증에는 시간당 T3 인스턴스 가격이 자동으로 적용됩니다.

예를 들어, dax.t3.small 인스턴스는 시간당 CPU 크레딧 24개의 비율로 계속 크레딧을 받습니다. 이 기능은 CPU 코어의 20%에 해당하는 기준 성능을 제공합니다(20% × 60분 = 12분). 인스턴스가 받은 크레딧을 사용하지 않는 경우 해당 크레딧은 CPU 크레딧 밸런스에 최대 576개 CPU 크레딧까지 저장됩니다. t3.small 인스턴스에서 코어의 20% 이상으로 높여야 하는 경우 CPU 크레딧 밸런스에서 가져와 이 급증을 자동으로 처리합니다.

DAX T2 인스턴스는 CPU 크레딧 밸런스가 0으로 내려가면 기본 성능으로 제한되지만, DAX T3 인스턴스는 CPU 크레딧 밸런스가 0인 경우에도 기준 이상으로 높일 수 있습니다. 대부분의 워크로드에 대해 평균 CPU 사용률이 기본 성능 이하인 경우에는 모든 CPU 버스트에 t3.small의 기본 시간당 가격이 적용됩니다. CPU 크레딧 밸런스가 0이 된 후 24시간 동안 인스턴스가 평균 25% CPU 사용률(기준보다 5% 높음)로 실행되는 경우 11.52센트(9.6센트/vCPU 시간 × 1 vCPU × 5% × 24시간)가 추가로 청구됩니다. 요금 세부 정보는 Amazon DynamoDB 요금을 참조하세요.

# DAX 액세스 제어

DynamoDB Accelerator(DAX)는 DynamoDB와 함께 작동하여 애플리케이션에 캐싱 계층을 원활하게 추가하도록 설계되었습니다. 그러나 DAX 및 DynamoDB의 액세스 제어 메커니즘은 별개입니다. 두 서비스 모두 AWS Identity and Access Management(IAM)를 사용하여 각자의 보안 정책을 구현하지만 DAX와 DynamoDB의 보안 모델은 서로 다릅니다.

두 보안 모델을 모두 이해하여 DAX를 사용하는 애플리케이션에 알맞은 보안 조치를 구현하는 것이 좋습니다.

이 단원에서는 DAX가 제공하는 액세스 제어 메커니즘을 설명하고, 사용자의 필요에 맞게 맞춤 설정할 수 있는 샘플 IAM 정책을 제공합니다.

DynamoDB를 사용하면 사용자가 개별 DynamoDB 리소스에 수행할 수 있는 작업을 제한하는 IAM 정책을 생성할 수 있습니다. 예를 들어, 사용자가 특정 DynamoDB 테이블에서 읽기 전용 작업만 수행하도록 하는 사용자 역할을 생성할 수 있습니다. (자세한 내용은 Amazon DynamoDB의 Identity and Access Management 섹션을 참조하세요.) 이에 비해, DAX 보안 모델은 클러스터 보안과 사용자를 대신하여 DynamoDB API 작업을 수행하는 클러스터 기능에 중점을 둡니다.

> **Warning**
>
> 현재 IAM 역할 및 정책을 사용하여 DynamoDB 테이블 데이터에 대한 액세스 권한을 제한하고 있다면, DAX 사용으로 해당 정책이 손상될 수 있습니다. 예를 들어, 사용자가 DAX를 통해 DynamoDB 테이블에 액세스할 수 있지만 DynamoDB에 직접 액세스하는 동일한 테이블에 대한 명시적 액세스 권한은 없을 수 있습니다. 자세한 내용은 Amazon DynamoDB의 Identity and Access Management 섹션을 참조하세요.

DAX는 DynamoDB의 데이터에 대해 사용자 수준 구분을 적용하지 않습니다. 대신, 사용자는 해당 클러스터에 액세스할 때 DAX 클러스터의 IAM 정책 권한을 상속합니다. 따라서 DAX를 통해 DynamoDB 테이블에 액세스하면 DAX 클러스터의 IAM 정책에 있는 권한만 액세스 제어 정책으로 적용됩니다. 다른 권한은 인식되지 않습니다.
격리가 필요한 경우 추가 DAX 클러스터를 생성하고 이에 맞게 각 클러스터에 대한 IAM 정책 범위를 지정하는 것이 좋습니다. 예를 들어, 여러 DAX 클러스터를 생성하고 각 클러스터가 단일 테이블에만 액세스할 수 있도록 허용할 수 있습니다.

## DAX의 IAM 서비스 역할

DAX 클러스터를 생성할 때 클러스터를 IAM 역할에 연결해야 합니다. 이를 클러스터에 대한 서비스 역할이라고 합니다.

DAXCluster01이라는 새 DAX 클러스터를 생성한다고 가정합니다. 이 경우 DAXServiceRole이라는 서비스 역할을 만들고 역할을 DAXCluster01에 연결할 수 있습니다. DAXServiceRole의 정책에서는 DAXCluster01과 상호 작용하는 사용자를 대신하여 DAXCluster01이 수행할 수 있는 DynamoDB 작업을 정의합니다.

서비스 역할을 생성할 때는 DAXServiceRole과 DAX 서비스 자체 간에 신뢰 관계를 지정해야 합니다. 신뢰 관계는 역할을 수임하고 권한을 사용할 수 있는 엔터티를 결정합니다. 다음은 DAXServiceRole의 신뢰 관계 문서의 예입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

이 신뢰 관계에서는 DAX 클러스터가 사용자 대신 DAXServiceRole을 수임하고 DynamoDB API 호출을 수행하도록 허용합니다.

허용되는 DynamoDB API 작업은 IAM 정책 문서에 설명되어 있습니다. 이 문서는 DAXServiceRole에 연결됩니다. 다음은 정책 문서의 예입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DaxAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
            ]
        }
    ]
}
```

이 정책에서는 DAX가 DynamoDB 테이블에서 필요한 DynamoDB API 작업을 수행할 수 있게 허용합니다. `dynamodb:DescribeTable` 작업은 DAX가 테이블에 대한 메타데이터를 유지하는 데 필요하며 다른 작업은 테이블의 항목에 대해 수행되는 읽기 및 쓰기 작업입니다. Books라는 테이블은 us-west-2 리전에 있으며 AWS 계정 ID 123456789012가 소유합니다.

> **Note**
>
> DAX는 크로스 서비스 액세스 중에 혼동된 대리자 문제를 방지하는 메커니즘을 지원합니다. 자세한 내용은 IAM 사용 설명서의 혼동된 대리자 문제를 참조하세요.

## DAX 클러스터 액세스를 허용하는 IAM 정책

DAX 클러스터를 생성했으면 사용자가 DAX 클러스터에 액세스할 수 있도록 사용자에게 권한을 부여해야 합니다.

예를 들어 Alice라는 사용자에게 DAXCluster01에 대한 액세스 권한을 부여한다고 가정합니다. 먼저 수신자가 액세스할 수 있는 DAX 클러스터 및 DAX API 작업을 정의하는 IAM 정책(AliceAccessPolicy)을 만듭니다. 그런 다음 이 정책을 사용자 Alice에게 연결하여 액세스 권한을 부여합니다.

다음은 수신자에게 DAXCluster01에 대한 전체 액세스 권한을 부여하는 정책 문서입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

이 정책 문서에서는 DAX 클러스터에 대한 액세스는 허용하지만 DynamoDB 권한은 부여하지 않습니다. (DynamoDB 권한은 DAX 서비스 역할에 의해 부여됩니다.)

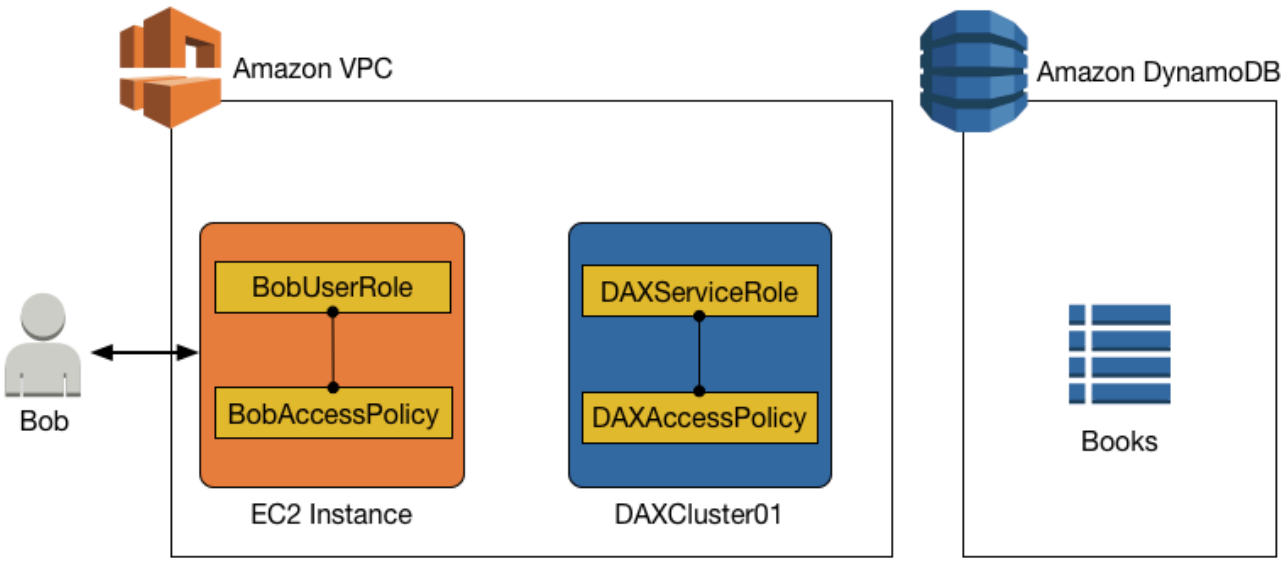
먼저 사용자 Alice에 대해 위에 있는 정책 문서를 사용해 `AliceAccessPolicy`를 생성합니다. 그런 다음 해당 정책을 Alice에 연결합니다.

> **Note**
>
> 정책을 사용자에게 연결하는 대신, IAM 역할에 연결할 수 있습니다. 그렇게 하면 해당 역할을 수임하는 모든 사용자에게 정책에 정의된 권한이 부여됩니다.

사용자 정책에서는 DAX 서비스 역할과 함께 수신자가 DAX를 통해 액세스할 수 있는 DynamoDB 리소스 및 API 작업을 결정합니다.

## 사례 연구: DynamoDB 및 DAX 액세스

다음 시나리오는 DAX와 함께 사용하는 IAM 정책을 더 깊이 이해하는 데 도움이 될 수 있습니다. (이 단원의 나머지 부분에서 이 시나리오를 참조할 것입니다.) 다음은 시나리오에 대한 종합적인 개요를 보여주는 다이어그램입니다.



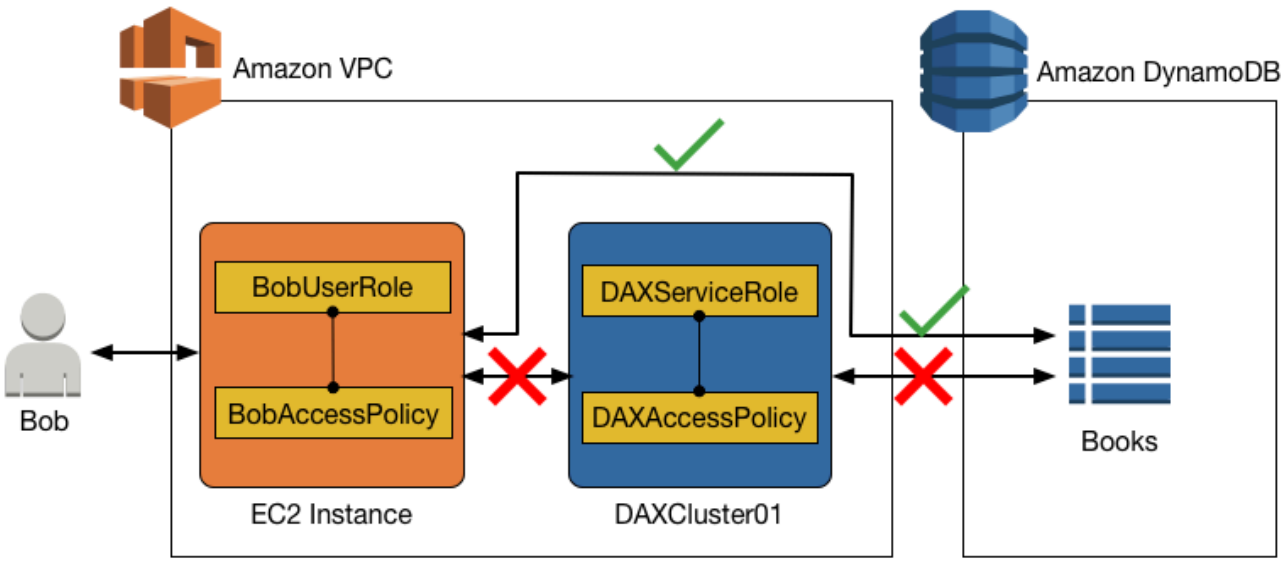

이 시나리오에는 다음 엔터티가 있습니다.

- 사용자(Bob)
- IAM 역할(`BobUserRole`). Bob은 런타임에 이 역할을 수임합니다.
- IAM 정책(`BobAccessPolicy`). 이 정책은 `BobUserRole`에 연결됩니다. `BobAccessPolicy`는 `BobUserRole`이 액세스할 수 있는 DynamoDB 및 DAX 리소스를 정의합니다.
- DAX 클러스터(`DAXCluster01`)

- IAM 서비스 역할(DAXServiceRole). 이 역할을 통해 DAXCluster01은 DynamoDB에 액세스할 수 있습니다.
- IAM 정책(DAXAccessPolicy). 이 정책은 DAXServiceRole에 연결됩니다. DAXAccessPolicy는 DAXCluster01이 액세스할 수 있는 DynamoDB API 및 리소스를 정의합니다.
- DynamoDB 테이블(Books)

BobAccessPolicy 및 DAXAccessPolicy의 정책 문 조합에서는 Bob이 Books 테이블로 할 수 있는 것이 무엇인지 결정합니다. 예를 들어 Bob은 Books에 직접 액세스하거나(DynamoDB 엔드포인트 사용), 간접 액세스하거나(DAX 클러스터 사용), 두 가지 방법을 모두 사용해 액세스할 수 있습니다. 또한 Books에서 데이터를 읽거나 데이터를 Books에 쓰거나 둘 다 수행할 수 있습니다.

## DynamoDB에 액세스할 수는 있지만 DAX로는 액세스할 수 없음




DynamoDB 테이블에 직접 액세스할 수는 있지만 DAX 클러스터를 사용하여 간접 액세스할 수는 없습니다. DynamoDB에 직접 액세스하는 경우 BobUserRole에 대한 권한은 (역할에 연결되는) BobAccessPolicy에 의해 결정됩니다.

## DynamoDB에 대한 읽기 전용 액세스(전용)

Bob은 BobUserRole을 사용해 DynamoDB에 액세스할 수 있습니다. 이 역할(BobAccessPolicy)에 연결된 IAM 정책은 BobUserRole이 액세스할 수 있는 DynamoDB 테이블과 BobUserRole이 호출할 수 있는 API를 결정합니다.

BobAccessPolicy에 대해 다음 정책 문서를 고려해 보세요.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

이 문서가 BobAccessPolicy에 연결되어 있으면 BobUserRole이 DynamoDB 엔드포인트에 액세스하고 Books 테이블에 대한 읽기 전용 작업을 수행할 수 있습니다.

DAX는 이 정책에 나와 있지 않으므로 DAX를 통한 액세스는 거부됩니다.

## DynamoDB에 대한 읽기/쓰기 액세스(전용)

BobUserRole이 DynamoDB에 대한 읽기/쓰기 액세스 권한이 필요한 경우 다음 정책을 사용하면 됩니다.

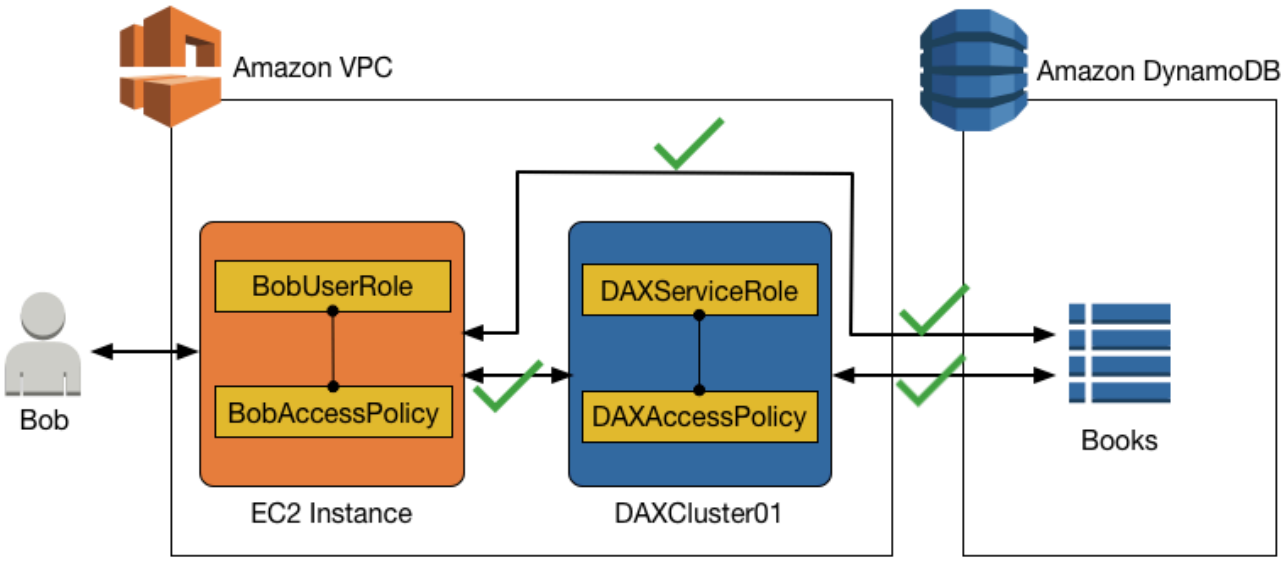
JSON

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

마찬가지로 DAX는 이 정책에 나와 있지 않으므로 DAX를 통한 액세스는 거부됩니다.

## DynamoDB 및 DAX에 대한 액세스



DAX 클러스터에 대한 액세스를 허용하려면 DAX 관련 작업을 IAM 정책에 포함해야 합니다.

다음은 DynamoDB API에 있는 비슷한 이름의 작업에 대응되는 DAX 관련 작업입니다.

- `dax:GetItem`
- `dax:BatchGetItem`
- `dax:Query`
- `dax:Scan`
- `dax:PutItem`
- `dax:UpdateItem`
- `dax:DeleteItem`
- `dax:BatchWriteItem`
- `dax:ConditionCheckItem`

`dax:EnclosingOperation` 조건 키의 경우도 마찬가지입니다.

## DynamoDB에 대한 읽기 전용 액세스 및 DAX에 대한 읽기 전용 액세스

DynamoDB 및 DAX에서 Books 테이블에 읽기 전용 액세스를 할 수 있는 권한이 Bob에게 필요하다고 가정합니다. 다음 정책(BobUserRole에 연결됨)에서 이 액세스 권한을 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
```

```
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

이 정책에는 DAX 액세스에 대한 문(DAXAccessStmt)과 DynamoDB 액세스에 대한 또 하나의 문(DynamoDBAccessStmt)이 있습니다. 이 문에서는 Bob이 GetItem, BatchGetItem, Query 및 Scan 요청을 DAXCluster01로 보내도록 허용합니다.

그러나 DAXCluster01에 대한 서비스 역할에도 DynamoDB의 Books 테이블에 대한 읽기 전용 액세스 권한이 필요합니다. 다음은 이러한 요구 사항을 만족하는 IAM 정책으로서 DAXServiceRole에 연결되어 있습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

## DynamoDB에 대한 읽기/쓰기 액세스 및 DAX를 사용한 읽기 전용 액세스

특정 사용자 역할에 대해 DynamoDB 테이블에 대한 읽기/쓰기 액세스 권한을 제공함과 동시에 DAX를 통한 읽기 전용 액세스 권한을 허용할 수 있습니다.

Bob의 경우 `BobUserRole`의 IAM 정책은 `Books` 테이블에 대한 DynamoDB 읽기 및 쓰기 작업을 허용하면서 `DAXCluster01`을 통한 읽기 전용 작업을 지원해야 합니다.

다음은 이러한 액세스 권한을 부여하는 `BobUserRole`에 대한 정책 문서의 예입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
```

```
    ]
}
```

이외에도 DAXServiceRole에는 DAXCluster01이 Books 테이블에 대한 읽기 전용 작업을 수행하도록 허용하는 IAM 정책이 필요합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:DescribeTable"
           ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

## DynamoDB에 대한 읽기/쓰기 액세스 및 DAX에 대한 읽기/쓰기 액세스

DynamoDB에서 직접 또는 DAXCluster01에서 간접적으로 Books 테이블에 대해 읽기/쓰기 액세스를 할 수 있는 권한이 Bob에게 필요하다고 가정합시다. 다음 정책 문서(BobAccessPolicy에 연결됨)에서 이 액세스 권한을 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
```

```
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

이외에도 DAXServiceRole에는 DAXCluster01이 Books 테이블에 대한 읽기/쓰기 작업을 수행하도록 허용하는 IAM 정책이 필요합니다.

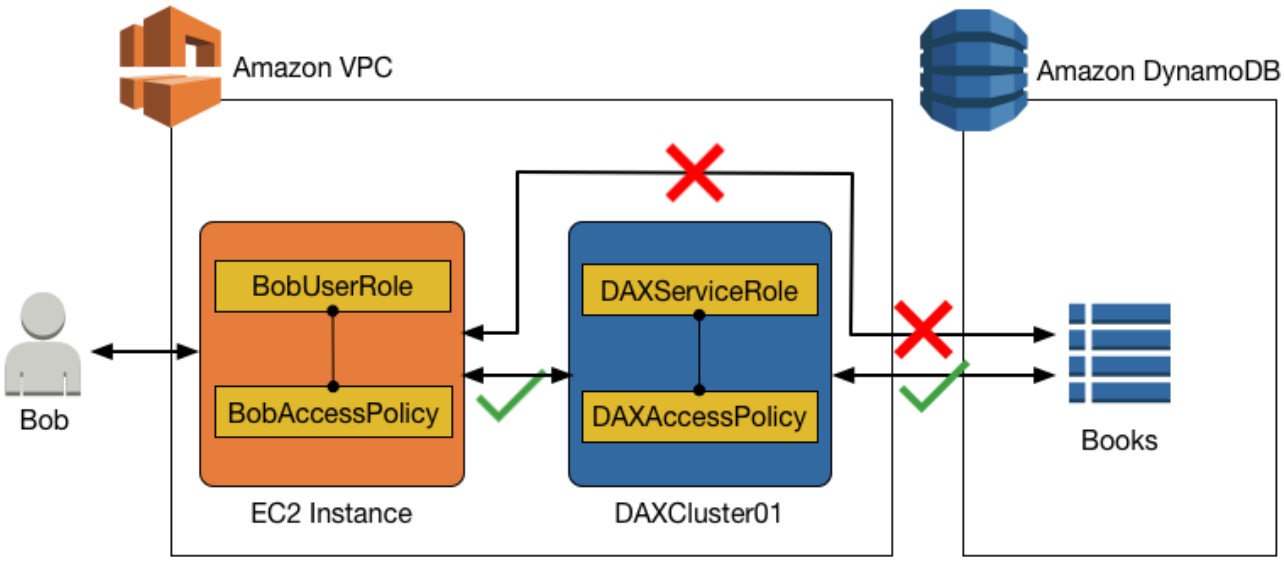
JSON

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

## DAX를 통해 DynamoDB에 액세스할 수 있지만 DynamoDB에 직접 액세스할 수는 없음

이 시나리오에서는 Bob이 DAX를 통해 Books 테이블에 액세스할 수 있지만 DynamoDB의 Books 테이블에는 직접 액세스할 수 없습니다. 따라서, Bob에게 DAX에 대한 액세스 권한이 부여되면, 다른 방법으로는 액세스할 수 없는 DynamoDB 테이블에도 액세스할 수 있게 됩니다. DAX 서비스 역할에 대해 IAM 정책을 구성하는 경우에는 사용자 액세스 정책을 통해 DAX 클러스터에 대한 액세스 권한을 부여받은 사용자가 해당 정책에서 지정한 테이블에 대한 액세스 권한을 갖게 된다는 점에 유의하세요. 이 경우 BobAccessPolicy는 DAXAccessPolicy에 지정된 테이블에 대한 액세스 권한을 갖게 됩니다.




현재 IAM 역할 및 정책을 사용하여 DynamoDB 테이블 및 데이터에 대한 액세스 권한을 제한하고 있는 경우 DAX를 사용하면 이러한 정책이 손상될 수 있습니다. 아래 정책에서는 Bob이 DAX를 통해 DynamoDB 테이블에 액세스할 수 있지만 DynamoDB의 동일한 테이블에는 명시적으로 직접 액세스할 수 없습니다.

다음은 이러한 액세스 권한을 부여하는 정책 문서(`BobAccessPolicy`)로서 `BobUserRole`에 연결됩니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
```

```
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        }
    ]
}
```

이 액세스 정책에는 DynamoDB에 직접 액세스할 수 있는 권한이 없습니다.

BobAccessPolicy와 함께 다음 DAXAccessPolicy에서는 BobUserRole이 Books 테이블에 직접 액세스할 수 없다 하더라도 DynamoDB 테이블 Books에 대한 액세스 권한을 BobUserRole에 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

이 예에 표시된 대로 사용자 액세스 정책 및 DAX 클러스터 액세스 정책에 대해 액세스 제어를 구성할 때는 최소 권한 원칙이 지켜지도록 종단 간 액세스에 대해 온전히 이해하고 있어야 합니다. 또한 사용자에게 DAX 클러스터에 대한 액세스 권한을 부여함으로 인해 이전에 설정한 액세스 제어 정책이 손상되지 않게 해야 합니다.

# DAX 저장 데이터 암호화

Amazon DynamoDB Accelerator(DAX) 저장 데이터 암호화는 기본 스토리지에 대한 무단 액세스로부터 데이터의 보안을 유지할 수 있도록 지원함으로써 추가 계층의 데이터 보호를 제공합니다. 조직의 정책, 업계나 정부 규범 및 규정 준수 요건에 따라 유휴 시 암호화를 사용하여 데이터를 보호해야 할 수 있습니다. 클라우드에 배포된 애플리케이션의 데이터 보안을 향상하기 위해 암호화를 사용할 수 있습니다.

저장 데이터 암호화를 사용하면 DAX에 의해 디스크에 유지되는 데이터가 256비트 Advanced Encryption Standard(AES-256 암호화라고도 함)를 통해 암호화됩니다. DAX는 기본 노드에서 읽기 전용 복제본으로 변경 사항을 전파하는 작업의 일환으로 디스크에 데이터를 작성합니다.

DAX 저장 데이터 암호화는 클러스터를 암호화하는 데 사용되는 단일 서비스 기본 키를 관리하기 위해 AWS Key Management Service(AWS KMS)와 자동으로 통합됩니다. 암호화된 DAX 클러스터를 생성할 때 서비스 기본 키가 존재하지 않는 경우 AWS KMS는 새 AWS 관리형 키를 자동으로 생성합니다. 이 키는 향후 생성되는 암호화된 클러스터에 사용됩니다. AWS KMS는 클라우드에 맞게 조정된 키 관리 시스템을 제공하기 위해 안전하고 가용성이 높은 하드웨어 및 소프트웨어를 결합합니다.

데이터가 암호화된 후 DAX가 성능에 미치는 영향을 최소화한 상태에서 데이터의 복호화를 투명하게 처리합니다. 암호화를 사용하도록 애플리케이션을 수정하지 않아도 됩니다.

> **Note**
>
> DAX는 모든 단일 DAX 작업에 대해 AWS KMS를 호출하지 않습니다. DAX는 클러스터 시작 시 키만 사용합니다. 액세스가 취소된 경우에도 DAX는 클러스터가 종료될 때까지 계속 데이터에 액세스할 수 있습니다. 고객이 지정한 AWS KMS 키는 지원되지 않습니다.

DAX 저장 데이터 암호화는 다음 클러스터 노드 유형에 사용할 수 있습니다.

| Family | 노드 유형 |
| --- | --- |
| 메모리 최적화(R4, R5, R7) | dax.r4.large |

| Family | 노드 유형 |
| --- | --- |
| | dax.r4.xlarge |
| | dax.r4.2xlarge |
| | dax.r4.4xlarge |
| | dax.r4.8xlarge |
| | dax.r4.16xlarge |
| | dax.r5.large |
| | dax.r5.xlarge |
| | dax.r5.2xlarge |
| | dax.r5.4xlarge |
| | dax.r5.8xlarge |
| | dax.r5.12xlarge |
| | dax.r5.16xlarge |
| | dax.r5.24xlarge |
| | dax.r7i.large |
| | dax.r7i.xlarge |
| | dax.r7i.2xlarge |
| | dax.r7i.4xlarge |
| | dax.r7i.8xlarge |
| | dax.r7i.12xlarge |
| | dax.r7i.16xlarge |
| | dax.r7i.24xlarge |

| Family | 노드 유형 |
| --- | --- |
| 일반용(T2) | dax.t2.small<br>dax.t2.medium |
| 일반용(T3) | dax.t3.small<br>dax.t3.medium |

> **Important**
>
> DAX 저장 데이터 암호화는 `dax.r3.*` 노드 유형에 대해 지원되지 않습니다.

클러스터가 생성된 후에는 유휴 시 암호화를 활성화하거나 비활성화할 수 없습니다. 생성 시 유휴 시 암호화가 활성화되지 않은 경우 클러스터를 다시 생성하여 유휴 시 암호화를 활성화해야 합니다.

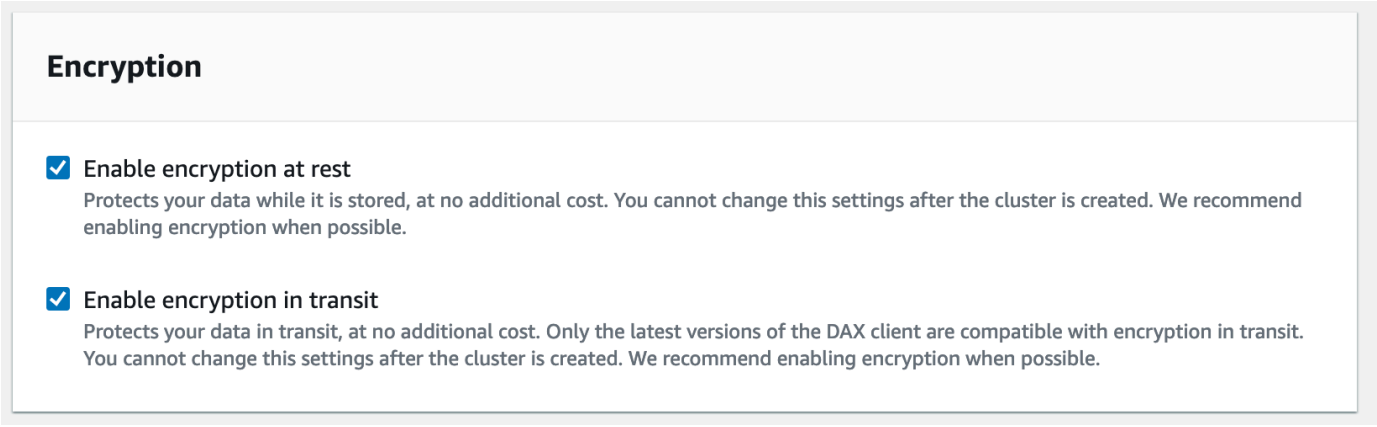
DAX 저장 데이터 암호화는 추가 비용 없이 제공됩니다(AWS KMS 암호화 키 사용 요금이 적용됨). 요금에 대한 자세한 내용은 Amazon DynamoDB 요금을 참조하세요.

## AWS Management Console을 사용하여 저장 데이터 암호화 활성화

다음 단계에 따라 콘솔을 사용하여 테이블에서 유휴 시 DAX 암호화를 활성화합니다.

DAX 저장 데이터 암호화를 활성화하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창의 DAX에서 클러스터를 선택합니다.
3. 클러스터 생성을 선택합니다.
4. 클러스터 이름에 클러스터에 대한 간략한 이름을 입력합니다. 클러스터의 모든 노드에 대해 노드 유형을 선택하고 클러스터 크기에 **3** 노드를 사용합니다.
5. Encryption(암호화)에서 암호화 활성이 선택되어 있는지 확인합니다.




6. IAM 역할, 서브넷 그룹, 보안 그룹 및 클러스터 설정을 선택한 후 클러스터 시작을 선택합니다.

클러스터가 암호화되었는지 확인하려면 클러스터 창에서 클러스터 세부 정보를 확인하세요. 암호화는 활성화됨이어야 합니다.

# DAX 전송 중 데이터 암호화

Amazon DynamoDB Accelerator(DAX)는 애플리케이션과 DAX 클러스터 간에 데이터의 전송 중 암호화를 지원하므로 암호화 요구 사항이 엄격한 애플리케이션에서 DAX를 사용할 수 있습니다.

전송 중 데이터 암호화를 선택했는지 여부에 관계없이 애플리케이션과 DAX 클러스터 간의 트래픽은 Amazon VPC에 남아 있습니다. 이 트래픽은 클러스터의 노드에 연결된 VPC에서 프라이빗 IP를 사용하여 탄력적 네트워크 인터페이스로 라우팅됩니다. VPC를 신뢰 경계로 사용하면 보안 그룹, 네트워크 ACL을 사용한 서브넷 조각화, VPC 흐름 추적 등과 같은 표준 도구를 사용하여 데이터 보안에 대한 제어를 향상할 수 있습니다. DAX 전송 중 데이터 암호화는 이러한 기본 수준의 기밀성에 더하여, 애플리케이션과 클러스터 간의 모든 요청 및 응답이 TLS(전송 수준 보안)로 암호화되고 클러스터에 대한 연결이 클러스터 x509 인증서 확인을 통해 인증될 수 있도록 합니다. DAX 클러스터를 생성할 때 저장 데이터 암호화를 선택하는 경우 DAX에서 디스크에 기록하는 데이터도 암호화됩니다.

DAX에서 전송 중 데이터 암호화는 쉽게 사용할 수 있습니다. 새 클러스터를 생성할 때 이 옵션을 선택하고 애플리케이션에서 최신 버전의 DAX 클라이언트를 사용하면 됩니다. 전송 중 데이터 암호화를 사용하는 클러스터는 암호화되지 않은 트래픽을 지원하지 않으므로 애플리케이션을 잘못 구성하고 암호화를 우회할 가능성이 없습니다. DAX 클라이언트는 연결을 설정할 때 클러스터의 x509 인증서를 사용하여 클러스터의 ID를 인증하므로 DAX 요청이 의도한 위치로 전달됩니다. DAX 클러스터를 생성하는 모든 방법에서 전송 중 데이터 암호화를 지원합니다(AWS Management Console, AWS CLI, 모든 SDK, AWS CloudFormation 등).

기존 DAX 클러스터에서는 전송 중 데이터 암호화를 활성화할 수 없습니다. 기존 DAX 애플리케이션에서 전송 중 데이터 암호화를 사용하려면 전송 중 데이터 암호화를 활성화하고 새 클러스터를 생성하고 애플리케이션의 트래픽을 해당 클러스터로 이동한 다음 이전 클러스터를 삭제합니다.

# DAX에 대한 서비스 연결 IAM 역할 사용

Amazon DynamoDB Accelerator(DAX)는AWS Identity and Access Management(IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 DAX에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 DAX에서 사전 정의하며 서비스에서 사용자 대신 다른 AWS 서비스를 호출하기 위해 필요한 모든 권한을 포함합니다.

서비스 연결 역할을 사용하면 필요한 권한을 수동으로 추가할 필요가 없으므로 DAX를 더 쉽게 설정할 수 있습니다. DAX에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, DAX만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함됩니다. 이 권한 정책은 다른 어떤 IAM 엔터티에도 연결할 수 없습니다.

먼저 역할의 관련 리소스를 삭제해야만 역할을 삭제할 수 있습니다. 이렇게 하면 DAX 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 다른 서비스에 대한 자세한 내용은 IAM 사용 설명서의 IAM으로 작업하는 AWS 서비스 단원을 참조하세요. Service-linked roles(서비스 연결 역할) 열에 Yes(예)라고 표시된 서비스를 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

주제

- DAX에 대한 서비스 연결 역할 권한
- DAX에 대한 서비스 연결 역할 생성
- DAX에 대한 서비스 연결 역할 편집
- DAX에 대한 서비스 연결 역할 삭제

## DAX에 대한 서비스 연결 역할 권한

DAX에서는 `AWSServiceRoleForDAX`라는 서비스 연결 역할을 사용합니다. 이 역할을 사용하여 DAX가 DAX 클러스터 대신 서비스를 호출할 수 있습니다.

> **Important**
>
> AWSServiceRoleForDAX 서비스 연결 역할을 사용하여 DAX 클러스터를 보다 손쉽게 설정하고 유지할 수 있습니다. 그러나 클러스터를 사용하려면 여전히 각 클러스터에 DynamoDB에 액세스 권한을 부여해야 합니다. 자세한 내용은 DAX 액세스 제어 단원을 참조하십시오.

AWSServiceRoleForDAX 서비스 연결 역할은 역할을 위임하기 위해 다음 서비스를 신뢰합니다.

- dax.amazonaws.com

역할 권한 정책을 통해 DAX가 지정된 리소스에서 다음 작업을 완료할 수 있습니다.

- ec2에 대한 작업:
  - AuthorizeSecurityGroupIngress
  - CreateNetworkInterface
  - CreateSecurityGroup
  - DeleteNetworkInterface
  - DeleteSecurityGroup
  - DescribeAvailabilityZones
  - DescribeNetworkInterfaces
  - DescribeSecurityGroups
  - DescribeSubnets
  - DescribeVpcs
  - ModifyNetworkInterfaceAttribute
  - RevokeSecurityGroupIngress

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한 섹션을 참조하세요.

IAM 엔터티가 AWSServiceRoleForDAX 서비스 연결 역할을 생성하도록 허용하려면

IAM 개체에 대한 권한에 다음 정책 설명을 추가합니다.

```
{
```

```
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "dax.amazonaws.com"}}
}
```

## DAX에 대한 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. 클러스터를 생성할 때 DAX가 서비스 연결 역할을 생성합니다.

> **Important**
>
> DAX 서비스가 서비스 연결 역할을 지원하기 시작한 2018년 2월 28일 이전에 이 서비스를 사용 중이었다면 DAX에서 사용자 계정에 AWSServiceRoleForDAX 역할을 이미 생성한 것입니다. 자세한 내용은 IAM 사용 설명서에서 내 AWS 계정에 표시되는 새 역할을 참조하세요.

이 서비스 연결 역할을 삭제한 다음 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 인스턴스 또는 클러스터를 생성할 때 클러스터가 서비스 연결 역할을 다시 생성합니다.

## DAX에 대한 서비스 연결 역할 편집

DAX에서는 AWSServiceRoleForDAX 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

## DAX에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 그러나 서비스 연결 역할을 삭제하려면 먼저 모든 DAX 클러스터를 삭제해야 합니다.

## 서비스 연결 역할 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에 활성 세션이 없는지 확인하고 역할에서 사용되는 리소스를 모두 제거해야 합니다.

IAM 콘솔에서 서비스 연결 역할에 활성 세션이 있는지 확인하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 역할(Roles)을 선택합니다. 그런 다음 AWSServiceRoleForDAX 역할의 이름(확인란 아님)을 선택합니다.
3. 선택한 역할의 요약 페이지에서 Access Advisor 탭을 선택합니다.
4. 액세스 관리자(Access Advisor) 탭에서 서비스 연결 역할의 최근 활동을 검토합니다.

> **Note**
>
> DAX에서 `AWSServiceRoleForDAX` 역할을 사용하는지 잘 모를 경우에는 역할을 삭제해 볼 수 있습니다. 서비스에서 역할을 사용하는 경우에는 삭제가 안 되어 역할이 사용 중인 리전을 볼 수 있습니다. 역할이 사용 중인 경우에는 DAX 클러스터를 삭제한 다음 역할을 삭제해야 합니다. 서비스 연결 역할에 대한 세션은 취소할 수 없습니다.

`AWSServiceRoleForDAX` 역할을 제거하려면 먼저 모든 DAX 클러스터를 삭제해야 합니다.

모든 DAX 클러스터 삭제

이러한 절차 중 하나에 따라 각 DAX 클러스터를 삭제합니다.

DAX 클러스터를 삭제하려면(콘솔)

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 탐색 창의 DAX에서 클러스터를 선택합니다.
3. 작업을 선택한 후 삭제를 선택합니다.
4. [Delete cluster confirmation] 대화 상자에서 [Delete]를 선택합니다.

DAX 클러스터를 삭제하려면(AWS CLI)

AWS CLI 명령 참조에서 delete-cluster를 참조하세요.

DAX 클러스터를 삭제하려면(API)

Amazon DynamoDB API 참조에서 DeleteCluster를 참조하세요.

서비스 연결 역할 삭제

IAM을 사용하여 수동으로 서비스 연결 역할을 삭제하려면

IAM 콘솔, IAM CLI 또는 IAM API를 사용하여 `AWSServiceRoleForDAX` 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하세요.

# AWS 계정 간 DAX 액세스

한 AWS 계정(계정 A)에서 DynamoDB Accelerator(DAX) 클러스터가 실행 중이고 다른 AWS 계정(계정 B)의 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스에서 DAX 클러스터에 액세스할 수 있어야 한다고 가정합니다. 이 자습서에서는 계정 B의 IAM 역할을 사용하여 계정 B에서 EC2 인스턴스를 시작한 다음, EC2 인스턴스의 임시 보안 자격 증명을 사용하여 계정 A의 IAM 역할을 수임합니다. 마지막으로 임시 보안 자격 증명을 사용해 계정 A의 IAM 역할을 수임하여 계정 A의 DAX 클러스터에 대한 피어링 연결을 통해 애플리케이션을 호출합니다. 이러한 작업을 수행하려면 두 AWS 계정 모두에서 관리 액세스 권한이 필요합니다.

> **Important**
>
> DAX 클러스터가 다른 계정에서 DynamoDB 테이블에 액세스하도록 할 수는 없습니다.

주제

- IAM 설정
- VPC 설정
- 크로스 계정 액세스를 허용하도록 DAX 클라이언트 수정

## IAM 설정

1. 다음 콘텐츠로 이름이 `AssumeDaxRoleTrust.json`인 텍스트 파일을 만들어서 Amazon EC2가 사용자를 대신하여 작업하도록 허용합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. 계정 B에서 인스턴스를 시작할 때 Amazon EC2가 사용할 수 있는 역할을 생성합니다.

```
aws iam create-role \
    --role-name AssumeDaxRole \
    --assume-role-policy-document file://AssumeDaxRoleTrust.json
```

3. 다음 콘텐츠로 이름이 AssumeDaxRolePolicy.json인 텍스트 파일을 만들어서 계정 B의 EC2 인스턴스에서 실행 중인 코드가 계정 A의 IAM 역할을 수임하도록 허용합니다. *accountA*를 계정 A의 실제 ID로 바꿉니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::111122223333:role/DaxCrossAccountRole"
        }
    ]
}
```

4. 방금 생성한 역할에 해당 정책을 추가합니다.

```
aws iam put-role-policy \
    --role-name AssumeDaxRole \
    --policy-name AssumeDaxRolePolicy \
    --policy-document file://AssumeDaxRolePolicy.json
```

5. 인스턴스 프로파일을 생성하여 인스턴스가 역할을 사용하도록 허용합니다.

```
aws iam create-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile
```

6. 역할을 인스턴스 프로파일과 연결합니다.

```
aws iam add-role-to-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile \
    --role-name AssumeDaxRole
```

7. 다음 콘텐츠로 이름이 DaxCrossAccountRoleTrust.json인 텍스트 파일을 만들어서 계정 B가 사용자 A 역할을 수임하도록 허용합니다. *accountB*를 계정 B의 실제 ID로 바꿉니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/AssumeDaxRole"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

8. 계정 A에서 계정 B가 수임할 수 있는 역할을 생성합니다.

```
aws iam create-role \
    --role-name DaxCrossAccountRole \
    --assume-role-policy-document file://DaxCrossAccountRoleTrust.json
```

9. DAX 클러스터에 대한 액세스를 허용하는 이름DaxCrossAccountPolicy.json이 인 텍스트 파일을 생성합니다. *dax-cluster-arn*을 DAX 클러스터의 올바른 Amazon 리소스 이름(ARN)으로 바꿉니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-east-1:111122223333:cache/dax-
cluster-name"
        }
    ]
}
```

10. 계정 A에서 역할에 정책을 추가합니다.

```
aws iam put-role-policy \
    --role-name DaxCrossAccountRole \
    --policy-name DaxCrossAccountPolicy \
    --policy-document file://DaxCrossAccountPolicy.json
```

## VPC 설정

1. 계정 A의 DAX 클러스터의 서브넷 그룹을 찾습니다. *cluster-name*을 계정 B가 액세스해야 하는 DAX 클러스터의 이름으로 바꿉니다.

```
aws dax describe-clusters \
    --cluster-name cluster-name
    --query 'Clusters[0].SubnetGroup'
```

2. 해당 *subnet-group*을 사용하여 클러스터의 VPC를 찾습니다.

```
aws dax describe-subnet-groups \
    --subnet-group-name subnet-group \
    --query 'SubnetGroups[0].VpcId'
```

3. 해당 *vpc-id*를 사용하여 VPC의 CIDR을 찾습니다.

```
aws ec2 describe-vpcs \
    --vpc vpc-id \
    --query 'Vpcs[0].CidrBlock'
```

4. 계정 B에서 이전 단계에서 찾은 것과 겹치지 않는 다른 CIDR을 사용하여 VPC를 생성합니다. 그런 다음 하나 이상의 서브넷을 생성합니다. VPC 생성 마법사는 AWS Management Console 또는 AWS CLI에서 사용할 수 있습니다.
5. 계정 B에서 VPC 피어링 연결 생성 및 수락에 설명된 대로 계정 A VPC에 대한 피어링 연결을 요청합니다. 계정 A에서 연결을 수락합니다.
6. 계정 B에서 새 VPC의 라우팅 테이블을 찾습니다. *vpc-id*를 계정 B에서 생성한 VPC의 ID로 바꿉니다.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id,Values=vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

7. 계정 A의 CIDR로 향하는 트래픽을 VPC 피어링 연결로 보내는 라우팅을 추가합니다. 각 *### ### ### ###*를 계정의 올바른 값으로 바꿉니다.

```
aws ec2 create-route \
    --route-table-id accountB-route-table-id \
    --destination-cidr accountA-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

8. 계정 A에서 이전에 찾은 *vpc-id*를 사용하여 DAX 클러스터의 라우팅 테이블을 찾습니다.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id, Values=accountA-vpc-id' \
```

```
    --query 'RouteTables[0].RouteTableId'
```

9. 계정 A에서 계정 B의 CIDR로 향하는 트래픽을 VPC 피어링 연결로 보내는 라우팅을 추가합니다. 각 *### ## ## ###*를 계정의 올바른 값으로 바꿉니다.

```
aws ec2 create-route \
    --route-table-id accountA-route-table-id \
    --destination-cidr accountB-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

10. 계정 B에서 이전에 생성한 VPC에서 EC2 인스턴스를 시작합니다. `AssumeDaxInstanceProfile`을 지정합니다. Launch Wizard는 AWS Management Console 또는 AWS CLI에서 사용할 수 있습니다. 인스턴스의 보안 그룹을 기록해 둡니다.

11. 계정 A에서 DAX 클러스터가 사용하는 보안 그룹을 찾습니다. *cluster-name*을 DAX 클러스터의 이름으로 바꿉니다.

```
aws dax describe-clusters \
    --cluster-name cluster-name \
    --query 'Clusters[0].SecurityGroups[0].SecurityGroupIdentifier'
```

12. 계정 B에서 생성한 EC2 인스턴스의 보안 그룹에서 오는 인바운드 트래픽을 허용하도록 DAX 클러스터의 보안 그룹을 업데이트합니다. *### ## ## ###*를 계정의 올바른 값으로 바꿉니다.

```
aws ec2 authorize-security-group-ingress \
    --group-id accountA-security-group-id \
    --protocol tcp \
    --port 8111 \
    --source-group accountB-security-group-id \
    --group-owner accountB-id
```

이때 계정 B의 EC2 인스턴스에 있는 애플리케이션은 인스턴스 프로파일을 사용하여 `arn:aws:iam::`*accountA-id*`:role/DaxCrossAccountRole` 역할을 수임하고 DAX 클러스터를 사용할 수 있습니다.

# 크로스 계정 액세스를 허용하도록 DAX 클라이언트 수정

> **Note**
>
> AWS Security Token Service(AWS STS) 자격 증명은 임시 자격 증명입니다. 일부 클라이언트는 새로 고침을 자동으로 처리하는 반면 다른 클라이언트는 자격 증명을 새로 고치기 위해 추가 로직이 필요합니다. 적절한 설명서의 지침을 따르는 것이 좋습니다.

Java

이 단원에서는 교차 계정 DAX 액세스를 허용하도록 기존 DAX 클라이언트 코드를 수정할 수 있습니다. DAX 클라이언트 코드가 아직 없는 경우 Java 및 DAX 자습서에서 작업 코드 예제를 찾을 수 있습니다.

1. 다음과 같은 가져오기를 추가합니다.

```
import com.amazonaws.auth.STSAssumeRoleSessionCredentialsProvider;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import
 com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
```

2. AWS STS에서 자격 증명 공급자를 가져오고 DAX 클라이언트 객체를 생성합니다. 각 *### ### ### ###*를 계정의 올바른 값으로 바꿉니다.

```
AWSSecurityTokenService awsSecurityTokenService =
 AWSSecurityTokenServiceClientBuilder
     .standard()
     .withRegion(region)
     .build();

STSAssumeRoleSessionCredentialsProvider credentials =  new
 STSAssumeRoleSessionCredentialsProvider.Builder("arn:aws:iam::accountA:role/
RoleName", "TryDax")
     .withStsClient(awsSecurityTokenService)
     .build();

DynamoDB client = AmazonDaxClientBuilder.standard()
    .withRegion(region)
    .withEndpointConfiguration(dax_endpoint)
    .withCredentials(credentials)
```

```
    .build();
```

### .NET

이 단원에서는 교차 계정 DAX 액세스를 허용하도록 기존 DAX 클라이언트 코드를 수정할 수 있습니다. DAX 클라이언트 코드가 아직 없는 경우 .NET 및 DAX 자습서에서 작업 코드 예제를 찾을 수 있습니다.

1. 솔루션에 AWSSDK.SecurityToken NuGet 패키지를 추가합니다.

```
<PackageReference Include="AWSSDK.SecurityToken" Version="latest version" />
```

2. `SecurityToken` 및 `SecurityToken.Model` 패키지를 사용합니다.

```
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
```

3. `AmazonSimpleTokenService`에서 임시 자격 증명을 가져오고 `ClusterDaxClient` 객체를 생성합니다. 각 *### ## ## ###*를 계정의 올바른 값으로 바꿉니다.

```
IAmazonSecurityTokenService sts = new AmazonSecurityTokenServiceClient();

var assumeRoleResponse = sts.AssumeRole(new AssumeRoleRequest
{
    RoleArn = "arn:aws:iam::accountA:role/RoleName",
                RoleSessionName = "TryDax"
});

Credentials credentials = assumeRoleResponse.Credentials;

var clientConfig = new DaxClientConfig(dax_endpoint, port)
{
    AwsCredentials = assumeRoleResponse.Credentials

};

var client = new ClusterDaxClient(clientConfig);
```

Go

이 단원에서는 교차 계정 DAX 액세스를 허용하도록 기존 DAX 클라이언트 코드를 수정할 수 있습니다. DAX 클라이언트 코드가 아직 없는 경우 GitHub에서 작업 코드 예제를 찾을 수 있습니다.

1. AWS STS 및 세션 패키지를 가져옵니다.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sts"
    "github.com/aws/aws-sdk-go/aws/credentials/stscreds"
)
```

2. AmazonSimpleTokenService에서 임시 자격 증명을 가져오고 DAX 클라이언트 객체를 생성합니다. 각 *### ## ## ###*를 계정의 올바른 값으로 바꿉니다.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String(region)},
)
if err != nil {
    return nil, err
}

stsClient := sts.New(sess)
arp := &stscreds.AssumeRoleProvider{
                Duration:     900 * time.Second,
                ExpiryWindow: 10 * time.Second,
                RoleARN:      "arn:aws:iam::accountA:role/role_name",
                Client:       stsClient,
                RoleSessionName: "session_name",
        }cfg := dax.DefaultConfig()

cfg.HostPorts = []string{dax_endpoint}
cfg.Region = region
cfg.Credentials = credentials.NewCredentials(arp)
daxClient := dax.New(cfg)
```

### Python

이 단원에서는 교차 계정 DAX 액세스를 허용하도록 기존 DAX 클라이언트 코드를 수정할 수 있습니다. DAX 클라이언트 코드가 아직 없는 경우 Python 및 DAX 자습서에서 작업 코드 예제를 찾을 수 있습니다.

1. boto3을 가져옵니다.

```
import boto3
```

2. sts에서 임시 자격 증명을 가져오고 AmazonDaxClient 객체를 생성합니다. 각 *### ## ## ###*를 계정의 올바른 값으로 바꿉니다.

```
sts = boto3.client('sts')
stsresponse =
 sts.assume_role(RoleArn='arn:aws:iam::accountA:role/
RoleName',RoleSessionName='tryDax')
credentials = botocore.session.get_session()['Credentials']

dax = amazondax.AmazonDaxClient(session, region_name=region,
 endpoints=[dax_endpoint], aws_access_key_id=credentials['AccessKeyId'],
 aws_secret_access_key=credentials['SecretAccessKey'],
 aws_session_token=credentials['SessionToken'])
client = dax
```

### Node.js

이 단원에서는 교차 계정 DAX 액세스를 허용하도록 기존 DAX 클라이언트 코드를 수정할 수 있습니다. DAX 클라이언트 코드가 아직 없는 경우 Node.js 및 DAX 자습서에서 작업 코드 예제를 찾을 수 있습니다. 각 *### ## ## ###*를 계정의 올바른 값으로 바꿉니다.

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');
const region = 'region';
const endpoints = [daxEndpoint1, ...];

const getCredentials = async() => {
  return new Promise((resolve, reject) => {
    const sts = new AWS.STS();
    const roleParams = {
      RoleArn: 'arn:aws:iam::accountA:role/RoleName',
```

```
      RoleSessionName: 'tryDax',
    };
    sts.assumeRole(roleParams, (err, session) => {
      if(err) {
        reject(err);
      } else {
        resolve({
          accessKeyId: session.Credentials.AccessKeyId,
          secretAccessKey: session.Credentials.SecretAccessKey,
          sessionToken: session.Credentials.SessionToken,
        });
      }
    });
  });
};

const createDaxClient = async() => {
  const credentials = await getCredentials();
  const daxClient = new AmazonDaxClient({endpoints: endpoints, region: region,
 accessKeyId: credentials.accessKeyId, secretAccessKey: credentials.secretAccessKey,
 sessionToken: credentials.sessionToken});
  return new AWS.DynamoDB.DocumentClient({service: daxClient});
};

createDaxClient().then((client) => {
  client.get(...);
  ...
}).catch((error) => {
  console.log('Caught an error: ' + error);
});
```

# DAX 클러스터 크기 조정 안내서

이 안내서는 애플리케이션에 적합한 Amazon DynamoDB Accelerator(DAX) 클러스터 크기 및 노드 유형을 선택하는 방법에 대한 조언을 제공합니다. 이 지침은 애플리케이션의 DAX 트래픽을 추정하고 클러스터 구성을 선택하고 테스트하는 단계를 안내합니다.

기존 DAX 클러스터가 있고 노드의 수와 크기가 적절한지 여부를 평가하려면 DAX 클러스터 크기 조정 단원을 참조하세요.

주제

- 개요
- 트래픽 추정
- 로드 테스트.


## 개요

새 클러스터를 생성하든 기존 클러스터를 유지하든 상관없이 워크로드에 맞게 DAX 클러스터 크기를 조정하는 것이 중요합니다. 시간이 지남에 따라 애플리케이션의 워크로드가 변경되면 주기적으로 조정 결정을 재검토하여 해당 결정이 여전히 적합한지 확인해야 합니다.

이 프로세스는 일반적으로 다음 단계를 따릅니다.

1. 트래픽 추정. 이 단계에서는 애플리케이션이 DAX에 전송할 트래픽 볼륨, 트래픽의 특성(읽기 및 쓰기 작업) 및 예상 캐시 적중률을 예측합니다.
2. 로드 테스트. 이 단계에서는 클러스터를 생성하고 이전 단계에서 추정한 트래픽을 반영하여 트래픽을 보냅니다. 적합한 클러스터 구성을 찾을 때까지 이 단계를 반복합니다.
3. 프로덕션 모니터링. 애플리케이션이 프로덕션 환경에서 DAX를 사용하는 동안 클러스터를 모니터링하여 시간에 따라 워크로드가 변경되어도 여전히 올바르게 크기 조정되는지 지속적으로 확인해야 합니다.

## 트래픽 추정

일반적인 DAX 워크로드를 특성화하는 다음 세 가지 주요 요소가 있습니다.

- 캐시 적중률
- 초당 읽기 용량 단위(RCU)
- 초당 쓰기 용량 단위(WCU)

### 캐시 적중률 추정

DAX 클러스터가 이미 있는 경우 `ItemCacheHits` 및 `ItemCacheMisses` Amazon CloudWatch 지표를 사용하여 캐시 적중률을 확인할 수 있습니다. 캐시 적중률은 `ItemCacheHits/(ItemCacheHits + ItemCacheMisses)`와(과) 같습니다. 워크로드에 `Query` 또는 `Scan` 작업이 포함되어 있는 경우 `QueryCacheHits`, `QueryCacheMisses`, `ScanCacheHits` 및 `ScanCacheMisses` 지표도 확인해야 합니다. 캐시 적중률은 애플리케이션마다 다르며 클러스터 유지 시간(TTL) 설정의 영향을 많이 받습니다. DAX를 사용하는 애플리케이션의 일반적인 적중률은 85~95%입니다.

## 읽기 및 쓰기 용량 단위 추정

애플리케이션의 DynamoDB 테이블이 이미 있는 경우 `ConsumedReadCapacityUnits` 및 `ConsumedWriteCapacityUnits` CloudWatch 지표를 확인하세요. `Sum` 통계를 사용하고 기간(초)으로 나눕니다.

이미 DAX 클러스터가 있는 경우 DynamoDB `ConsumedReadCapacityUnits` 지표는 캐시 누락만 설명합니다. 따라서 DAX 클러스터에서 처리하는 초당 읽기 용량 단위에 대해 알아보려면 숫자를 캐시 누락률(즉, 1 - 캐시 적중률)로 나눕니다.

DynamoDB 테이블이 없는 경우 읽기 및 쓰기 용량 단위에 대한 설명서를 참조하여 애플리케이션의 추정 요청 비율, 요청당 액세스한 항목 및 항목 크기에 따라 트래픽을 추정합니다.

트래픽을 추정할 때는 클러스터가 트래픽 증가를 위한 충분한 공간을 확보할 수 있도록 향후 증가와 예상된 피크 및 예상치 못한 피크에 대한 계획을 세워야 합니다.

## 로드 테스트.

트래픽을 추정한 후 다음 단계는 로드에서 클러스터 구성을 테스트하는 것입니다.

1. 초기 로드 테스트의 경우 가장 저렴한 고정 성능의 메모리 최적화 노드 유형인 `dax.r4.large` 노드 유형부터 시작하는 것이 좋습니다.
2. 내결함성 클러스터에는 3개의 가용 영역에 분산된 최소 3개의 노드가 필요합니다. 이 경우 가용 영역을 사용할 수 없게 되면 유효 가용 영역 수가 1/3 가량 감소합니다. 초기 로드 테스트의 경우 3노드 클러스터에서 하나의 가용 영역의 장애를 시뮬레이션하는 2노드 클러스터로 시작하는 것이 좋습니다.
3. 로드 테스트 기간 동안 테스트 클러스터로 지속적 트래픽(이전 단계에서 추정)을 유도합니다.
4. 로드 테스트 중에 클러스터의 성능을 모니터링합니다.

이상적으로, 로드 테스트 중에 유도하는 트래픽 프로필은 애플리케이션의 실제 트래픽과 최대한 유사해야 합니다. 여기에는 작업 배포(예: 70% `GetItem`, 25% `Query`, 5% `PutItem`), 각 작업에 대한 요청 비율, 요청당 액세스되는 항목 수 및 항목 크기 배포가 포함됩니다. 애플리케이션의 예상 캐시 적중률과 유사한 캐시 적중률을 달성하려면 테스트 트래픽의 키 분포에 주의해야 합니다.

> **Note**
>
> T2 노드 유형(`dax.t2.small` 및 `dax.t2.medium`)을 로드 테스트할 때 주의하세요. T2 노드 유형은 노드의 CPU 크레딧 밸런스에 따라 시간이 지남에 따라 달라지는 버스트 가능한 CPU

성능을 제공합니다. T2 노드에서 실행 중인 DAX 클러스터가 정상적으로 작동하는 것처럼 보일 수 있지만 노드가 인스턴스의 기준 성능을 초과하면 노드가 누적된 CPU 크레딧 밸런스를 사용하게 됩니다. 크레딧 밸런스가 낮으면 성능이 점차 기준 성과 수준으로 낮아집니다.

로드 테스트 중에 DAX 클러스터를 모니터링하여 로드 테스트에 사용 중인 노드 유형이 적합한 노드 유형인지 확인합니다. 또한 로드 테스트 중에 요청 속도와 캐시 적중률을 모니터링하여 테스트 인프라에서 의도한 트래픽 양을 실제로 유도하는지 확인해야 합니다.

선택한 클러스터 인스턴스 유형의 네트워크 바이트 사용량에 주의를 기울여야 합니다. Amazon EC2 인스턴스의 사용 가능한 기준 대역폭을 초과하면 클러스터가 애플리케이션 워크로드를 감당하지 못할 수 있으므로 규모를 조정해야 합니다.

로드 테스트 결과 선택한 클러스터 구성이 애플리케이션의 워크로드를 유지할 수 없는 것으로 나타나는 경우, 특히 클러스터의 프라이머리 노드에서 CPU 사용률이 높거나 제거율이 높거나 캐시 메모리 사용률이 높은, 더 큰 노드 유형으로 전환해야 합니다. 적중률이 지속적으로 높고 읽기 대 쓰기 트래픽의 비율이 높으면 클러스터에 노드를 추가하는 것을 고려해 볼 수 있습니다. 더 큰 노드 유형을 사용하거나(수직적 조정) 노드를 더 추가하는 경우(수평적 조정)에 대한 추가 지침은 DAX 클러스터 크기 조정 단원을 참조하세요.

클러스터 구성을 변경한 후 로드 테스트를 반복해야 합니다.

# DynamoDB 테이블을 위한 데이터 모델링

데이터 모델링에 대해 자세히 알아보기 전에 DynamoDB의 몇 가지 기본 사항을 이해하는 것이 중요합니다. DynamoDB는 유연한 스키마를 지원하는 키-값 NoSQL 데이터베이스입니다. 각 항목의 주요 속성을 제외한 데이터 속성 세트는 균일하거나 이산적일 수 있습니다. DynamoDB 키 스키마는 파티션 키로 항목을 고유하게 식별하는 단순 프라이머리 키 형식이거나 파티션 키와 정렬 키의 조합으로 항목을 고유하게 정의하는 복합 기본 키 형식입니다. 파티션 키는 해시되어 데이터의 물리적 위치를 확인하고 데이터를 검색합니다. 따라서 데이터를 균일하게 분배하려면 카디널리티가 높고 수평적으로 확장 가능한 속성을 파티션 키로 선택하는 것이 중요합니다. 정렬 키 속성은 키 스키마에서 선택 사항이며 정렬 키가 있으면 DynamoDB에서 일대다 관계를 모델링하고 항목 컬렉션을 생성할 수 있습니다. 정렬 키는 범위 키라고도 합니다. 정렬 키는 항목 컬렉션의 항목을 정렬하는 데 사용되며 유연한 범위 기반 작업을 가능하게 합니다.

DynamoDB 키 스키마에 대한 자세한 내용 및 모범 사례는 다음을 참조하세요.

- the section called “DynamoDB의 파티션 및 데이터 배포”
- the section called “파티션 키 설계”
- the section called “정렬 키 설계”
- 올바른 DynamoDB 파티션 키 선택

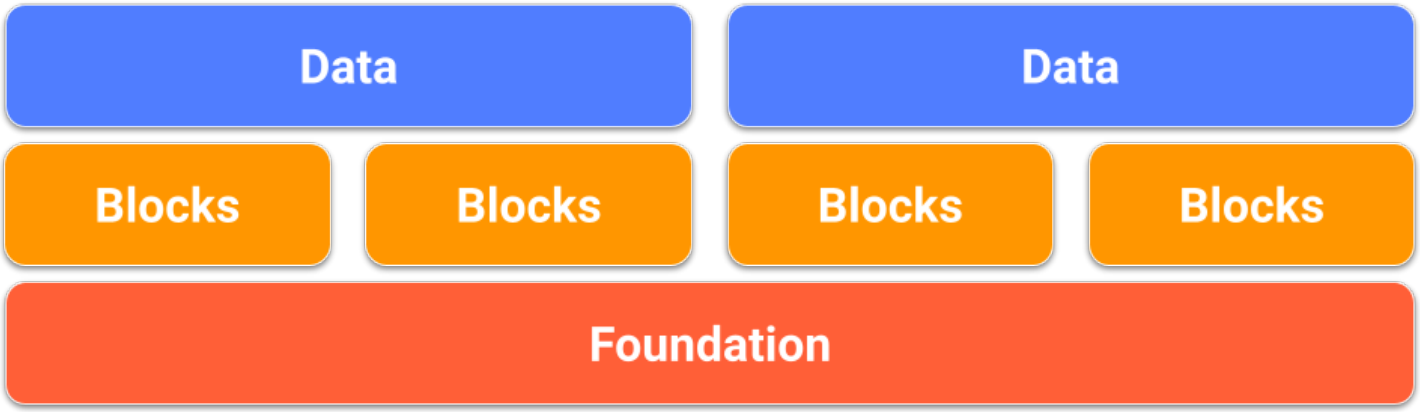
DynamoDB에서 추가 쿼리 패턴을 지원하려면 보조 인덱스가 필요한 경우가 많습니다. 보조 인덱스는 동일한 데이터가 기본 테이블과 다른 키 스키마를 통해 구성된 섀도우 테이블입니다. 로컬 보조 인덱스(LSI)는 기본 테이블과 동일한 파티션 키를 공유하며, 대체 정렬 키를 사용하여 기본 테이블의 용량을 공유할 수 있도록 합니다. 글로벌 보조 인덱스(GSI)는 기본 테이블과 다른 파티션 키와 정렬 키 속성을 가질 수 있습니다. 이는 GSI의 처리량 관리가 기본 테이블과 독립적임을 의미합니다.

보조 인덱스에 대한 자세한 내용 및 모범 사례는 다음을 참조하세요.

- the section called “인덱스 작업”
- the section called “보조 인덱스”

이제 데이터 모델링에 대해 좀더 자세히 살펴보겠습니다. DynamoDB 또는 NoSQL 데이터베이스에서 유연하고 고도로 최적화된 스키마를 설계하는 프로세스는 배우기 어려운 기술일 수 있습니다. 이 모듈의 목표는 사용 사례에서 시작해 프로덕션 단계까지 이어지는 스키마 설계를 위한 멘탈 흐름 차트를 개발하는 데 도움을 주는 것입니다. 먼저 모든 설계의 기본 선택, 즉 단일 테이블 설계와 다중 테이블 설계

를 소개하는 것으로 시작하겠습니다. 그런 다음 애플리케이션의 다양한 조직적 결과 또는 성능 결과를 달성하는 데 사용할 수 있는 여러 설계 패턴(빌딩 블록)을 검토하겠습니다. 마지막으로 다양한 사용 사례와 산업을 위한 완전한 스키마 설계 패키지가 다양하게 포함되어 있습니다.




주제

- 항목 컬렉션 - DynamoDB에서 일대다 관계를 모델링하는 방법
- DynamoDB의 데이터 모델링 기초
- DynamoDB의 데이터 모델링 빌딩 블록
- DynamoDB의 데이터 모델링 스키마 설계 패키지
- DynamoDB의 관계형 데이터 모델링 모범 사례

## 항목 컬렉션 - DynamoDB에서 일대다 관계를 모델링하는 방법

DynamoDB에서 항목 컬렉션은 동일한 파티션 키 값을 공유하는 항목 그룹으로, 항목이 관련되어 있음을 의미합니다. 항목 컬렉션은 DynamoDB에서 일대다 관계를 모델링하는 기본 메커니즘입니다. 항목 컬렉션은 복합 기본 키를 사용하도록 구성된 테이블 또는 인덱스에만 존재할 수 있습니다.

> **Note**
>
> 항목 컬렉션은 기본 테이블 또는 보조 인덱스에 존재할 수 있습니다. 항목 컬렉션이 인덱스와 상호 작용하는 방식에 대한 자세한 내용은 로컬 보조 인덱스의 항목 컬렉션 단원을 참조하세요.

세 명의 서로 다른 사용자와 게임 내 인벤토리를 보여 주는 다음 표를 살펴보십시오.

| Primary key | | Attributes | | |
|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | |
| account1234 | inventory::armor | data | | |
| | | {"armor": [ {"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382} ]} | | |
| | inventory::weapons | data | | |
| | | {"weapons" : [{ "name" : "Sword of the Ancients", "type" : "sword", "gear score" : 320} ]} | | |
| | login-data | pw | state | last-login |
| | | d1e8a70b5ccab1dc2f56bbf7e99f064a660c08e361a35751b9c483c88943d082 | Active | 1649276737 |
| account1387 | info | data | | |
| | | {"email" : "bot123@gmail.com"} | | |
| | inventory::armor | data | | |
| | | {"armor": [ {"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382} ]} | | |
| | login-data | pw | state | last-login |
| | | k2g8jk0m5ppab1dc2f56bbf7e99f064a660c08e361a35751b9c464r23943i082 | Banned | 1649456737 |
| account1138 | info | data | | |
| | | {"email" : "luh-3417@gmail.com" } | | |
| | login-data | pw | state | last-login |
| | | 88A41A9A62B11CCC8C120861928765A3EA41DEB9EAFE261D90F619473B89A2D4 | Active | 14275516966 |

각 컬렉션의 일부 항목에 대해 정렬 키는 `inventory::armor`, `inventory::weapon` 또는 `info`와 같은 데이터를 그룹화하는 데 사용되는 정보로 구성된 연결입니다. 각 항목 컬렉션은 이러한 속성의 서로 다른 조합을 정렬 키로 가질 수 있습니다. 사용자 `account1234`에게는 `inventory::weapons` 항목이 있지만 사용자 `account1387`에게는 없습니다(아직 찾지 못했기 때문). 사용자 `account1138`은 정렬 키로 두 개의 항목만 사용하고(아직 인벤토리가 없기 때문) 다른 사용자는 세 개를 사용합니다.

DynamoDB를 사용하면 이러한 항목 컬렉션에서 항목을 선택적으로 검색하여 다음을 수행할 수 있습니다.

- 특정 사용자의 모든 항목 검색
- 특정 사용자로부터 하나의 항목만 검색
- 특정 사용자에 속하는 특정 유형의 모든 항목 검색

## 항목 컬렉션으로 데이터를 구성하여 쿼리 속도 향상

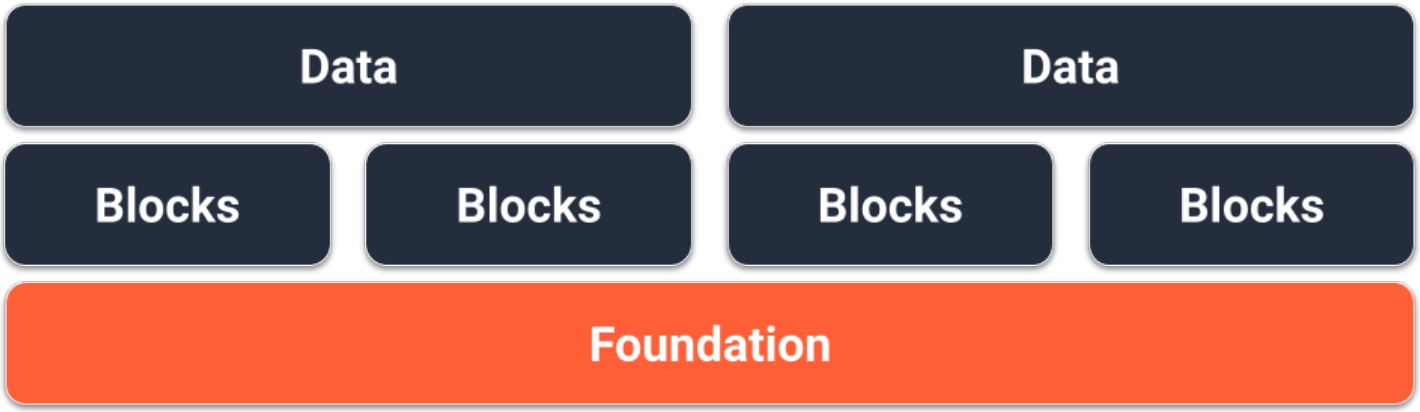
이 예제에서 이 세 항목 컬렉션의 각 항목은 게임 및 플레이어의 액세스 패턴을 기반으로 선택한 플레이어와 데이터 모델을 나타냅니다. 게임에 필요한 데이터는 무엇입니까? 언제 필요합니까? 얼마나 자주 필요합니까? 이렇게 하는 데 비용이 얼마나 듭니까? 이러한 데이터 모델링 결정은 이러한 질문에 대한 답을 바탕으로 이루어졌습니다.

이 게임에서는 무기 인벤토리와 갑옷 인벤토리에 대해 서로 다른 페이지가 플레이어에게 표시됩니다. 플레이어가 인벤토리를 열면 무기가 먼저 표시됩니다. 이 페이지가 매우 빠르게 로드된 다음에 후속 인벤토리 페이지가 로드되도록 하려는 것입니다. 플레이어가 게임 내 항목을 점점 더 많이 획득함에 따라 이러한 각 항목 유형은 상당히 커질 수 있으므로 각 인벤토리 페이지를 데이터베이스의 플레이어 항목 컬렉션에 있는 자체 항목으로 만들었습니다.

다음 단원에서는 `Query` 작업을 통해 항목 컬렉션과 상호 작용하는 방법을 자세히 설명합니다.

# DynamoDB의 데이터 모델링 기초

이 섹션에서는 단일 테이블과 다중 테이블이라는 두 가지 테이블 설계 유형을 살펴보며 기초 계층을 다룹니다.




## 단일 테이블 설계 기초

DynamoDB 스키마의 기초를 위한 한 가지 선택은 단일 테이블 설계입니다. 단일 테이블 설계는 단일 DynamoDB 테이블에 여러 유형(엔터티)의 데이터를 저장할 수 있는 패턴입니다. 그 목표는 여러 테이블을 유지 관리할 필요와 테이블 간의 복잡한 관계를 제거함으로써 데이터 액세스 패턴을 최적화하고 성능을 개선하며 비용을 절감하는 것입니다. 이것이 가능한 이유는 DynamoDB가 동일한 파티션 키를 가진 항목들(항목 컬렉션이라고 함)을 서로 동일한 파티션에 저장하기 때문입니다. 이 설계에서는 서로 다른 유형의 데이터가 동일한 테이블에 항목으로 저장되고, 각 항목은 고유한 정렬 키로 식별됩니다.

| Primary key | | Attributes | |
|---|---|---|---|
| Partition key: PK | Sort key: SK | | |
| UserID | Address#USA#CA#LA#90029 | data | GSI-SK |
| | | "Street Address" | Default |
| | Cart#ACTIVE#Coffee | data | GSI-SK |
| | | CoffeeSKU | 2019-11-27T103324 |
| | Cart#ACTIVE#Spice | data | GSI-SK |
| | | SpiceSKU | 2019-11-28T091245 |
| | Cart#SAVED#Cocoa | data | GSI-SK |
| | | CocoaSKU | 2019-11-28T125642 |
| | OrderHistory#OrderUID | data | GSI-SK |
| | | {Order:DataMap} | 2019-10-08T132612 |
| | ProfileName | data | |
| | | "Paul Atreides" | |
| | Store#StoreUID | data | GSI-SK |
| | | Los Angeles | Active |

장점

- 단일 데이터베이스 직접 호출로 여러 엔터티 유형에 대한 쿼리를 지원하는 데이터 로컬리티
- 전체 읽기 비용 및 지연 시간 비용 절감:
  - 합계 4KB 미만인 두 항목에 대한 단일 쿼리는 0.5RCU 최종적으로 일관된 읽기입니다.
  - 합계 4KB 미만인 두 항목에 대한 두 번의 쿼리는 1RCU 최종적으로 일관된 읽기입니다(각각 0.5 RCU).
  - 두 개의 개별 데이터베이스 직접 호출을 반환하는 데 걸리는 시간은 평균적으로 단일 직접 호출보다 깁니다.
- 관리할 테이블 수 감소:
  - 여러 IAM 역할 또는 정책에서 권한을 유지 관리할 필요가 없습니다.
  - 테이블 용량 관리는 모든 엔터티에서 평균화되므로 일반적으로 소비 패턴의 예측 가능성이 높아집니다.
  - 모니터링에 필요한 경보 감소
  - 고객 관리형 암호화 키는 한 테이블에서만 교체하면 됩니다.

- 테이블로 가는 트래픽 평탄화:
  - 여러 사용량 패턴을 동일한 테이블에 집계하면 전체 사용량이 더 평탄해지고(주가 지수의 성과가 개별 주식보다 평탄해지는 것처럼), 프로비저닝된 모드 테이블을 사용하면 사용률을 높이는 데 더 효과적입니다.

단점

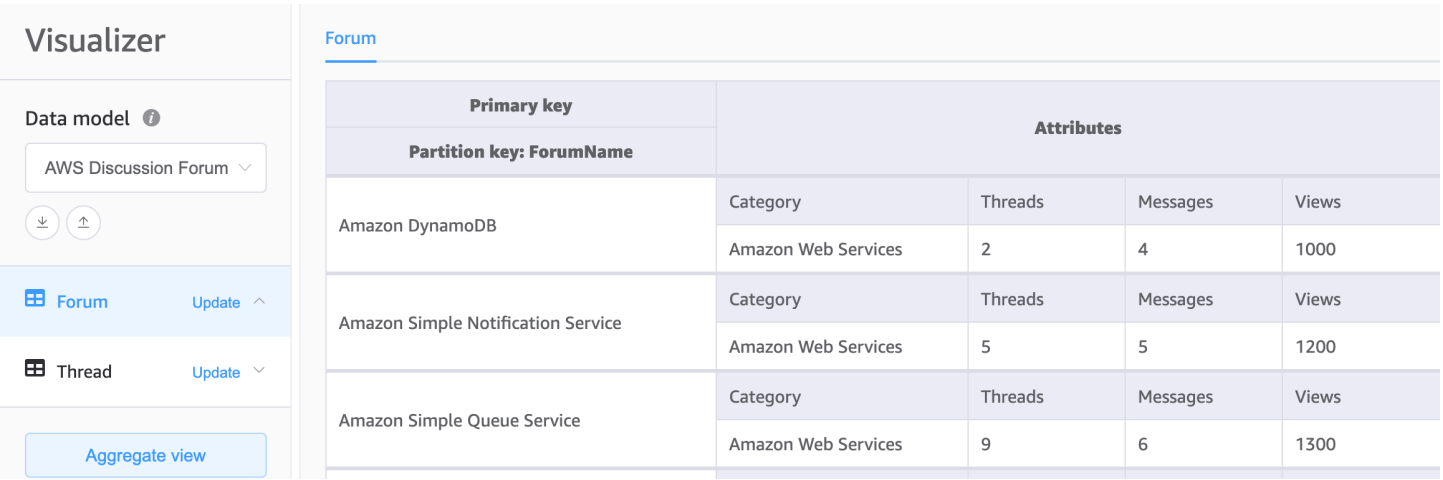
- 관계형 데이터베이스와 비교할 때 역설적인 설계로 인해 학습 곡선이 가팔라질 수 있습니다.
- 모든 엔터티 유형에서 데이터 요구 사항이 일관되어야 합니다.
  - 백업은 전부 아니면 전무 방식이므로 업무상 중요하지 않은 데이터가 있다면 별도의 테이블에 보관하는 것이 좋습니다.
  - 테이블 암호화가 모든 항목에서 공유됩니다. 개별 테넌트 암호화 요구 사항이 있는 멀티테넌트 애플리케이션의 경우, 클라이언트측 암호화가 필요합니다.
  - 기록 데이터와 운영 데이터가 혼합된 테이블에서는 Infrequent Access 스토리지 클래스를 활성화해도 이점이 그리 크지 않습니다. 자세한 내용은 DynamoDB 테이블 클래스 섹션을 참조하세요.
- 일부 엔터티만 처리해야 하는 경우에도 변경된 모든 데이터가 DynamoDB Streams로 전파됩니다.
  - Lambda 이벤트 필터 덕분에 Lambda를 사용할 때는 이것이 청구서에 영향을 미치지 않지만 Kinesis Consumer Library를 사용할 때는 추가 비용이 발생합니다.
- GraphQL을 사용할 경우, 단일 테이블 설계를 구현하기가 더 어렵습니다.
- Java의 `DynamoDBMapper` 또는 Enhanced Client 같은 상위 수준 SDK 클라이언트를 사용하는 경우, 동일한 응답의 항목들이 서로 다른 클래스에 연결될 수 있으므로 결과를 처리하기가 더 어려울 수 있습니다.

사용해야 하는 경우

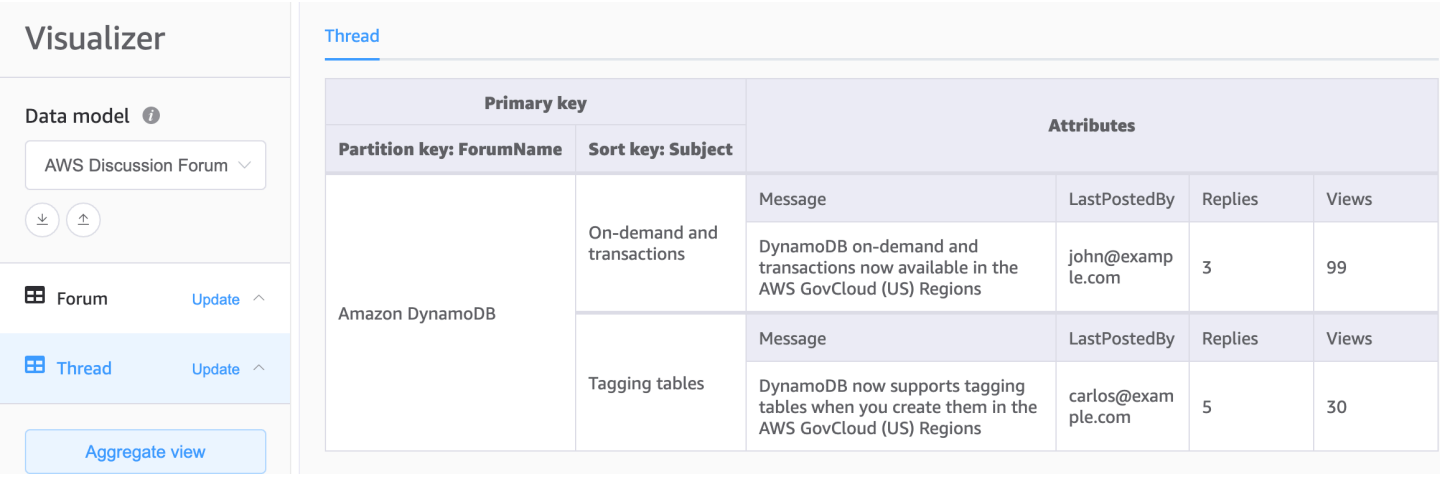
단일 테이블 설계는 여러 엔터티 유형을 함께 자주 쿼리하거나 서로 다른 데이터 유형 간의 관계를 유지해야 하는 애플리케이션에 적합합니다. 액세스 패턴이 데이터 지역성의 이점을 얻고 여러 테이블 관리의 오버헤드를 최소화하려는 경우에 특히 효과적입니다.

## 다중 테이블 설계 기초

DynamoDB 스키마의 기초를 위한 두 번째 선택은 다중 테이블 설계입니다. 다중 테이블 설계는 각 DynamoDB 테이블에 단일 유형(엔터티)의 데이터를 저장하는 기존 데이터베이스 설계와 비슷한 패턴입니다. 각 테이블 내의 데이터는 여전히 파티션 키에 의해 구성되므로 단일 엔터티 유형 내의 성능은 확장성과 성능에 최적화되지만 여러 테이블에 걸친 쿼리는 독립적으로 수행해야 합니다.




| Primary key | Attributes | | | |
|---|---|---|---|---|
| Partition key: ForumName | | | | |
| Amazon DynamoDB | Category | Threads | Messages | Views |
| | Amazon Web Services | 2 | 4 | 1000 |
| Amazon Simple Notification Service | Category | Threads | Messages | Views |
| | Amazon Web Services | 5 | 5 | 1200 |
| Amazon Simple Queue Service | Category | Threads | Messages | Views |
| | Amazon Web Services | 9 | 6 | 1300 |




| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| Partition key: ForumName | Sort key: Subject | | | | |
| Amazon DynamoDB | On-demand and transactions | Message | LastPostedBy | Replies | Views |
| | | DynamoDB on-demand and transactions now available in the AWS GovCloud (US) Regions | john@example.com | 3 | 99 |
| | Tagging tables | Message | LastPostedBy | Replies | Views |
| | | DynamoDB now supports tagging tables when you create them in the AWS GovCloud (US) Regions | carlos@example.com | 5 | 30 |

## 장점

- 단일 테이블 설계에 익숙하지 않은 사용자들이 설계하기에 더 간단합니다.
- 각 리졸버가 단일 엔터티(테이블)에 매핑되므로 GraphQL 리졸버를 더 쉽게 구현할 수 있습니다.
- 다양한 엔터티 유형에서 고유한 데이터 요구 사항 가능:
  - 업무상 중요한 개별 테이블을 백업할 수 있습니다.
  - 각 테이블의 테이블 암호화를 관리할 수 있습니다. 개별 테넌트 암호화 요구 사항이 있는 멀티테넌트 애플리케이션의 경우, 별도의 테넌트 테이블을 통해 각 고객이 자체 암호화 키를 가질 수 있습니다.
  - 기록 데이터가 있는 테이블에서만 Infrequent Access 스토리지 클래스를 활성화하여 비용 절감 효과를 극대화할 수 있습니다. 자세한 내용은 DynamoDB 테이블 클래스 섹션을 참조하세요.
- 각 테이블에는 고유한 변경 데이터 스트림이 있으므로 단일 모놀리식 프로세서 대신 각 항목 유형에 맞는 전용 Lambda 함수를 설계할 수 있습니다.

단점

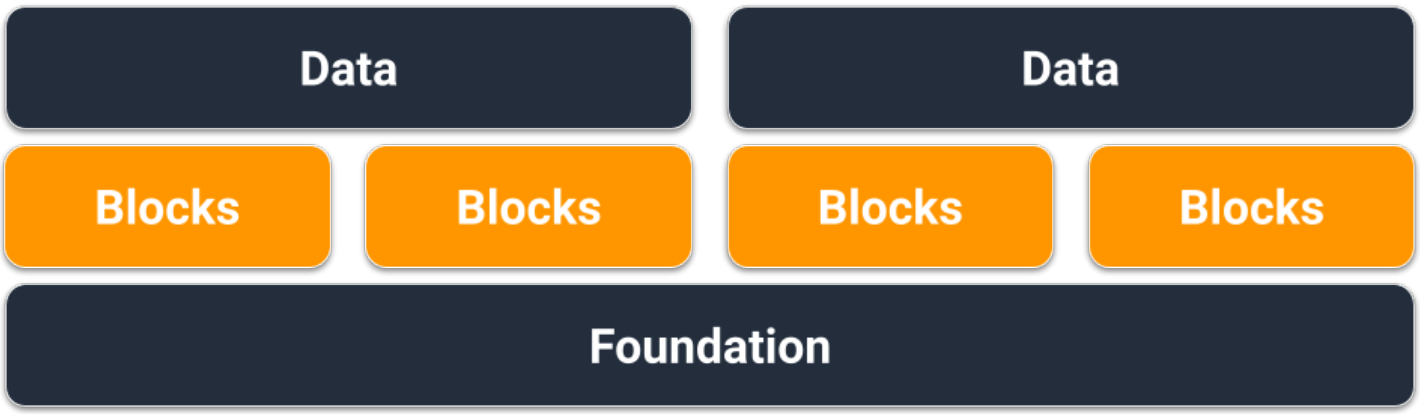
- 여러 테이블의 데이터가 필요한 액세스 패턴의 경우, DynamoDB에서 여러 번 읽어야 하며, 클라이언트 코드에서 데이터를 처리/결합해야 할 수 있습니다.
- 여러 테이블을 운영 및 모니터링하려면 더 많은 CloudWatch 경보가 필요하며, 각 테이블 규모를 독립적으로 조정해야 합니다.
- 각 테이블 권한을 개별적으로 관리해야 합니다. 향후 테이블을 추가하려면 필요한 IAM 역할 또는 정책을 변경해야 합니다.

사용해야 하는 경우

애플리케이션의 액세스 패턴에서 여러 엔터티 또는 테이블을 함께 쿼리할 필요가 없다면 다중 테이블 설계가 좋고 충분한 접근 방식입니다.

# DynamoDB의 데이터 모델링 빌딩 블록

이 섹션에서는 애플리케이션에 사용할 수 있는 설계 패턴을 제공하는 빌딩 블록 계층을 다룹니다.




주제

- 복합 정렬 키 빌딩 블록
- 멀티테넌시 빌딩 블록
- 희소 인덱스 빌딩 블록
- TTL(Time To Live) 빌딩 블록
- 아카이브용 TTL(Time To Live) 빌딩 블록
- 수직 파티셔닝 빌딩 블록

- 쓰기 샤딩 빌딩 블록

## 복합 정렬 키 빌딩 블록

NoSQL을 비관계형이라고 생각할 수도 있습니다. 궁극적으로 관계를 DynamoDB 스키마에 배치하지 못할 이유는 없습니다. 관계는 관계형 데이터베이스 및 그 외래 키와 다르게 보일 뿐입니다. DynamoDB에서 데이터의 논리적 계층 구조를 개발하는 데 사용할 수 있는 가장 중요한 패턴 중 하나는 복합 정렬 키입니다. 가장 일반적인 설계 스타일은 계층 구조의 각 계층(부모 계층 > 자식 계층 > 손자 계층)을 해시태그로 구분하는 것입니다. 예: `PARENT#CHILD#GRANDCHILD#ETC`.

| Primary key | |
|---|---|
| **Partition key: PK** | **Sort key: SK** |
| UserID | CART#ACTIVE#Apples |
| | CART#ACTIVE#Bananas |
| | CART#SAVED#Oranges |
| | CART#SAVED#Pears |
| | WISH#VEGGIES#Carrots |

DynamoDB에서 데이터를 쿼리하려면 파티션 키에 항상 정확한 값이 필요하지만, 이진 트리를 통과하는 것과 비슷하게 왼쪽부터 오른쪽으로 정렬 키에 부분적 조건을 적용할 수 있습니다.

위의 예에는 사용자 세션에서 유지해야 하는 장바구니가 있는 전자 상거래 상점이 있습니다. 사용자가 로그인할 때마다 나중을 위해 저장된 항목을 포함하여 전체 장바구니를 보고 싶어 할 수 있습니다. 하지만 사용자가 결제에 들어갈 때는 구매를 위해 활성 장바구니에 있는 항목만 로드되어야 합니다. 이 두 `KeyConditions`는 모두 명시적으로 CART 정렬 키를 요청하므로 DynamoDB는 읽기 시 추가 위시리스트 데이터를 무시합니다. 저장된 항목과 활성 항목은 모두 동일한 장바구니의 일부이지만 애플리케이션의 서로 다른 부분에서 서로 다르게 처리해야 하므로 정렬 키의 접두사에 `KeyCondition`을 적용하는 것이 애플리케이션의 각 부분에 필요한 데이터만 검색하는 가장 최적화된 방법입니다.

이 빌딩 블록의 주요 특징

- 효과적인 데이터 액세스를 위해 관련 항목들이 서로에 대해 로컬로 저장됩니다
- `KeyCondition` 식을 사용하면 계층 구조의 하위 집합을 선택적으로 검색할 수 있으므로 낭비되는 RCU가 없습니다

- 애플리케이션의 서로 다른 부분이 특정 접두사 아래 항목을 저장하여 항목 덮어쓰기나 쓰기 충돌을 방지합니다

## 멀티테넌시 빌딩 블록

많은 고객이 DynamoDB를 사용하여 멀티테넌트 애플리케이션의 데이터를 호스팅합니다. 이런 시나리오에서 단일 테넌트의 모든 데이터를 테이블의 자체 논리적 파티션에 보관하는 방식으로 스키마를 설계하려 합니다. 이 설계는 동일한 파티션 키를 가진 DynamoDB 테이블의 모든 항목을 가리키는 용어인 항목 컬렉션이라는 개념을 활용합니다. DynamoDB의 멀티테넌시 접근 방식에 대한 자세한 내용은 Multitenancy on DynamoDB를 참조하세요.

| Primary key | | Attributes |
|---|---|---|
| **Partition key: PK** | **Sort key: SK** | |
| UserOne | PhotoID1 | ImageURL |
| | | https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE] |
| | PhotoID2 | ImageURL |
| | | https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE] |
| UserTwo | PhotoID3 | ImageURL |
| | | https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE] |
| | PhotoID4 | ImageURL |
| | | https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE] |
| UserThree | PhotoID5 | ImageURL |
| | | https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE] |

이 예제에서는 사용자가 수천 명인 사진 호스팅 사이트를 운영 중입니다. 각 사용자는 처음에는 자신의 프로필에만 사진을 업로드하지만 기본적으로 사용자가 다른 사용자의 사진을 보는 것을 허용하지 않을 것입니다. 각 사용자의 API 직접 호출 승인에 추가 격리 수준을 추가하여 해당 사용자가 자신의 파티션에서만 데이터를 요청하도록 하는 것이 이상적이지만 스키마 수준에서는 고유한 파티션 키로도 충분합니다.

이 빌딩 블록의 주요 특징

- 한 사용자 또는 테넌트가 읽는 데이터 양은 해당 파티션에 있는 항목의 총량을 초과할 수 없습니다
- 계정 폐쇄나 규정 준수 요청으로 인한 테넌트의 데이터 삭제를 요령 있고 저렴하게 수행할 수 있습니다. 파티션 키가 테넌트 ID와 같은 쿼리를 실행한 다음 반환된 각 프라이머리 키에 대해 `DeleteItem` 작업을 실행하기만 하면 됩니다.

> **Note**
>
> 멀티테넌시를 염두에 두고 설계되었으므로 단일 테이블에서 다양한 암호화 키 제공자를 사용하여 데이터를 안전하게 격리할 수 있습니다. AWS Amazon DynamoDB용 데이터베이스 암호화 SDK를 사용하면 DynamoDB 워크로드에 클라이언트측 암호화를 포함할 수 있습니다. 속성 수준 암호화를 수행하여 DynamoDB 테이블에 저장하기 전에 특정 속성 값을 암호화하고, 전체 데이터베이스를 미리 해독하지 않고도 암호화된 속성을 검색할 수 있습니다.

## 희소 인덱스 빌딩 블록

액세스 패턴에서 드문 항목 또는 상태를 수신하는 항목(에스컬레이션된 응답 필요)과 일치하는 항목을 찾아야 하는 경우가 있습니다. 전체 데이터 세트에서 이러한 항목을 정기적으로 쿼리하는 대신 글로벌 보조 인덱스(GSI)에 데이터가 드물게 로드된다는 사실을 활용할 수 있습니다. 즉, 인덱스에 속성이 정의된 기본 테이블의 항목만 인덱스에 복제하는 것입니다.

| Primary key | | Attributes | | |
|---|---|---|---|---|
| Partition key: DeviceID | Sort key: State#Date | | | |
| d#12345 | NORMAL#2020-04-24T14:55:00 | Operator | Date | |
| | | Liz | 2020-04-24 | |
| | WARNING1#2020-04-24T14:45:00 | Operator | Date | |
| | | Liz | 2020-04-24 | |
| | WARNING1#2020-04-24T14:50:00 | Operator | Date | |
| | | Liz | 2020-04-24 | |
| d#54321 | NORMAL#2020-04-11T06:00:00 | Operator | Date | |
| | | Liz | 2020-04-11 | |
| | NORMAL#2020-04-11T09:30:00 | Operator | Date | |
| | | Sue | 2020-04-11 | |
| | WARNING2#2020-04-11T09:25:00 | Operator | Date | |
| | | Sue | 2020-04-11 | |
| | WARNING3#2020-04-11T05:55:00 | Operator | Date | |
| | | Liz | 2020-04-11 | |
| d#11223 | WARNING4#2020-04-27T16:10:00 | Operator | Date | |
| | | Sue | 2020-04-27 | |
| | WARNING4#2020-04-27T16:15:00 | Operator | Date | EscalatedTo |
| | | Sue | 2020-04-27 | Sara |

| Primary key | | Attributes | |
|---|---|---|---|
| Partition key: EscalatedTo | Sort key: State#Date | | |
| Sara | WARNING4#2020-04-27T16:15:00 | DeviceID | Operator |
| | | d#11223 | Sue |

이 예제에서는 현장의 각 디바이스가 정기적으로 상태를 보고하는 IoT 사용 사례를 볼 수 있습니다. 대부분의 보고서에서는 모든 것이 정상이라고 디바이스가 보고하기를 기대하지만, 때때로 고장이 발생할 수 있으며, 고장은 수리 기술자에게 에스컬레이션해야 합니다. 보고서에 에스컬레이션이 있으면 `EscalatedTo` 속성이 항목에 추가되지만 없다면 이 속성이 나타나지 않습니다. 이 예제의 GSI는

`EscalatedTo`를 기준으로 파티셔닝되며, GSI가 기본 테이블에서 키를 가져오므로 어떤 DeviceID가 언제 고장을 보고했는지 여전히 알 수 있습니다.

DynamoDB에서는 읽기가 쓰기보다 저렴하지만, 특정 항목 유형의 인스턴스가 드물면서도 이를 찾기 위한 읽기는 흔한 사용 사례에서 희소 인덱스는 매우 강력한 도구입니다.

이 빌딩 블록의 주요 특징

- 희소 GSI의 쓰기 및 스토리지 비용은 키 패턴과 일치하는 항목에만 적용되므로 모든 항목이 복제되는 다른 GSI보다 GSI 비용이 훨씬 저렴할 수 있습니다.
- 복합 정렬 키를 사용하면 원하는 쿼리와 일치하는 항목의 범위를 더욱 좁힐 수 있습니다. 예를 들어 정렬 키에 타임스탬프를 사용하여 지난 X분 동안 보고된 고장을 볼 수 있습니다(`SK > 5 minutes ago, ScanIndexForward: False`).

## TTL(Time To Live) 빌딩 블록

대부분의 데이터에는 해당 데이터를 기본 데이터 스토어에 보관할 가치가 있다고 간주되는 일정한 기간이 있습니다. 오래된 데이터를 쉽게 제거할 수 있도록 DynamoDB에는 TTL(Time to Live)이라는 기능이 있습니다. TTL 기능을 사용하면 (과거의) epoch 타임스탬프가 있는 항목에 대한 모니터링이 필요한 특정 속성을 테이블 수준에서 정의할 수 있습니다. 이렇게 하면 만료된 레코드를 무료로 테이블에서 삭제할 수 있습니다.

> **Note**
>
> 글로벌 테이블의 글로벌 테이블 버전 2019.11.21(현재)을 사용하고 Time To Live 기능도 사용한다면 DynamoDB는 TTL 삭제를 모든 복제본 테이블에 복제합니다. 최초 TTL 삭제는 TTL 만료가 발생하는 리전의 쓰기 용량을 사용하지 않습니다. 그러나 복제본 테이블에 복제되는 TTL 삭제는 각 복제본 리전의 복제된 쓰기 용량을 사용하며, 해당 요금이 적용됩니다.

| Primary key | | Attributes | |
|---|---|---|---|
| Partition key: PK | Sort key: MessageTimestamp | | |
| UserID | 2030-06-30T12:12:12 | TTL | Message |
| | | 1909570332 | Hello |
| | 2030-06-30T12:17:22 | TTL | Message |
| | | 1909570647 | DynamoDB |
| | 2030-06-30T12:22:27 | TTL | Message |
| | | 1909570947 | TTL |

이 예제에는 사용자가 짧은 수명의 메시지를 생성할 수 있도록 설계된 애플리케이션이 있습니다. DynamoDB에서 메시지가 생성되면 애플리케이션 코드에 의해 TTL 속성이 7일 후 날짜로 설정됩니다. 약 7일 후, DynamoDB는 이러한 항목의 에포크 타임스탬프가 과거임을 확인하고 삭제합니다.

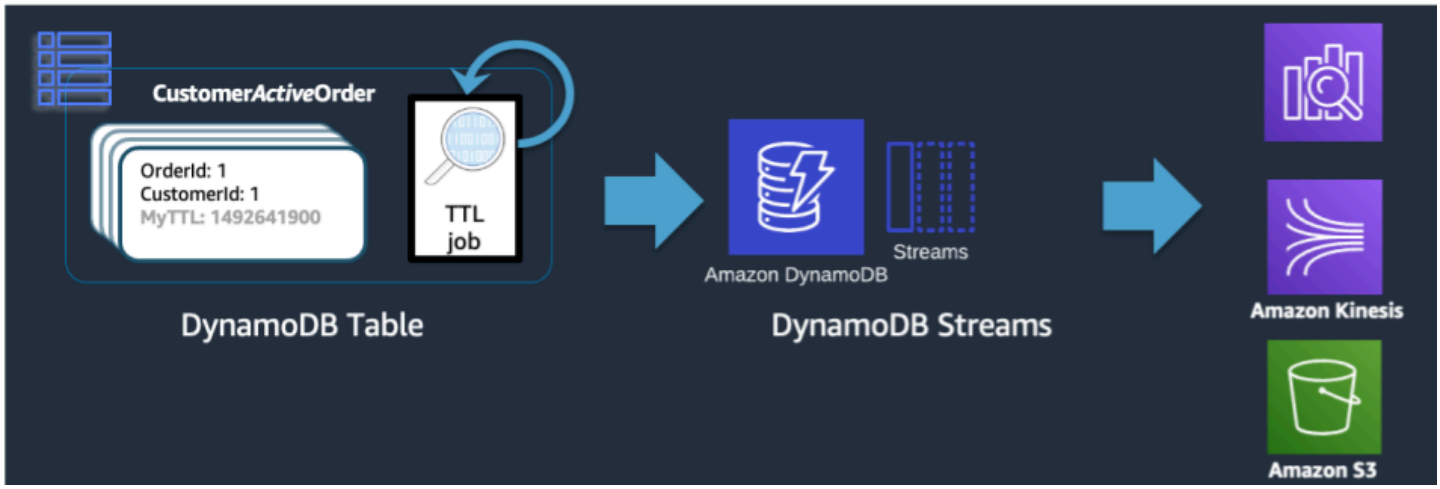
TTL을 통한 삭제는 무료이므로 이 기능을 사용하여 테이블에서 기록 데이터를 제거하는 것이 좋습니다. 이렇게 하면 매달 전체 스토리지 요금이 줄어들고, 사용자 쿼리를 통해 검색되는 데이터가 줄어들기 때문에 사용자 읽기 비용도 절감할 수 있습니다. TTL은 테이블 수준에서 활성화되어 있지만 어떤 항목이나 엔터티에 TTL 속성을 생성할지와 에포크 타임스탬프를 얼마나 먼 장래로 설정할지는 사용자에게 달려 있습니다.

이 빌딩 블록의 주요 특징

- TTL 삭제는 테이블 성능에 영향을 주지 않고 백그라운드에서 실행됩니다.
- TTL은 대략 6시간마다 실행되는 비동기 프로세스이지만 만료된 레코드를 삭제하는 데는 48시간 이상 걸릴 수 있습니다.
  - 오래된 데이터를 48시간 이내에 정리해야 하는 경우, 잠금 기록이나 상태 관리 같은 사용 사례에서 TTL 삭제를 사용하지 마세요.
- TTL 속성에 유효한 속성 이름을 지정할 수 있지만 값은 숫자 유형이어야 합니다.

## 아카이브용 TTL(Time To Live) 빌딩 블록

TTL은 DynamoDB에서 오래된 데이터를 삭제하는 데 효과적인 도구이지만, 기본 데이터 스토어보다 더 긴 기간 동안 데이터 아카이브를 보관해야 하는 사용 사례가 많습니다. 이 경우 TTL의 시간 설정된 레코드 삭제를 활용하면 만료된 레코드를 장기 데이터 스토어로 푸시할 수 있습니다.




DynamoDB가 TTL 삭제를 수행하면 삭제 작업은 DynamoDB Stream에도 `Delete` 이벤트로 푸시됩니다. 하지만 DynamoDB TTL이 삭제를 수행하는 경우, `principal:dynamodb`의 스트림 레코드에 속성이 있습니다. DynamoDB Stream의 Lambda 구독자를 사용하면 DynamoDB 보안 주체 속성에만 이벤트 필터를 적용할 수 있으며, 해당 필터와 일치하는 모든 레코드가 S3 Glacier 같은 아카이브 스토어로 푸시된다는 것을 알 수 있습니다.

이 빌딩 블록의 주요 특징

- 기록 항목에 대해 지연 시간이 짧은 DynamoDB 읽기가 더 이상 필요하지 않게 되면 해당 항목을 S3 Glacier와 같은 콜드 스토리지 서비스로 마이그레이션하여 사용 사례의 데이터 규정 준수 요구 사항을 충족하는 동시에 스토리지 비용을 크게 줄일 수 있습니다.
- 데이터가 Amazon S3에 보관되는 경우, Amazon Athena나 Redshift Spectrum 같은 비용 효율적인 분석 도구를 사용하여 데이터의 기록 분석을 수행할 수 있습니다.

## 수직 파티셔닝 빌딩 블록

문서 모델 데이터베이스에 익숙한 사용자라면 모든 관련 데이터를 단일 JSON 문서에 저장한다는 개념에 익숙할 것입니다. DynamoDB는 JSON 데이터 유형을 지원하지만 중첩된 JSON에서 `KeyConditions`를 실행하는 것은 지원하지 않습니다. 디스크에서 읽는 데이터의 양과 쿼리가 실제로 사용하는 RCU 수를 결정하는 것은 `KeyConditions`이므로 규모가 커지면 비효율성이 초래될 수 있습니다. DynamoDB의 쓰기 및 읽기를 더 잘 최적화하려면 문서의 개별 엔터티를 개별 DynamoDB 항목으로 분리하는 것이 좋습니다. 이를 수직 파티셔닝이라고도 합니다.

```
{
        ”UserProfile” : {
                "FirstName": "Paul",
                "LastName": "Atreides",
                "DateJoined":"1965-08-01"},
        ”Store” : {
                ”store_id”: ”STOREUID”,
                 "city": "Los Angeles",
                 "zip_code":"90029”}
        "ShoppingCart" : [
                {"Spice":
                        { "SKU": "SpiceSKU",
                          "CategoryID":"FictionalSpice",
                          "DateAddded " :"2019-06-11"}},
                {"EspressoBeans":
                        { "SKU": "CaffeineSKU",
                          "CategoryID":"FOODANDDRINK",
                          "DateAddded " :"2019-06-10"}}],
        "ShippingAddress" : {
                 "street_address": "1234 Arrakis Dr",
                 "city": "Los Angeles",
                 "zip_code":"90029",
                 "status": "default"}
        ”OrderHistory#OrderUID” : {
                ”ProductA”: ”SKU_A”,
                ”ProductB”: ”SKU_B”,
                "DateOrdered":”2018-09-28"}
}
```

| Primary key | | Attributes | |
|---|---|---|---|
| Partition key: PK | Sort key: SK | | |
| UserID | Address#USA#CA#LA#90029 | data | GSI-SK |
| | | "Street Address" | Default |
| | Cart#ACTIVE#Coffee | data | GSI-SK |
| | | CoffeeSKU | 2019-11-27T103324 |
| | Cart#ACTIVE#Spice | data | GSI-SK |
| | | SpiceSKU | 2019-11-28T091245 |
| | Cart#SAVED#Cocoa | data | GSI-SK |
| | | CocoaSKU | 2019-11-28T125642 |
| | OrderHistory#OrderUID | data | GSI-SK |
| | | {Order:DataMap} | 2019-10-08T132612 |
| | ProfileName | data | |
| | | "Paul Atreides" | |
| | Store#StoreUID | data | GSI-SK |
| | | Los Angeles | Active |

위에 나온 것처럼 수직 파티셔닝은 실제 단일 테이블 설계의 주요 예이지만 원하는 경우 여러 테이블에 걸쳐 구현할 수도 있습니다. DynamoDB는 1KB 단위로 쓰기 요금을 청구하므로 항목이 1KB 미만이 되는 방식으로 문서를 파티셔닝하는 것이 좋습니다.

이 빌딩 블록의 주요 특징

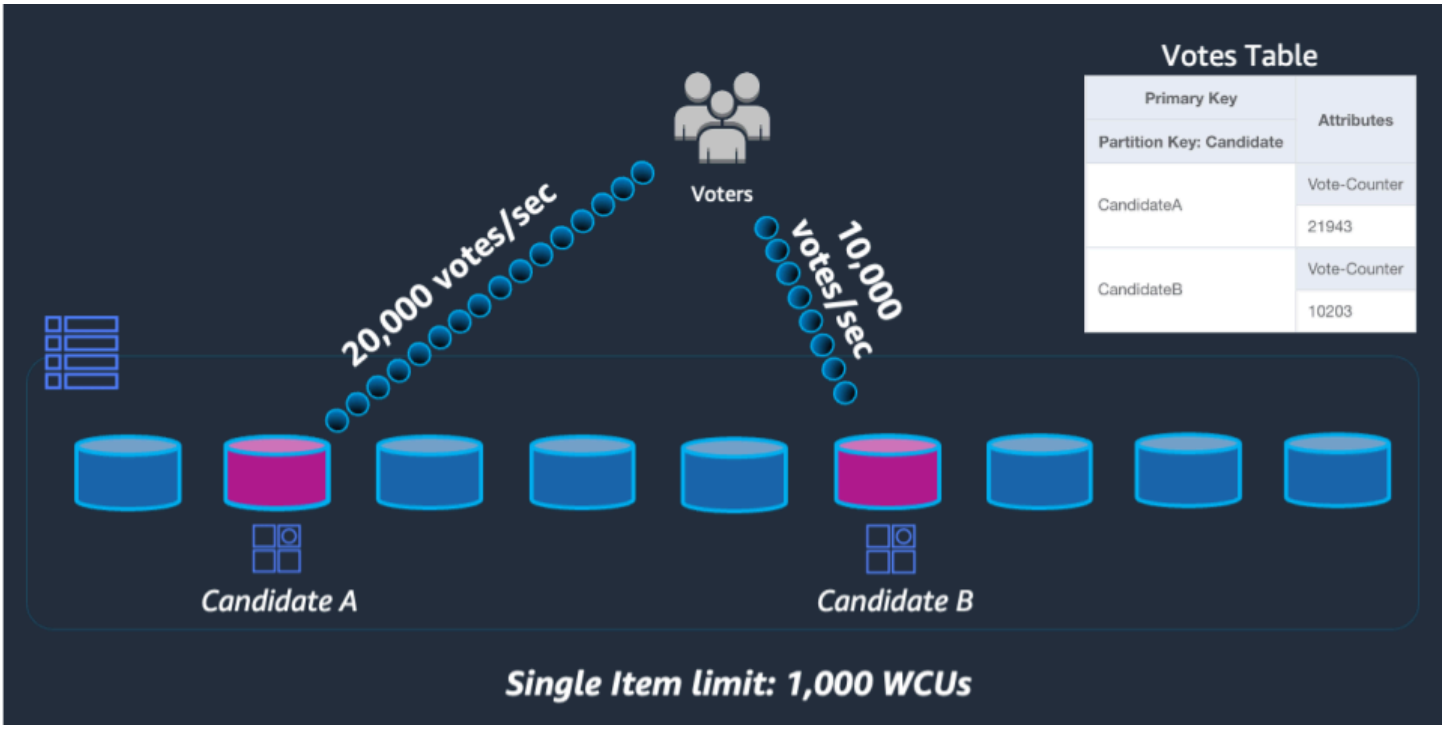
- 데이터 관계의 계층 구조는 정렬 키 접두사를 통해 유지되므로 필요한 경우 클라이언트측에서 단일 문서 구조를 다시 구축할 수 있습니다.
- 데이터 구조의 단일 구성 요소를 독립적으로 업데이트할 수 있으므로 작은 항목 업데이트의 WCU는 1에 불과합니다.
- 정렬 키 `BeginsWith`를 사용하면 애플리케이션이 단일 쿼리로 유사한 데이터를 검색할 수 있으므로 줄어든 총 비용/지연 시간에 대해 읽기 비용이 집계됩니다.
- 큰 문서는 DynamoDB의 개별 항목 크기 제한인 400KB를 쉽게 초과할 수 있으며, 수직 파티셔닝으로 이 제한을 피할 수 있습니다.

# 쓰기 샤딩 빌딩 블록

DynamoDB에 있는 몇 안 되는 엄격한 제한 중 하나는 단일 물리적 파티션이 초당 유지할 수 있는 처리량(단일 파티션 키일 필요는 없음)에 대한 제한입니다. 이러한 제한은 현재 다음과 같습니다.

- 1,000WCU(또는 1KB 이하 항목 초당 쓰기 1,000회) 및 3,000 RCU(또는 4KB 이하 초당 읽기 3,000회)의 강력한 일관성 또는
- 4KB 이하 초당 읽기 6,000회의 최종 일관성

테이블에 대한 요청이 이러한 제한 중 어느 하나라도 초과하는 경우, `ThroughputExceededException`의 클라이언트 SDK에 오류가 다시 전송되는데, 이를 보다 일반적으로는 제한(throttling)이라고 합니다. 이 제한을 초과하는 읽기 작업이 필요한 사용 사례에서는 대부분 DynamoDB 앞에 읽기 캐시를 배치하는 것이 가장 좋지만 쓰기 작업에는 쓰기 샤딩이라는 스키마 수준 설계가 필요합니다.



| Primary Key | Attributes | |
|---|---|---|
| Partition Key: Candidate | | |
| CandidateA#1 | Vote-Counter | Last-Update |
| | 10238 | 2019-09-30T11:35:53 |
| CandidateA#2 | Vote-Counter | Last-Update |
| | 8452 | 2019-09-30T11:35:53 |
| CandidateA#3 | Vote-Counter | Last-Update |
| | 9148 | 2019-09-30T11:35:53 |
| CandidateA#4 | Vote-Counter | Last-Update |
| | 11092 | 2019-09-30T11:35:53 |

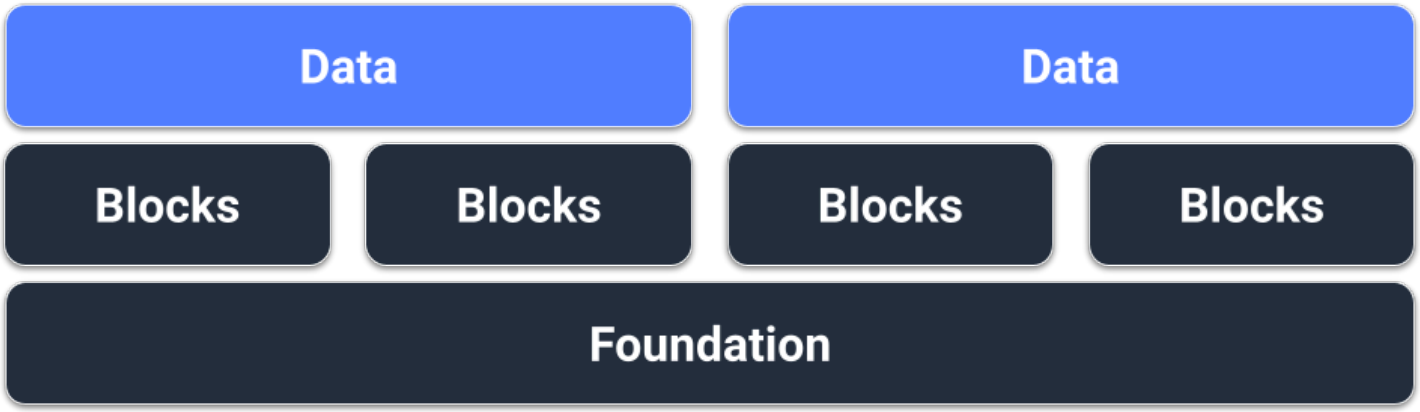
이 문제를 해결하기 위해 애플리케이션의 `UpdateItem` 코드에서 각 참가자의 파티션 키 끝에 무작위 정수를 추가하겠습니다. 난수 생성기의 범위는 주어진 참가자의 초당 예상 쓰기 횟수를 1,000으로 나눈 값과 일치하거나 초과하는 상한값을 가져야 합니다. 초당 20,000회의 투표를 지지하기 위해 이 값은 rand(0,19)처럼 보일 것입니다. 이제 데이터가 별도의 논리적 파티션에 저장되었으므로 읽을 때 데이터를 다시 결합해야 합니다. 투표 합계가 실시간일 필요는 없으므로 X분마다 모든 투표 파티션을 읽도록 예약된 Lambda 함수가 각 참가자에 대해 집계를 때때로 수행하여 실시간으로 읽을 수 있도록 단일 투표 합계 레코드에 다시 쓸 수 있습니다.

이 빌딩 블록의 주요 특징

- 특정 파티션 키의 매우 높은 쓰기 처리량이 불가피한 사용 사례의 경우, 여러 DynamoDB 파티션에 쓰기 작업을 인위적으로 분산할 수 있습니다.
- 카디널리티가 낮은 파티션 키가 있는 GSI도 이 패턴을 활용해야 합니다. GSI에서 제한이 발생하면 기본 테이블의 쓰기 작업에 역압이 가해지기 때문입니다.

# DynamoDB의 데이터 모델링 스키마 설계 패키지

DynamoDB용 데이터 모델링 스키마 설계 패키지에 대해 알아봅니다. 여기에는 소셜 네트워크, 게임 프로필, 불만 사항 관리, 반복 결제, 디바이스 상태 및 온라인 상점에 대한 사용 사례, 액세스 패턴 및 최종 스키마 설계가 포함됩니다.




## 사전 조건

DynamoDB용 스키마를 설계하기 전에 먼저 스키마가 지원해야 하는 사용 사례에 대한 몇 가지 필수 조건 데이터를 수집해야 합니다. 관계형 데이터베이스와 달리 DynamoDB는 기본적으로 샤딩됩니다. 즉, 데이터가 백그라운드에서 여러 서버에 저장되므로 데이터 로컬리티에 맞게 설계하는 것이 중요합니다. 각 스키마 설계마다 다음 목록을 작성해야 합니다.

- 엔터티 목록(ER 다이어그램)
- 각 엔터티의 예상 볼륨 및 처리량
- 지원해야 할 액세스 패턴(쿼리 및 쓰기)
- 데이터 보존 요구 사항

주제

- DynamoDB의 소셜 네트워크 스키마 설계
- DynamoDB의 게임 프로필 스키마 설계
- DynamoDB에서 불만 관리 시스템 스키마 설계
- DynamoDB에서 반복 결제 스키마 설계
- DynamoDB에서 디바이스 상태 업데이트 모니터링
- DynamoDB를 온라인 상점용 데이터 스토어로 사용

# DynamoDB의 소셜 네트워크 스키마 설계

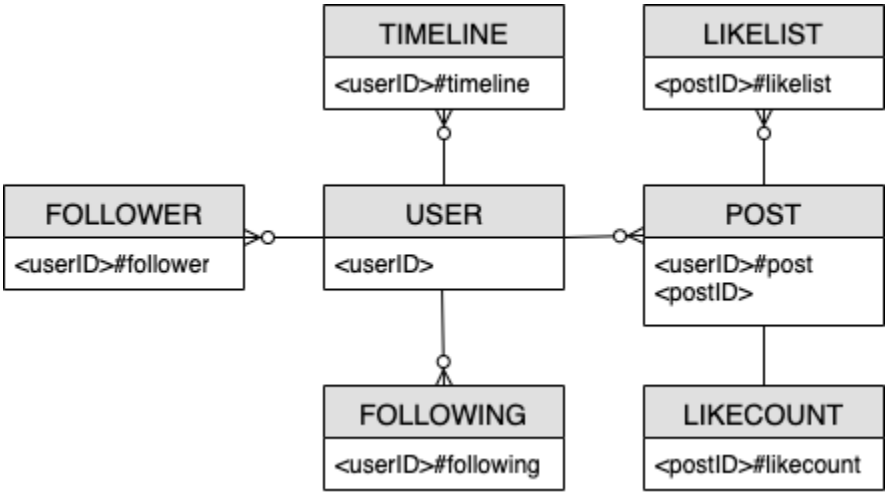
## 소셜 네트워크 비즈니스 사용 사례

이 사용 사례에서는 DynamoDB를 소셜 네트워크로 사용하는 방법을 설명합니다. 소셜 네트워크는 여러 사용자가 서로 상호 작용할 수 있는 온라인 서비스입니다. 설계할 소셜 네트워크에서 사용자는 자신의 게시물, 팔로워, 팔로우하는 사람, 팔로우하는 사람이 작성한 게시물로 구성된 타임라인을 볼 수 있습니다. 이 스키마 설계의 액세스 패턴은 다음과 같습니다.

- 주어진 userID의 사용자 정보 가져오기
- 주어진 userID의 팔로워 목록 가져오기
- 주어진 userID가 팔로우하는 사용자 목록 가져오기
- 주어진 userID의 게시물 목록 가져오기
- 주어진 postID의 게시물을 좋아하는 사용자 목록 가져오기
- 주어진 postID의 좋아요 개수 가져오기
- 주어진 userID의 타임라인 가져오기

## 소셜 네트워크 엔터티 관계 다이어그램

다음은 소셜 네트워크 스키마 설계에 사용할 엔터티 관계 다이어그램(ERD)입니다.




## 소셜 네트워크 액세스 패턴

다음은 소셜 네트워크 스키마 설계 시 고려할 액세스 패턴입니다.

- `getUserInfoByUserID`

- `getFollowerListByUserID`
- `getFollowingListByUserID`
- `getPostListByUserID`
- `getUserLikesByPostID`
- `getLikeCountByPostID`
- `getTimelineByUserID`

## 소셜 네트워크 스키마 설계 진화

DynamoDB는 NoSQL 데이터베이스이므로 여러 데이터베이스의 데이터를 결합하는 조인 작업은 수행할 수 없습니다. DynamoDB에 익숙하지 않은 고객은 그럴 필요가 없을 때 관계형 데이터베이스 관리 시스템(RDBMS) 설계 철학(예: 각 엔터티별 테이블 생성)을 DynamoDB에 적용할 수 있습니다. DynamoDB의 단일 테이블 설계의 목적은 애플리케이션의 액세스 패턴에 따라 미리 조인된 형태로 데이터를 쓴 다음 추가 계산 없이 데이터를 즉시 사용하는 것입니다. 자세한 내용은 Single-table vs. multi-table design in DynamoDB를 참조하세요.

이제 모든 액세스 패턴을 처리하도록 스키마 설계를 발전시키는 방법을 단계별로 살펴보겠습니다.

1단계: 액세스 패턴 1(**`getUserInfoByUserID`**) 처리

주어진 사용자의 정보를 가져오려면 키 조건 `PK=<userID>`를 사용하여 기본 테이블을 `Query`해야 합니다. 쿼리 작업을 사용하면 결과에 페이지를 매길 수 있으며, 이는 사용자의 팔로어가 많을 때 유용할 수 있습니다. 쿼리에 대한 자세한 내용은 DynamoDB에서 테이블 쿼리 섹션을 참조하세요.

이 예제에서는 'count'와 'info'라는 두 가지 유형의 사용자 데이터를 추적합니다. 사용자의 'count'에는 사용자의 팔로워 수, 사용자가 팔로우 중인 사용자 수, 사용자가 작성한 게시물 수가 반영됩니다. 사용자의 'info'에는 이름과 같은 개인 정보가 반영됩니다.

이 두 종류의 데이터는 아래의 두 항목으로 표현됩니다. 정렬 키(SK)에 'count'가 있는 항목은 'info'가 있는 항목보다 변경될 가능성이 높습니다. DynamoDB는 업데이트 전후에 표시되는 항목 크기를 고려하며, 사용된 프로비저닝된 처리량은 이러한 항목 크기 중 더 큰 크기를 반영합니다. 따라서 항목 속성의 하위 집합을 업데이트하더라도 `UpdateItem`은 프로비저닝된 처리량(이전 항목 크기와 이후 항목 크기 중 더 큰 것)을 모두 소비합니다. 한 번의 `Query` 작업으로 항목을 가져오고 `UpdateItem`을 사용하여 기존 숫자 속성에 더하거나 뺄 수 있습니다.

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | |
| u#12345 | "count" | follower# | following# | post# | |
| | | 3000000000 | 971 | 4945 | |
| | "info" | name | content | imageUrl | etc |
| | | hyuklee | My name is Hyuk Lee | s3://.... | ... |

2단계: 액세스 패턴 2(**`getFollowerListByUserID`**) 처리

주어진 사용자를 팔로우하는 사용자 목록을 가져오려면 키 조건 `PK=<userID>#follower`를 사용하여 기본 테이블을 `Query`해야 합니다.

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | |
| u#12345 | "count" | follower# | following# | post# | |
| | | 3000000000 | 971 | 4945 | |
| | "info" | name | content | imageUrl | etc |
| | | hyuklee | My name is Hyuk Lee | s3://.... | ... |
| u#12345#follower | u#23456 | | | | |
| | u#34567 | | | | |
| | u#45678 | | | | |

3단계: 액세스 패턴 3(**`getFollowingListByUserID`**) 처리

주어진 사용자가 팔로우하는 사용자 목록을 가져오려면 키 조건 `PK=<userID>#following`를 사용하여 기본 테이블을 `Query`해야 합니다. 그런 다음 `TransactWriteItems` 작업을 사용하여 여러 요청을 그룹화하고 다음을 수행할 수 있습니다.

- 사용자 A를 사용자 B의 팔로워 목록에 추가한 다음 사용자 B의 팔로워 수를 1 늘립니다.
- 사용자 B를 사용자 A의 팔로워 목록에 추가한 다음 사용자 A의 팔로워 수를 1 늘립니다.

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | |
| u#12345 | "count" | follower# | following# | post# | |
| | | 3000000000 | 971 | 4945 | |
| | "info" | name | content | imageUrl | etc |
| | | hyuklee | My name is Hyuk Lee | s3://.... | ... |
| u#12345#follower | u#23456 | | | | |
| | u#34567 | | | | |
| | u#45678 | | | | |
| u#12345#following | u#56789 | | | | |
| | u#67890 | | | | |
| | u#78912 | | | | |

4단계: 액세스 패턴 4(**`getPostListByUserID`**) 처리

주어진 사용자가 작성한 게시물 목록을 가져오려면 키 조건 `PK=<userID>#post`를 사용하여 기본 테이블을 `Query`해야 합니다. 여기서 한 가지 중요한 점은 사용자의 postID가 증분적이어야 한다는 것입니다. 즉, 두 번째 postID 값은 첫 번째 postID 값보다 커야 합니다(사용자는 자신의 게시물을 정렬된 방식으로 보기 원하므로). 이렇게 하려면 ULID(Universally Unique Lexicographically Sortable Identifier) 같은 시간 값을 기반으로 postID를 생성하면 됩니다.

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | |
| u#12345 | "count" | follower# | following# | post# | |
| | | 3000000000 | 971 | 4945 | |
| | "info" | name | content | imageUrl | etc |
| | | hyuklee | My name is Hyuk Lee | s3://.... | ... |
| u#12345#follower | u#23456 | | | | |
| | u#34567 | | | | |
| | u#45678 | | | | |
| u#12345#following | u#56789 | | | | |
| | u#67890 | | | | |
| | u#78912 | | | | |
| u#12345#post | p#12345 | content | imageUrl | timestamp | |
| | | content for a post | s3://.... | 1571827560 | |
| | p#23456 | content | imageUrl | timestamp | |
| | | content for a post | s3://.... | 1571827561 | |

5단계: 액세스 패턴 5(**`getUserLikesByPostID`**) 처리

주어진 사용자의 게시물에 좋아요를 누른 사용자 목록을 가져오려면 키 조건 PK=<postID>#likelist를 사용하여 기본 테이블을 Query해야 합니다. 이 접근 방식은 액세스 패턴 2(getFollowerListByUserID)와 액세스 패턴 3(getFollowingListByUserID)에서 팔로워 목록 및 팔로우하는 사용자 목록을 검색하는 데 사용한 것과 동일한 패턴입니다.

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| **Partition key: PK** | **Sort key: SK** | | | | |
| u#12345 | "count" | follower# | following# | post# | |
| | | 3000000000 | 971 | 4945 | |
| | "info" | name | content | imageUrl | etc |
| | | hyuklee | My name is Hyuk Lee | s3://.... | ... |
| u#12345#follower | u#23456 | | | | |
| | u#34567 | | | | |
| | u#45678 | | | | |
| u#12345#following | u#56789 | | | | |
| | u#67890 | | | | |
| | u#78912 | | | | |
| u#12345#post | p#12345 | content | imageUrl | timestamp | |
| | | content for a post | s3://.... | 1571827560 | |
| | p#23456 | content | imageUrl | timestamp | |
| | | content for a post | s3://.... | 1571827561 | |
| p#12345#likelist | u#23456 | | | | |
| | u#34567 | | | | |
| | u#45678 | | | | |

6단계: 액세스 패턴 6(**getLikeCountByPostID**) 처리

주어진 게시물에 대한 좋아요 수를 가져오려면 키 조건 PK=<postID>#likecount를 사용하여 기본 테이블에서 GetItem 작업을 수행해야 합니다. 파티션의 처리량이 초당 1,000WCU를 초과하면 제한이 발생하기 때문에 이 액세스 패턴은 팔로워가 많은 사용자(예: 유명인)가 게시물을 작성할 때마다 제한 문제를 일으킬 수 있습니다. 이 문제는 DynamoDB로 인한 것은 아니며, 소프트웨어 스택의 끝에 있기 때문에 DynamoDB에 나타납니다.

모든 사용자가 좋아요 수를 동시에 보는 것이 정말 필요한지 아니면 시간이 지남에 따라 점진적으로 표시되어도 무방한지 평가해야 합니다. 일반적으로 게시물의 좋아요 수가 즉시 100% 정확할 필요는 없습니다. 애플리케이션과 DynamoDB 사이에 대기열을 두어 정기적으로 업데이트가 이루어지도록 하면 이 전략을 구현할 수 있습니다.

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| **Partition key: PK** | **Sort key: SK** | | | | |
| u#12345 | "count" | follower# | following# | post# | |
| | | 3000000000 | 971 | 4945 | |
| | "info" | name | content | imageUrl | etc |
| | | hyuklee | My name is Hyuk Lee | s3://.... | ... |
| u#12345#follower | u#23456 | | | | |
| | u#34567 | | | | |
| | u#45678 | | | | |
| u#12345#following | u#56789 | | | | |
| | u#67890 | | | | |
| | u#78912 | | | | |
| u#12345#post | p#12345 | content | imageUrl | timestamp | |
| | | content for a post | s3://.... | 1571827560 | |
| | p#23456 | content | imageUrl | timestamp | |
| | | content for a post | s3://.... | 1571827561 | |
| p#12345#likelist | u#23456 | | | | |
| | u#34567 | | | | |
| | u#45678 | | | | |
| p#12345#likecount | "count" | etc | | | |
| | | 100 | | | |

7단계: 액세스 패턴 7(**`getTimelineByUserID`**) 처리

주어진 사용자의 타임라인을 가져오려면 키 조건 `PK=<userID>#timeline`를 사용하여 기본 테이블에서 `Query` 작업을 수행해야 합니다. 사용자의 팔로워들이 게시물을 동기식으로 봐야 하는 시나리오를 생각해 보겠습니다. 사용자가 게시물을 작성할 때마다 팔로워 목록을 읽고 팔로워의 userID와 postID가 모든 팔로워의 타임라인 키에 천천히 입력됩니다. 그런 다음 애플리케이션이 시작되면 `Query` 작업으로 타임라인 키를 읽고 새 항목에 `BatchGetItem` 작업을 사용하여 타임라인 화면을 userID와 postID의 조합으로 채울 수 있습니다. API 직접 호출로 타임라인을 읽을 수는 없지만 게시물을 자주 수정할 수 있다면 이것이 더 비용 효율적인 솔루션입니다.

타임라인은 최근 게시물을 보여 주는 곳이므로 이전 게시물을 정리할 방법이 필요합니다. WCU를 사용하여 삭제하는 대신 DynamoDB의 TTL 기능을 사용하여 무료로 삭제할 수 있습니다.

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| **Partition key: PK** | **Sort key: SK** | | | | |
| u#12345 | "count" | follower# | following# | post# | |
| | | 3000000000 | 971 | 4945 | |
| | "info" | name | content | imageUrl | etc |
| | | hyuklee | My name is Hyuk Lee | s3://.... | ... |
| u#12345#follower | u#23456 | | | | |
| | u#34567 | | | | |
| | u#45678 | | | | |
| u#12345#following | u#56789 | | | | |
| | u#67890 | | | | |
| | u#78912 | | | | |
| u#12345#post | p#12345 | content | imageUrl | timestamp | |
| | | content for a post | s3://.... | 1571827560 | |
| | p#23456 | content | imageUrl | timestamp | |
| | | content for a post | s3://.... | 1571827561 | |
| p#12345#likelist | u#23456 | | | | |
| | u#34567 | | | | |
| | u#45678 | | | | |
| p#12345#likecount | "count" | etc | | | |
| | | 100 | | | |
| u#12345#timeline | p#34567#u#56789 | ttl | | | |
| | | 1571827560 | | | |
| | p#45678#u#67890 | ttl | | | |
| | | 1571827560 | | | |
| | p#56789#u#78901 | ttl | | | |
| | | 1571827560 | | | |

모든 액세스 패턴과 스키마 설계에서 이를 처리하는 방법이 아래 표에 요약되어 있습니다.

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 | 기타 조건/필터 |
|---|---|---|---|---|---|
| getUserInfoByUserID | 기본 테이블 | 쿼리 | PK=<userID> | | |
| getFollowerListByUserID | 기본 테이블 | 쿼리 | PK=<userID>#follower | | |

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 | 기타 조건/필터 |
|---|---|---|---|---|---|
| getFollowingListByUserID | 기본 테이블 | 쿼리 | PK=<userID>#following | | |
| getPostListByUserID | 기본 테이블 | 쿼리 | PK=<userID>#post | | |
| getUserLikesByPostID | 기본 테이블 | 쿼리 | PK=<postID>#likelist | | |
| getLikeCountByPostID | 기본 테이블 | GetItem | PK=<postID>#likecount | | |
| getTimelineByUserID | 기본 테이블 | 쿼리 | PK=<userID>#timeline | | |

## 소셜 네트워크 최종 스키마

다음은 최종 스키마 설계입니다. 이 스키마 설계를 JSON 파일로 다운로드하려면 GitHub의 DynamoDB 예제를 참조하세요.

기본 테이블:

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| **Partition key: PK** | **Sort key: SK** | | | | |
| u#12345 | "count" | follower# | following# | post# | |
| | | 3000000000 | 971 | 4945 | |
| | "info" | name | content | imageUrl | etc |
| | | hyuklee | My name is Hyuk Lee | s3://.... | ... |
| u#12345#follower | u#23456 | | | | |
| | u#34567 | | | | |
| | u#45678 | | | | |
| u#12345#following | u#56789 | | | | |
| | u#67890 | | | | |
| | u#78912 | | | | |
| u#12345#post | p#12345 | content | imageUrl | timestamp | |
| | | content for a post | s3://.... | 1571827560 | |
| | p#23456 | content | imageUrl | timestamp | |
| | | content for a post | s3://.... | 1571827561 | |
| p#12345#likelist | u#23456 | | | | |
| | u#34567 | | | | |
| | u#45678 | | | | |
| p#12345#likecount | "count" | etc | | | |
| | | 100 | | | |
| u#12345#timeline | p#34567#u#56789 | ttl | | | |
| | | 1571827560 | | | |
| | p#45678#u#67890 | ttl | | | |
| | | 1571827560 | | | |
| | p#56789#u#78901 | ttl | | | |
| | | 1571827560 | | | |

## 이 스키마 설계와 함께 NoSQL Workbench 사용

이 최종 스키마를 DynamoDB 데이터 모델링, 데이터 시각화, 쿼리 개발 기능을 제공하는 시각적 도구인 NoSQL Workbench로 가져와서 새 프로젝트를 추가로 탐색하고 편집할 수 있습니다. 시작하려면 다음 단계를 따릅니다.

1. NoSQL Workbench 다운로드 자세한 내용은 the section called “다운로드” 단원을 참조하십시오.
2. 위에 나열된 JSON 스키마 파일을 다운로드합니다. 이 파일은 이미 NoSQL Workbench 모델 형식으로 되어 있습니다.
3. JSON 스키마 파일을 NoSQL Workbench로 가져옵니다. 자세한 내용은 the section called “기존 모델 가져오기” 단원을 참조하십시오.

4. NOSQL Workbench로 가져온 후 데이터 모델을 편집할 수 있습니다. 자세한 내용은 the section called “기존 모델 편집” 단원을 참조하십시오.
5. 데이터 모델을 시각화하거나, 샘플 데이터를 추가하거나, CSV 파일에서 샘플 데이터를 가져오려면 NoSQL Workbench의 Data Visualizer 기능을 사용하세요.

# DynamoDB의 게임 프로필 스키마 설계

## 게임 프로필 비즈니스 사용 사례

이 사용 사례에서는 DynamoDB를 사용하여 게임 시스템을 위한 플레이어 프로필을 저장하는 방법을 설명합니다. 사용자(이 경우 플레이어)는 많은 최신 게임, 특히 온라인 게임을 이용하기 전에 프로필을 생성해야 합니다. 게임 프로필에는 일반적으로 다음이 포함됩니다.

- 사용자 이름과 같은 기본 정보
- 아이템 및 장비와 같은 게임 데이터
- 태스크 및 활동과 같은 게임 레코드
- 친구 목록과 같은 소셜 정보

이 애플리케이션의 세분화된 데이터 쿼리 액세스 요구 사항을 충족하기 위해 프라이머리 키(파티션 키 및 정렬 키)는 일반 이름(PK 및 SK)을 사용하므로 아래에서 볼 수 있듯이 다양한 유형의 값으로 오버로드될 수 있습니다.

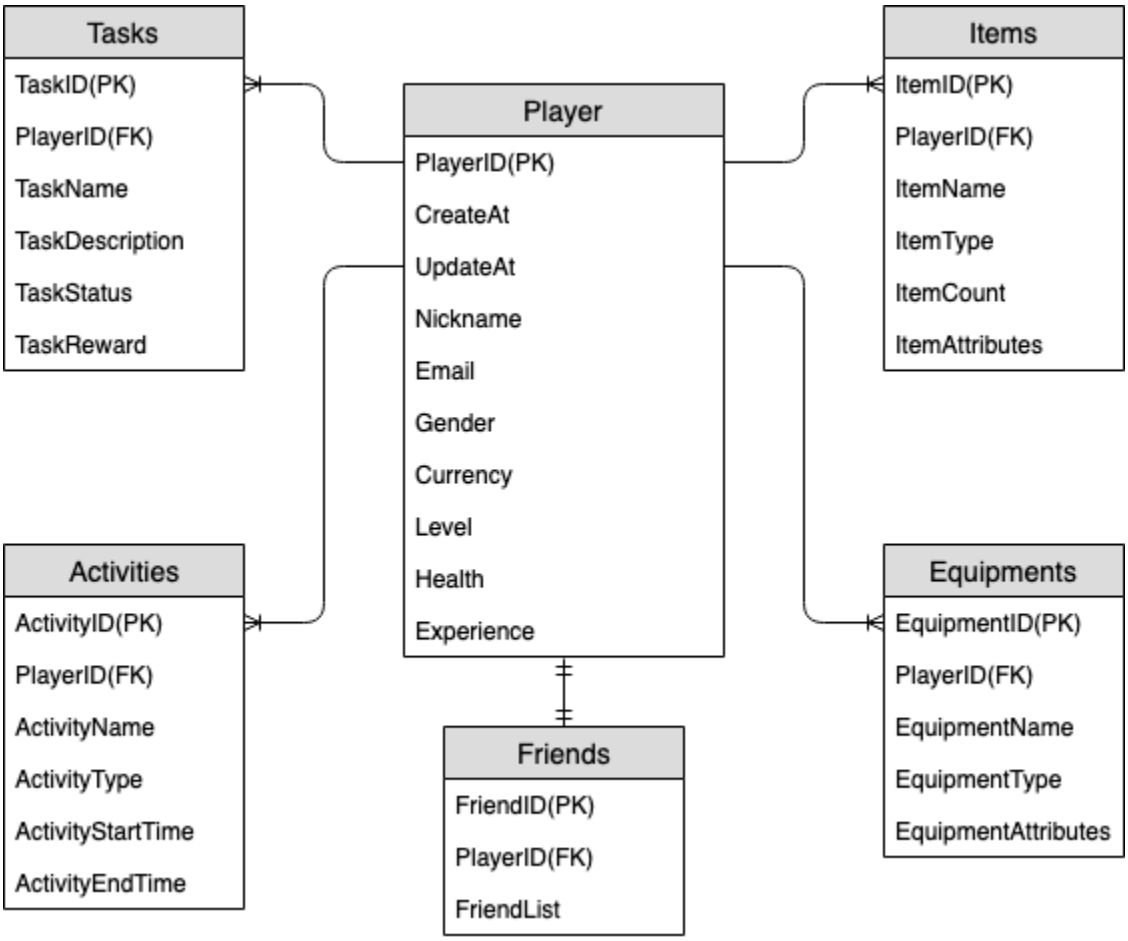
이 스키마 설계의 액세스 패턴은 다음과 같습니다.

- 사용자의 친구 목록 가져오기
- 플레이어의 모든 정보 가져오기
- 사용자의 아이템 목록 가져오기
- 사용자의 아이템 목록에서 특정 아이템 가져오기
- 사용자 캐릭터 업데이트
- 사용자의 아이템 개수 업데이트

게임 프로필의 크기는 게임마다 다릅니다. 큰 속성 값 압축을 통해 DynamoDB의 항목 제한에 맞출 수 있고, 비용을 절감할 수 있습니다. 처리량 관리 전략은 플레이어 수, 초당 플레이한 게임 수, 워크로드의 계절성 같은 다양한 요인에 따라 달라집니다. 일반적으로 새로 출시되는 게임의 경우, 플레이어 수와 인기도를 알 수 없으므로 온디맨드 처리량 모드로 시작하겠습니다.

## 게임 프로필 엔터티 관계 다이어그램

다음은 게임 프로필 스키마 설계에 사용할 엔터티 관계 다이어그램(ERD)입니다.




## 게임 프로필 액세스 패턴

다음은 소셜 네트워크 스키마 설계 시 고려할 액세스 패턴입니다.

- `getPlayerFriends`
- `getPlayerAllProfile`
- `getPlayerAllItems`
- `getPlayerSpecificItem`
- `updateCharacterAttributes`
- `updateItemCount`

## 게임 프로필 스키마 설계 진화

위의 ERD에서 이것이 데이터 모델링의 일대다 관계 유형임을 알 수 있습니다. DynamoDB에서는 일대다 데이터 모델을 항목 컬렉션으로 구성할 수 있는데, 이는 외래 키를 통해 여러 테이블을 생성하고 연결하는 기존 관계형 데이터베이스와는 다릅니다. 항목 컬렉션은 동일한 파티션 키 값을 공유하지만 정렬 키 값이 다른 항목의 그룹입니다. 항목 컬렉션 내에서 각 항목은 다른 항목과 구분되는 고유한 정렬 키 값을 갖습니다. 이 점을 염두에 두고 각 엔터티 유형별로 HASH 값과 RANGE 값에 다음과 같은 패턴을 사용해 보겠습니다.

우선, 모델을 향후에도 사용할 수 있도록 PK 및 SK 같은 일반 이름으로 다양한 유형의 엔터티를 동일한 테이블에 저장합니다. 가독성을 높이기 위해 데이터 유형을 나타내는 접두사를 포함하거나 Entity_type 또는 Type이라는 임의의 속성을 포함할 수 있습니다. 현재 예제에서는 player로 시작하는 문자열을 사용하여 player_ID를 PK로 저장하고, entity name#를 SK의 접두사로 사용하고, 이 데이터가 어떤 엔터티 유형인지 나타내는 Type 속성을 추가합니다. 이를 통해 향후 더 많은 엔터티 유형을 저장할 수 있고, GSI Overloading 및 Sparse GSI 같은 고급 기술을 사용하여 더 많은 액세스 패턴을 충족할 수 있습니다.

액세스 패턴 구현을 시작해 보겠습니다. 플레이어 추가와 장비 추가 같은 액세스 패턴은 PutItem 작업을 통해 구현할 수 있으므로 무시해도 됩니다. 이 문서에서는 위에 나열된 일반적인 액세스 패턴에 초점을 맞추겠습니다.

1단계: 액세스 패턴 1(**getPlayerFriends**) 처리

이 단계로 액세스 패턴 1(getPlayerFriends)을 처리합니다. 현재 설계에서는 우정이 단순하고 게임 내 친구 수가 적습니다. 간단히 하기 위해 목록 데이터 유형을 사용하여 친구 목록을 저장합니다(1:1 모델링). 이 설계에서는 GetItem을 사용하여 이 액세스 패턴을 충족합니다. GetItem 작업에서는 특정 항목을 가져오기 위해 파티션 키 및 정렬 키 값을 명시적으로 제공합니다.

하지만 게임에 많은 수의 친구가 있고 친구 간의 관계가 복잡한 경우(예: 친구 관계가 초대 구성 요소와 수락 구성 요소가 모두 있는 양방향 우정), 친구 목록 크기를 무제한 확장하려면 다대다 관계를 사용하여 각 친구를 개별적으로 저장해야 합니다. 또한 우정을 변경하려면 여러 항목에서 동시에 작업해야 하는 경우, DynamoDB 트랜잭션을 사용하여 여러 작업을 그룹화하여 단일한 전부 아니면 전무 방식 TransactWriteItems 또는 TransactGetItems 작업으로 제출할 수 있습니다.

| Primary key | | Attributes | |
|---|---|---|---|
| Partition key: PK | Sort key: SK | | |
| player001 | FRIENDS#player001 | Type | FriendList |
| | | Friends | [{"M": {"FriendId": {"S":"player002"},"FriendName": {"S":"Alice"}}}, {"M":{"FriendId": {"S":"player003"},"FriendName": {"S":"Bob"}}}] |

2단계: 액세스 패턴 2(**`getPlayerAllProfile`**), 3(**`getPlayerAllItems`**), 4(**`getPlayerSpecificItem`**) 처리

이 단계를 사용하여 액세스 패턴 2(`getPlayerAllProfile`), 3(`getPlayerAllItems`), 4(`getPlayerSpecificItem`)를 처리합니다. 이 세 가지 액세스 패턴에 공통적으로 있는 것은 `Query` 작업을 사용하는 범위 쿼리입니다. 쿼리 범위에 따라 실제 개발에서 일반적으로 사용되는 키 조건 및 필터 식이 사용됩니다.

쿼리 작업에서는 파티션 키에 단일 값을 제공하고 해당 파티션 키 값을 가진 모든 항목을 가져옵니다. 액세스 패턴 2(`getPlayerAllProfile`)는 이 방식으로 구현됩니다. 원하는 경우, 테이블에서 읽을 항목을 결정하는 문자열인 정렬 키 조건식을 추가할 수 있습니다. 액세스 패턴 3(`getPlayerAllItems`)은 `ITEMS#`로 시작하는 정렬 키의 키 조건을 추가하여 구현됩니다. 또한 애플리케이션 측 개발을 단순화하기 위해 필터 식을 사용하여 액세스 패턴 4(`getPlayerSpecificItem`)를 구현할 수 있습니다.

다음은 `Weapon` 카테고리의 항목을 필터링하는 필터 식을 사용하는 의사 코드 예제입니다.

```
filterExpression: "ItemType = :itemType"
expressionAttributeValues: {":itemType": "Weapon"}
```

| Primary key | | Attributes | | | | |
|---|---|---|---|---|---|---|
| **Partition key: PK** | **Sort key: SK** | | | | | |
| player001 | ITEMS#001 | Type | ItemName | ItemType | ItemCount | ItemAttributes |
| | | Item | Health Potion | Consumable | 5 | {"M":{"HP":{"N":"50"}}} |
| | ITEMS#002 | Type | ItemName | ItemType | ItemCount | ItemAttributes |
| | | Item | Armor of the Knight | Armor | 1 | {"M":{"DEF":{"N":"100"}}} |
| | ITEMS#003 | Type | ItemName | ItemType | ItemCount | ItemAttributes |
| | | Item | Sword of the Dragon | Weapon | 1 | {"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}} |

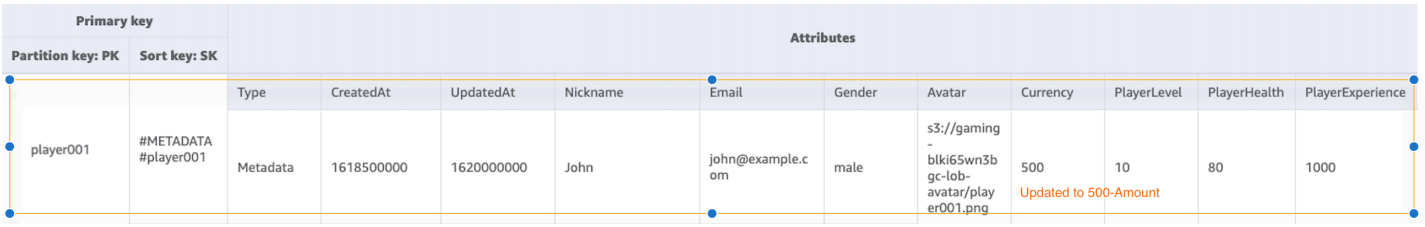
> **Note**
>
> 필터 식은 쿼리가 완료된 후 결과가 클라이언트에 반환되기 전에 적용됩니다. 따라서 쿼리는 필터 식이 있는지 여부에 상관없이 동일한 양의 읽기 용량을 사용합니다.

액세스 패턴이 대규모 데이터 세트를 쿼리하고 대량의 데이터를 필터링하여 작은 데이터 하위 집합만 유지하는 것이라면 DynamoDB 파티션 키와 정렬 키를 보다 효과적으로 설계하는 것이 적절한 접근 방식입니다. 예를 들어 특정 `ItemType`을 얻는 위의 예제에서 각 플레이어마다 항목이 많고 특정 `ItemType`을 쿼리하는 것이 일반적인 액세스 패턴이라면 `ItemType`을 복합 키로 SK에 가져오는 것이 더 효율적입니다. 데이터 모델은 다음과 같습니다. `ITEMS#ItemType#ItemId`

3단계: 액세스 패턴 5(**`updateCharacterAttributes`**) 및 6(**`updateItemCount`**) 처리

이 단계를 사용하여 액세스 패턴 5(`updateCharacterAttributes`)와 6(`updateItemCount`)을 처리합니다. 플레이어가 화폐를 줄이거나 아이템 중 특정 무기 수량을 수정하는 등 캐릭터를 수정해야 할 경우, `UpdateItem`을 사용하여 이러한 액세스 패턴을 구현합니다. 플레이어의 화폐를 업데이트하되 최소 금액 이하로 내려가지 않도록 하기 위해 최소 금액 이상일 경우에만 잔액을 줄일 수 있는 the section called “CLI 예제”을 추가할 수 있습니다. 다음은 의사 코드 예제입니다.

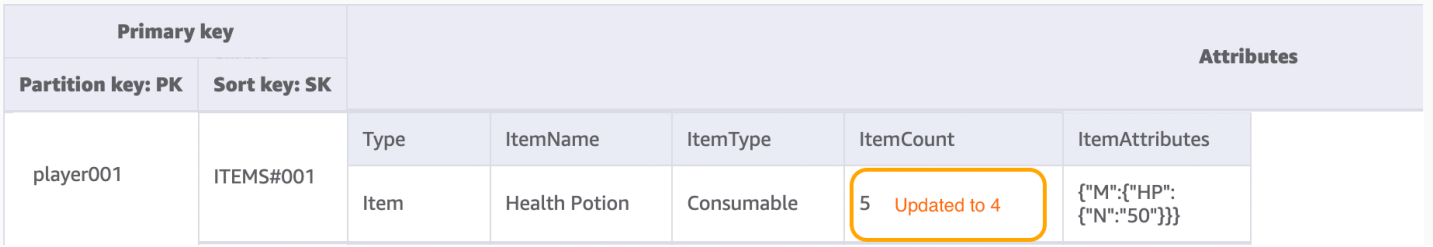
```
UpdateExpression: "SET currency = currency - :amount"
ConditionExpression: "currency >= :minAmount"
```



| Primary key | | Attributes | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | Type | CreatedAt | UpdatedAt | Nickname | Email | Gender | Avatar | Currency | PlayerLevel | PlayerHealth | PlayerExperience |
| player001 | #METADATA #player001 | Metadata | 1618500000 | 1620000000 | John | john@example.com | male | s3://gaming-blki65wn3bgc-lob-avatar/player001.png | 500 Updated to 500-Amount | 10 | 80 | 1000 |

DynamoDB로 개발하고 Atomic Counters를 사용하여 인벤토리를 줄이는 경우, 낙관적 잠금을 사용하여 멱등성을 보장할 수 있습니다. 다음은 Atomic Counters의 의사 코드 예제입니다.

```
UpdateExpression: "SET ItemCount = ItemCount - :incr"
expression-attribute-values: '{":incr":{"N":"1"}}'
```



| Primary key | | Attributes | | | | |
|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | Type | ItemName | ItemType | ItemCount | ItemAttributes |
| player001 | ITEMS#001 | Item | Health Potion | Consumable | 5 Updated to 4 | {"M":{"HP":{"N":"50"}}} |

또한 플레이어가 화폐로 아이템을 구매하는 시나리오에서는 전체 프로세스가 통화를 차감하는 동시에 아이템을 추가해야 합니다. DynamoDB Transactions를 사용하여 여러 작업을 그룹화하고 이를 하나의 전부 아니면 전무 `TransactWriteItems` 또는 `TransactGetItems` 작업으로 제출할 수 있습니다. `TransactWriteItems`는 하나의 전부 아니면 전무 작업에서 최대 100개의 쓰기 작업을 그룹화하는 동기식 및 멱등성 쓰기 작업입니다. 작업은 원자 단위로 완료되므로 작업이 모두 성공하거나 모두 실패하게 됩니다. 트랜잭션은 화폐 중복 또는 소멸의 위험을 없애는 데 도움이 됩니다. 트랜잭션에 대한 자세한 내용은 DynamoDB Transactions 예 섹션을 참조세요.

모든 액세스 패턴과 스키마 설계에서 이를 처리하는 방법이 아래 표에 요약되어 있습니다.

| 액세스 패턴 | 기본 테이블/ GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 | 기타 조건/필터 |
|---|---|---|---|---|---|
| getPlayer Friends | 기본 테이블 | GetItem | PK=PlayerID | SK=“FRIENDS#playerID” | |
| getPlayer AllProfile | 기본 테이블 | 쿼리 | PK=PlayerID | | |

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 | 기타 조건/필터 |
|---|---|---|---|---|---|
| getPlayerAllItems | 기본 테이블 | 쿼리 | PK=PlayerID | SK begins_with “ITEMS#” | |
| getPlayerSpecificItem | 기본 테이블 | 쿼리 | PK=PlayerID | SK begins_with “ITEMS#” | filterExpression: "ItemType = :itemType" expressionAttributeValues: { ":itemType": "Weapon" } |
| updateCharacterAttributes | 기본 테이블 | UpdateItem | PK=PlayerID | SK=“#METADATA#playerID” | UpdateExpression: "SET currency = currency - :amount" ConditionExpression: "currency >= :minAmount" |

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 | 기타 조건/필터 |
|---|---|---|---|---|---|
| updateItemCount | 기본 테이블 | UpdateItem | PK=PlayerID | SK =“ITEMS#ItemID” | update-expression: "SET ItemCount = ItemCount - :incr" expression-attribute-values : '{":incr": {"N":"1"}}' |

## 게임 프로필 최종 스키마

다음은 최종 스키마 설계입니다. 이 스키마 설계를 JSON 파일로 다운로드하려면 GitHub의 DynamoDB 예제를 참조하세요.

기본 테이블:

| Primary key | | Attributes | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | | | | | | | | |
| player001 | #METADATA#player001 | Type | CreatedAt | UpdatedAt | Nickname | Email | Gender | Avatar | Currency | PlayerLevel | PlayerHealth | PlayerExperience |
| | | Metadata | 1618500000 | 1620000000 | John | john@example.com | male | s3://gaming-blki65wn3bgc-lob-avatar/player001.png | 500 | 10 | 80 | 1000 |
| | ACTIVITY#001 | Type | ActivityEndTime | ActivityName | ActivityReward | ActivityStartTime | ActivityType | | | | | |
| | | Activity | 1647475199 | Hunting Trip | {"M":{"Gold":{"N":"50"},"XP":{"N":"200"}}} | 1647388800 | Hunting | | | | | |
| | ACTIVITY#002 | Type | ActivityEndTime | ActivityName | ActivityReward | ActivityStartTime | ActivityType | | | | | |
| | | Activity | 1647647999 | Mining Adventure | {"M":{"Gold":{"N":"1000"},"XP":{"N":"500"}}} | 1647561600 | Mining | | | | | |
| | ACTIVITY#003 | Type | ActivityEndTime | ActivityName | ActivityReward | ActivityStartTime | ActivityType | | | | | |
| | | Activity | 1647820799 | Arena Challenge | {"M":{"Gold":{"N":"2000"},"XP":{"N":"1000"}}} | 1647734400 | Arena | | | | | |
| | EQUIPMENTS#001 | Type | EquipmentName | EquipmentType | EquipmentAttributes | | | | | | | |
| | | Equipment | Sword of the Dragon | Weapon | {"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}} | | | | | | | |
| | EQUIPMENTS#001EQUIPMENTS#002 | Type | EquipmentName | EquipmentType | EquipmentAttributes | | | | | | | |
| | | Equipment | Armor of the Knight | Armor | {"M":{"DEF":{"N":"100"}}} | | | | | | | |
| | EQUIPMENTS#003 | Type | EquipmentName | EquipmentType | EquipmentAttributes | | | | | | | |
| | | Equipment | Ring of the Mage | Accessory | {"M":{"SP":{"N":"50"}}} | | | | | | | |
| | FRIENDS#player001 | Type | FriendList | | | | | | | | | |
| | | Friends | [{"M":{"FriendId":{"S":"player002"},"FriendName":{"S":"Alice"}}},{"M":{"FriendId":{"S":"player003"},"FriendName":{"S":"Bob"}}}] | | | | | | | | | |
| | ITEMS#001 | Type | ItemName | ItemType | ItemCount | ItemAttributes | | | | | | |
| | | Item | Health Potion | Consumable | 5 | {"M":{"HP":{"N":"50"}}} | | | | | | |
| | ITEMS#002 | Type | ItemName | ItemType | ItemCount | ItemAttributes | | | | | | |
| | | Item | Armor of the Knight | Armor | 1 | {"M":{"DEF":{"N":"100"}}} | | | | | | |
| | ITEMS#003 | Type | ItemName | ItemType | ItemCount | ItemAttributes | | | | | | |
| | | Item | Sword of the Dragon | Weapon | 1 | {"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}} | | | | | | |
| | TASK#001 | Type | TaskName | TaskDescription | TaskStatus | TaskReward | | | | | | |
| | | Task | Find the Lost TreasureFind the Lost Treasure | Get clues from a lost adventurer and find the lost treasure. | InProgress | {"M":{"Gold":{"N":"100"},"XP":{"N":"50"}}} | | | | | | |
| | TASK#002 | Type | TaskName | TaskDescription | TaskStatus | TaskReward | | | | | | |
| | | Task | Defeat Magic Monsters | Go to the Magic Forest and defeat three magic monsters. | Completed | {"M":{"Gold":{"N":"200"},"XP":{"N":"100"}}} | | | | | | |
| | TASK#003 | Type | TaskName | TaskDescription | TaskStatus | TaskReward | | | | | | |
| | | Task | Rescue the Princess | Go to the Demon King's Castle and rescue the princess who is being held captive by the Demon King. | Available | {"M":{"Gold":{"N":"500"},"XP":{"N":"200"}}} | | | | | | |

### 이 스키마 설계와 함께 NoSQL Workbench 사용

이 최종 스키마를 DynamoDB 데이터 모델링, 데이터 시각화, 쿼리 개발 기능을 제공하는 시각적 도구인 NoSQL Workbench로 가져와서 새 프로젝트를 추가로 탐색하고 편집할 수 있습니다. 시작하려면 다음 단계를 따릅니다.

1. NoSQL Workbench 다운로드 자세한 내용은 the section called “다운로드” 단원을 참조하십시오.
2. 위에 나열된 JSON 스키마 파일을 다운로드합니다. 이 파일은 이미 NoSQL Workbench 모델 형식으로 되어 있습니다.
3. JSON 스키마 파일을 NoSQL Workbench로 가져옵니다. 자세한 내용은 the section called “기존 모델 가져오기” 단원을 참조하십시오.
4. NOSQL Workbench로 가져온 후 데이터 모델을 편집할 수 있습니다. 자세한 내용은 the section called “기존 모델 편집” 단원을 참조하십시오.
5. 데이터 모델을 시각화하거나, 샘플 데이터를 추가하거나, CSV 파일에서 샘플 데이터를 가져오려면 NoSQL Workbench의 Data Visualizer 기능을 사용하세요.

## DynamoDB에서 불만 관리 시스템 스키마 설계

### 불만 관리 시스템 비즈니스 사용 사례

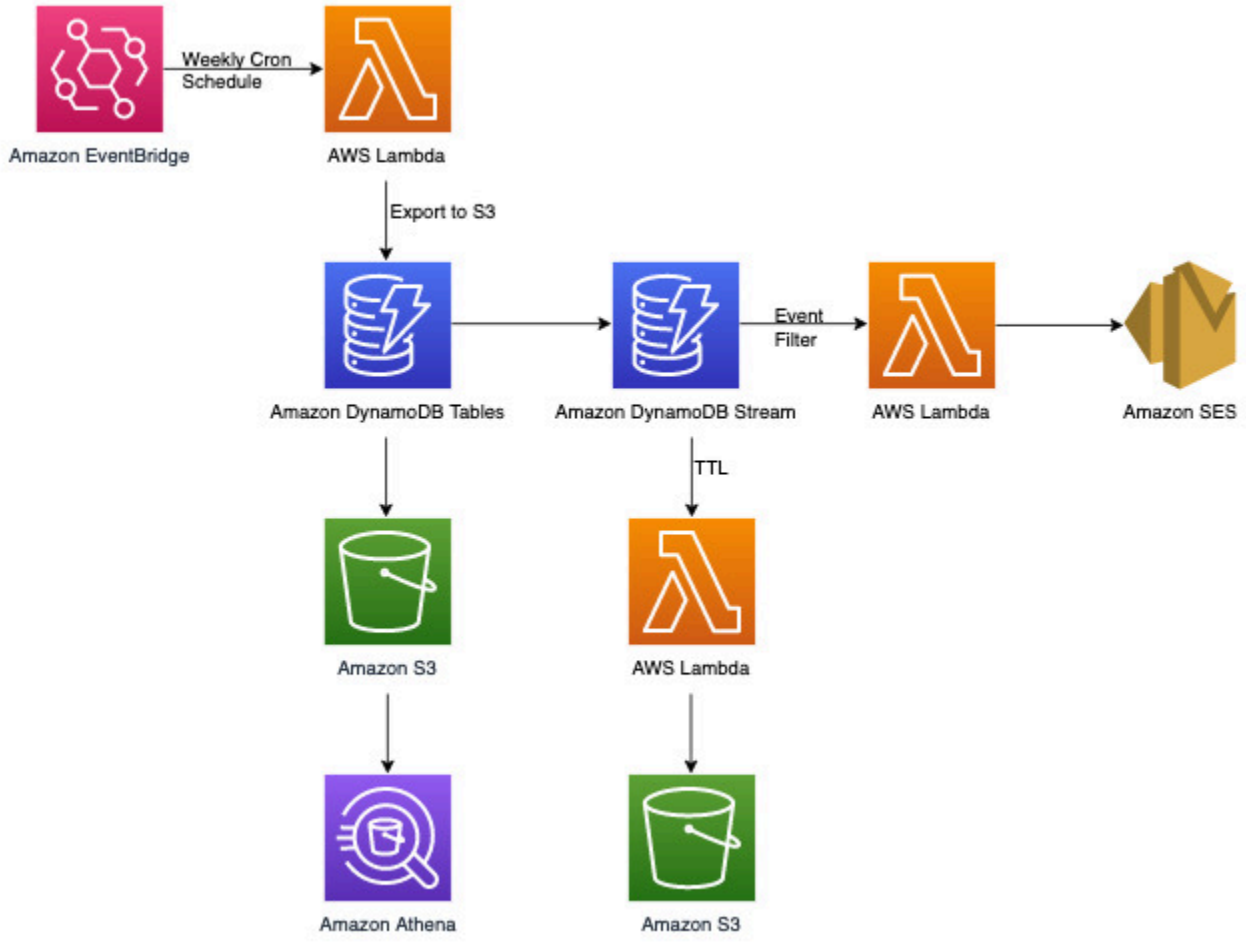
DynamoDB는 불만 관리 시스템(또는 콜센터) 사용 사례에 적합한 데이터베이스입니다. 관련된 대부분의 액세스 패턴이 키-값 기반 트랜잭션 조회이기 때문입니다. 이 시나리오의 일반적인 액세스 패턴은 다음과 같습니다.

- 불만 사항 생성 및 업데이트
- 불만 사항 에스컬레이션
- 불만 사항에 대한 의견 작성 및 읽기
- 고객의 모든 불만 사항 접수
- 에이전트의 모든 의견 가져오기 및 모든 에스컬레이션 가져오기

일부 의견에는 불만 사항 또는 해결 방법을 설명하는 첨부 파일이 있을 수 있습니다. 이러한 패턴은 모두 키-값 액세스 패턴이지만 불만 사항에 새 의견이 추가될 때 알림을 보내거나 분석 쿼리를 실행하여 주간 심각도(또는 에이전트 성과)별로 불만 사항 분포를 확인하는 등의 추가 요구 사항이 있을 수 있습니다. 수명 주기 관리 또는 규정 준수와 관련된 추가 요구 사항은 불만 사항을 기록한 지 3년이 지난 후에 불만 사항 데이터를 아카이빙하는 것입니다.

## 불만 관리 시스템 아키텍처 다이어그램

다음 다이어그램은 불만 관리 시스템의 아키텍처 다이어그램입니다. 이 다이어그램은 불만 관리 시스템에서 사용하는 다양한 AWS 서비스 통합을 보여니다.




나중에 DynamoDB 데이터 모델링 섹션에서 다루게 될 키-값 트랜잭션 액세스 패턴 외에도 세 가지 비트랜잭션 요구 사항이 있습니다. 위의 아키텍처 다이어그램은 다음 세 가지 워크플로로 나눌 수 있습니다.

1. 불만 사항에 새 의견이 추가되면 알림 보내기
2. 주간 데이터에 대한 분석 쿼리 실행
3. 3년 이상 경과된 데이터 아카이빙

각각에 대해 좀더 자세히 살펴보겠습니다.

불만 사항에 새 의견이 추가되면 알림 보내기

아래 워크플로를 사용하여 이 요구 사항을 달성할 수 있습니다.




DynamoDB Streams는 DynamoDB 테이블에 대한 모든 쓰기 작업을 기록하는 변경 데이터 캡처 메커니즘입니다. 이러한 변경 사항의 일부 또는 전부를 트리거하도록 Lambda 함수를 구성할 수 있습니다. 사용 사례와 관련이 없는 이벤트를 필터링하도록 Lambda 트리거에 이벤트 필터를 구성할 수 있습니다. 이 경우 필터를 사용하여 새 의견이 추가된 경우에만 Lambda를 트리거하고 AWS Secrets Manager 또는 기타 보안 인증 정보 스토어에서 가져올 수 있는 관련 이메일 ID로 알림을 보낼 수 있습니다.

주간 데이터에 대한 분석 쿼리 실행

DynamoDB는 주로 온라인 트랜잭션 처리(OLTP)에 중점을 둔 워크로드에 적합합니다. 분석 요구 사항이 있는 다른 10%~20% 액세스 패턴의 경우, DynamoDB 테이블의 실시간 트래픽에 영향을 주지 않고 관리형 Amazon S3로 내보내기 기능을 사용하여 데이터를 S3으로 내보낼 수 있습니다. 아래에서 이 워크플로를 살펴보세요.




Amazon EventBridge를 사용하면 일정에 따라 AWS Lambda를 트리거할 수 있습니다. 이를 통해 Lambda 호출이 주기적으로 발생하도록 cron 표현식을 구성할 수 있습니다. Lambda는 `ExportToS3` API를 호출하고 DynamoDB 데이터를 S3에 저장할 수 있습니다. 그런 다음 Amazon Athena와 같은 SQL 엔진에서 이 S3 데이터에 액세스하여 테이블의 실시간 트랜잭션 워크로드에 영향

을 주지 않고 DynamoDB 데이터에 대한 분석 쿼리를 실행할 수 있습니다. 심각도 수준별 불만 사항 수를 확인하기 위한 샘플 Athena 쿼리는 다음과 같습니다.

```
SELECT Item.severity.S as "Severity", COUNT(Item) as "Count"
FROM "complaint_management"."data"
WHERE NOT Item.severity.S = ''
GROUP BY Item.severity.S ;
```

이 Athena 쿼리는 다음 결과를 반환합니다.

**Results** (3)

Search rows

| # | Severity | Count |
|---|---|---|
| 1 | P3 | 1 |
| 2 | P2 | 2 |
| 3 | P1 | 1 |

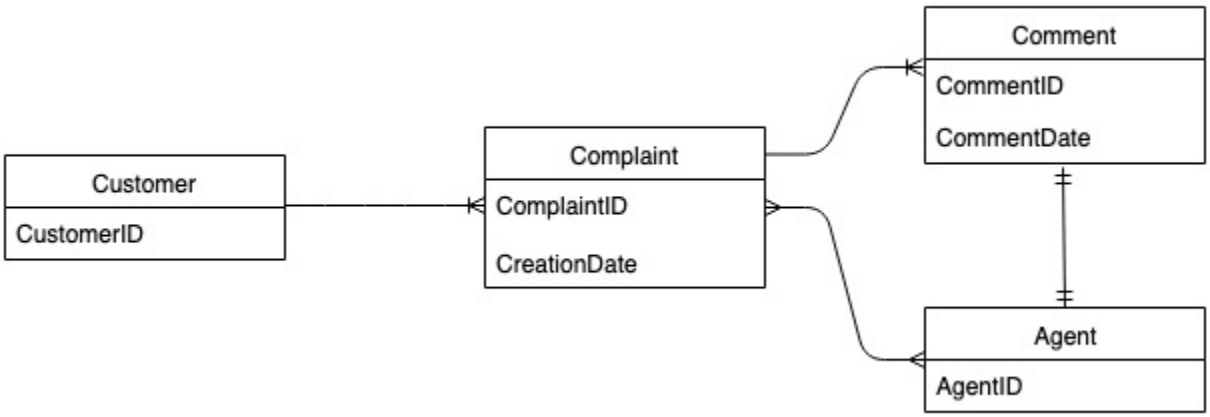
3년 이상 경과된 데이터 아카이빙

DynamoDB TTL(Time to Live) 기능을 활용하면 추가 비용 없이 DynamoDB 테이블에서 사용되지 않는 데이터를 삭제할 수 있습니다(다른 리전에 복제된 TTL 삭제가 쓰기 용량을 소비하는 2019.11.21(현재) 버전의 글로벌 테이블 복제본인 경우 제외). 이 데이터는 표시되며 DynamoDB Streams에서 소비되어 Amazon S3에 아카이빙될 수 있습니다. 이 요구 사항에 대한 워크플로는 다음과 같습니다.




## 불만 관리 시스템 엔터티 관계 다이어그램

다음은 불만 관리 스키마 설계에 사용할 엔터티 관계 다이어그램(ERD)입니다.




## 불만 관리 시스템 액세스 패턴

다음은 불만 관리 스키마를 설계할 때 고려할 액세스 패턴입니다.

1. createComplaint
2. updateComplaint
3. updateSeveritybyComplaintID
4. getComplaintByComplaintID
5. addCommentByComplaintID
6. getAllCommentsByComplaintID
7. getLatestCommentByComplaintID
8. getAComplaintbyCustomerIDAndComplaintID
9. getAllComplaintsByCustomerID
10. escalateComplaintByComplaintID
11. getAllEscalatedComplaints
12. getEscalatedComplaintsByAgentID(최신 것부터 가장 오래된 것 순)
13. getCommentsByAgentID(두 날짜 사이)

## 불만 관리 시스템 스키마 설계 진화

이는 불만 관리 시스템이므로 대부분의 액세스 패턴에서 불만 사항이 기본 엔터티입니다.
`ComplaintID`는 카디널리티가 높을 때 기본 파티션에 데이터가 균일하게 배포되도록 하며 식별된 액

세스 패턴에 대한 가장 일반적인 검색 기준이기도 합니다. 따라서 ComplaintID는 이 데이터 세트에서 좋은 파티션 키 후보입니다.

1단계: 액세스 패턴 1(**createComplaint**), 2(**updateComplaint**), 3(**updateSeveritybyComplaintID**), 4(**getComplaintByComplaintID**) 처리

'metadata'(또는 'AA')라는 일반 정렬 키 값을 사용하여 CustomerID, State, Severity, CreationDate와 같은 불만 사항별 정보를 저장할 수 있습니다. PK=ComplaintID 및 SK="metadata"인 싱글톤 작업을 사용하여 다음을 수행합니다.

1. PutItem을 통해 새 불만 사항 생성
2. UpdateItem을 통해 불만 사항 메타데이터의 심각도 또는 기타 필드 업데이트
3. GetItem을 통해 불만 사항에 대한 메타데이터 가져오기

| Primary key | | Attributes | | | | |
|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | | |
| Complaint1321 | metadata | customer_id | current_state | creation_time | severity | complaint_description |
| | | custXYZ | assigned | 2023-05-10T15:58:00 | P2 | <Complaint Description> |

2단계: 액세스 패턴 5(**addCommentByComplaintID**) 처리

이 액세스 패턴에는 불만 사항과 불만 사항에 대한 의견 간의 일대다 관계 모델이 필요합니다. 여기서는 수직 파티셔닝 기법을 통해 정렬 키를 사용하며 다양한 유형의 데이터로 항목 컬렉션을 생성합니다. 액세스 패턴 6(getAllCommentsByComplaintID) 및 7(getLatestCommentByComplaintID)을 살펴보면 의견을 시간별로 정렬해야 한다는 것을 알 수 있습니다. 또한 여러 개의 의견이 동시에 들어올 수 있으므로 복합 정렬 키 기법을 사용하여 정렬 키 속성에 time과 CommentID를 추가할 수 있습니다.

이러한 의견 충돌 가능성에 대처하기 위한 다른 옵션은 타임스탬프의 세분성을 높이거나 Comment_ID를 사용하는 대신 증분 숫자를 접미사로 추가하는 것입니다. 이 경우 범위 기반 작업을 활성화하기 위해 의견에 해당하는 항목의 정렬 키 값 앞에 'comm#'을 붙입니다.

또한 불만 사항 메타데이터의 currentState가 새 의견이 추가될 때의 상태를 반영하는지 확인해야 합니다. 의견을 추가하면 불만 사항이 에이전트에게 배정되었거나 해결되었다는 등의 메시지가 표시될 수 있습니다. 의견 추가 및 불만 사항 메타데이터의 현재 상태 업데이트를 전부 아니면 전무 방식으로 묶기 위해 TransactWriteItems API를 사용합니다. 이제 결과 테이블 상태는 다음과 같습니다.

| Primary key | | Attributes | | | | |
|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | | |
| Complaint1321 | comm#2023-05-10T16:00:00#comm3 | comm_id | comm_date | complaint_state | comm_text | agentID |
| | | comm3 | 2023-05-10T16:00:00 | investigating | <Comment text> | AgentB |
| | metadata | customer_id | current_state | creation_time | severity | complaint_description |
| | | custXYZ | investigating | 2023-05-10T15:58:00 | P2 | <Complaint Description> |

테이블에 더 많은 데이터를 추가하고 `ComplaintID`에 대한 추가 인덱스가 필요한 경우 모델의 미래 유용성을 높이기 위해 `ComplaintID`를 PK와 별도의 필드로 추가해 보겠습니다. 또한 일부 의견에는 첨부 파일이 있을 수 있습니다. 이 첨부 파일은 Amazon Simple Storage Service에 저장하고 해당 참조 또는 URL만 DynamoDB에 보관합니다. 비용과 성능을 최적화하기 위해 트랜잭션 데이터베이스를 최대한 간결하게 유지하는 것이 좋습니다. 데이터는 이제 다음과 같습니다.

| Primary key | | Attributes | | | | | |
|---|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | | | |
| Complaint123 | comm#2023-04-30T12:00:24#comm1 | comm_id | comm_date | complaint_state | comm_text | agentID | |
| | | comm1 | 2023-04-30T12:00:24 | investigating | <comm text> | AgentA | |
| | comm#2023-04-30T12:35:54#comm2 | comm_id | comm_date | complaint_state | comm_text | attachments | agentID |
| | | comm2 | 2023-04-30T12:35:54 | resolved | <comm text> | ["s3://URL_for_attachment1","s3://URL_for_attachment2"] | AgentA |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | custABC | Complaint123 | resolved | 2023-04-30T12:00:00 | P2 | <description text> |
| Complaint1321 | comm#2023-05-10T16:00:00#comm3 | comm_id | comm_date | complaint_state | comm_text | agentID | |
| | | comm3 | 2023-05-10T16:00:00 | investigating | <comm text> | AgentB | |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | custXYZ | Complaint1321 | investigating | 2023-05-10T15:58:00 | P2 | <descr_text> |

### 3단계: 액세스 패턴 6(**`getAllCommentsByComplaintID`**) 및 7(**`getLatestCommentByComplaintID`**) 처리

불만 사항에 대한 모든 의견을 가져오려면 정렬 키의 query 조건과 함께 `begins_with` 작업을 사용할 수 있습니다. 이와 같은 정렬 키 조건을 사용하면 메타데이터 항목을 읽기 위해 읽기 용량을 추가로 소비하고 관련 결과를 필터링하는 오버헤드가 발생하는 대신 필요한 항목만 읽을 수 있습니다. 예를 들

어, PK=Complaint123 및 SK begins_with comm#을 사용한 쿼리 작업은 메타데이터 항목을 건너뛰는 동안 다음을 반환합니다.

| Primary key | | Attributes | | | | | |
|---|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | | | |
| Complaint123 | comm#2023-04-30T12:00:24#comm1 | comm_id | comm_date | complaint_state | comm_text | agentID | |
| | | comm1 | 2023-04-30T12:00:24 | investigating | <comm text> | AgentA | |
| | comm#2023-04-30T12:35:54#comm2 | comm_id | comm_date | complaint_state | comm_text | attachments | agentID |
| | | comm2 | 2023-04-30T12:35:54 | resolved | <comm text> | ["s3://URL_for_attachment1","s3://URL_for_attachment2"] | AgentA |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | custABC | Complaint123 | resolved | 2023-04-30T12:00:00 | P2 | <description text> |
| Complaint1321 | comm#2023-05-10T16:00:00#comm3 | comm_id | comm_date | complaint_state | comm_text | agentID | |
| | | comm3 | 2023-05-10T16:00:00 | investigating | <comm text> | AgentB | |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | custXYZ | Complaint1321 | investigating | 2023-05-10T15:58:00 | P2 | <descr_text> |

패턴 7(getLatestCommentByComplaintID)의 불만 사항에 대한 최신 의견이 필요하므로 두 개의 추가 쿼리 파라미터를 사용하겠습니다.

1. 결과를 내림차순으로 정렬하려면 ScanIndexForward를 False로 설정해야 합니다.
2. 최신(단 하나) 의견을 가져오려면 Limit를 1로 설정해야 합니다.

액세스 패턴 6(getAllCommentsByComplaintID)과 유사하게 begins_with comm#을 정렬 키 조건으로 사용하여 메타데이터 항목을 건너뜁니다. 이제 PK=Complaint123, SK=begins_with comm#, ScanIndexForward=False, Limit 1과 함께 쿼리 작업을 사용하여 이 설계에서 액세스 패턴 7을 수행할 수 있습니다. 대상으로 지정된 다음 항목이 결과로 반환됩니다.

| Partition key: PK | Sort key: SK | Attributes | | | | | |
|---|---|---|---|---|---|---|---|
| | comm#2023-04-30T12:00:24#comm1 | comm_id | comm_date | complaint_state | comm_text | agentID | |
| | | comm1 | 2023-04-30T12:00:24 | investigating | <comm text> | AgentA | |
| Complaint123 | comm#2023-04-30T12:35:54#comm2 | comm_id | comm_date | complaint_state | comm_text | attachments | agentID |
| | | comm2 | 2023-04-30T12:35:54 | resolved | <comm text> | ["s3://URL_for_attachment1","s3://URL_for_attachment2"] | AgentA |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | custABC | Complaint123 | resolved | 2023-04-30T12:00:00 | P2 | <description text> |
| Complaint1321 | comm#2023-05-10T16:00:00#comm3 | comm_id | comm_date | complaint_state | comm_text | agentID | |
| | | comm3 | 2023-05-10T16:00:00 | investigating | <comm text> | AgentB | |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | custXYZ | Complaint1321 | investigating | 2023-05-10T15:58:00 | P2 | <descr_text> |

테이블에 더미 데이터를 더 추가해 보겠습니다.

| Primary key | | Attributes | | | | | |
|---|---|---|---|---|---|---|---|
| **Partition key: PK** | **Sort key: SK** | | | | | | |
| Complaint123 | comm#2023-04-30T12:00:24#comm1 | comm_id | comm_date | complaint_state | comm_text | agentID | |
| | | comm1 | 2023-04-30T12:00:24 | investigating | <comm text> | AgentA | |
| | comm#2023-04-30T12:35:54#comm2 | comm_id | comm_date | complaint_state | comm_text | attachments | agentID |
| | | comm2 | 2023-04-30T12:35:54 | resolved | <comm text> | ["s3://URL_for_attachment1","s3://URL_for_attachment2"] | AgentA |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | custABC | Complaint123 | resolved | 2023-04-30T12:00:00 | P2 | <description text> |
| Complaint1321 | comm#2023-05-10T16:00:00#comm3 | comm_id | comm_date | complaint_state | comm_text | agentID | |
| | | comm3 | 2023-05-10T16:00:00 | investigating | <comm text> | AgentB | |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | custXYZ | Complaint1321 | investigating | 2023-05-10T15:58:00 | P2 | <descr_text> |
| Complaint1444 | comm#2022-12-31T19:32:00#comm4 | comm_id | comm_date | complaint_state | comm_text | | |
| | | comm4 | 2022-12-31T19:32:00 | waiting | <comm text> | | |
| | comm#2022-12-31T19:40:00#comm5 | comm_id | comm_date | complaint_state | comm_text | attachments | agentID |
| | | comm5 | 2022-12-31T19:40:00 | assigned | <comm text> | ["s3://URL_for_attachment1"] | AgentC |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | custXY32 | Complaint1444 | assigned | 2022-12-31T19:39:57 | P1 | <description text> |
| Complaint0987 | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | custXYZ | Complaint0987 | assigned | 2023-06-10T12:30:08 | P3 | <description text> |

### 4단계: 액세스 패턴 8(**getAComplaintbyCustomerIDAndComplaintID**) 및 9(**getAllComplaintsByCustomerID**) 처리

액세스 패턴 8(`getAComplaintbyCustomerIDAndComplaintID`) 및 9(`getAllComplaintsByCustomerID`)에는 `CustomerID`라는 새로운 검색 기준이 도입되었습니다. 기존 테이블에서 가져오려면 모든 데이터를 읽고 해당 `CustomerID`에 대한 관련 항목을 필터링하기 위해 비용이 많이 드는 Scan이 필요합니다. `CustomerID`를 파티션 키로 사용하여 글로벌 보조 인덱

스(GSI)를 생성하면 이 검색을 더욱 효율적으로 만들 수 있습니다. 고객과 불만 사항 간의 일대다 관계와 액세스 패턴 9(`getAllComplaintsByCustomerID`)을 염두에 두면 `ComplaintID`가 정렬 키에 적합한 후보가 됩니다.

GSI의 데이터는 다음과 같습니다.

| Primary key | | Attributes | | | | | |
|---|---|---|---|---|---|---|---|
| **Partition key: customer_id** | **Sort key: complaint_id** | | | | | | |
| custABC | Complaint123 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint123 | metadata | resolved | 2023-04-30T12:00:00 | P2 | <description text> |
| custXY32 | Complaint1444 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint1444 | metadata | assigned | 2022-12-31T19:39:57 | P1 | <description text> |
| custXYZ | Complaint0987 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint0987 | metadata | assigned | 2023-06-10T12:30:08 | P3 | <description text> |
| | Complaint1321 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint1321 | metadata | investigating | 2023-05-10T15:58:00 | P2 | <descr_text> |

액세스 패턴 8(`getAComplaintbyCustomerIDAndComplaintID`)에 대한 이 GSI의 쿼리 예시는 `customer_id=custXYZ`, `sort key=Complaint1321`입니다. 결과는 다음과 같습니다.

| Primary key | | Attributes | | | | | |
|---|---|---|---|---|---|---|---|
| **Partition key: customer_id** | **Sort key: complaint_id** | | | | | | |
| custABC | Complaint123 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint123 | metadata | resolved | 2023-04-30T12:00:00 | P2 | <description text> |
| custXY32 | Complaint1444 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint1444 | metadata | assigned | 2022-12-31T19:39:57 | P1 | <description text> |
| custXYZ | Complaint0987 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint0987 | metadata | assigned | 2023-06-10T12:30:08 | P3 | <description text> |
| | Complaint1321 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint1321 | metadata | investigating | 2023-05-10T15:58:00 | P2 | <descr_text> |

액세스 패턴 9(`getAllComplaintsByCustomerID`)에 대한 고객의 모든 불만 사항을 가져오려면 GSI에 대한 쿼리는 파티션 키 조건으로 `customer_id=custXYZ`입니다. 결과는 다음과 같습니다.

| Primary key | | Attributes | | | | | |
|---|---|---|---|---|---|---|---|
| **Partition key: customer_id** | **Sort key: complaint_id** | | | | | | |
| custABC | Complaint123 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint123 | metadata | resolved | 2023-04-30T12:00:00 | P2 | <description text> |
| custXY32 | Complaint1444 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint1444 | metadata | assigned | 2022-12-31T19:39:57 | P1 | <description text> |
| custXYZ | Complaint0987 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint0987 | metadata | assigned | 2023-06-10T12:30:08 | P3 | <description text> |
| | Complaint1321 | PK | SK | current_state | creation_time | severity | complaint_description |
| | | Complaint1321 | metadata | investigating | 2023-05-10T15:58:00 | P2 | <descr_text> |

5단계: 액세스 패턴 10(**`escalateComplaintByComplaintID`**) 처리

이 액세스에는 에스컬레이션 측면이 도입됩니다. 불만 사항을 에스컬레이션하기 위해 `UpdateItem`을 사용하여 `escalated_to` 및 `escalation_time`과 같은 속성을 기존 불만 사항 메타데이터 항목에 추가할 수 있습니다. DynamoDB는 유연한 스키마 설계를 제공하므로 키가 아닌 속성 세트가 여러 항목 간에 균일하거나 이산적일 수 있습니다. 아래 예를 참조하세요.

`UpdateItem with PK=Complaint1444, SK=metadata`

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Complaint1444 | comm#2022-12-31T19:32:00#comm4 | comm_id | comm_date | complaint_state | comm_text | | | |
| | | comm4 | 2022-12-31T19:32:00 | waiting | <comm text> | | | |
| | comm#2022-12-31T19:40:00#comm5 | comm_id | comm_date | complaint_state | comm_text | attachments | agentID | |
| | | comm5 | 2022-12-31T19:40:00 | assigned | <comm text> | ["s3://URL_for_attachment1"] | AgentC | |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description | escalated_to | escalation_time |
| | | custXY32 | Complaint1444 | assigned | 2022-12-31T19:39:57 | P1 | <description text> | AgentB | 2023-01-03T04:00:07 |

6단계: 액세스 패턴 11(**`getAllEscalatedComplaints`**) 및 12(**`getEscalatedComplaintsByAgentID`**) 처리

전체 데이터 세트에서 소수의 불만 사항만 에스컬레이션될 것으로 예상됩니다. 따라서 에스컬레이션 관련 속성에 대한 인덱스를 생성하면 효율적인 조회는 물론 비용 효율적인 GSI 저장이 가능합니다. 이를 위해 희소 인덱스 기법을 활용할 수 있습니다. 파티션 키가 `escalated_to`이고 정렬 키가 `escalation_time`인 GSI는 다음과 같습니다.

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Partition key: escalated_to | Sort key: escalation_time | | | | | | | | |
| AgentB | 2023-01-03T04:00:07 | PK | SK | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | Complaint1444 | metadata | custXY32 | Complaint1444 | assigned | 2022-12-31T19:39:57 | P1 | <description text> |
| | 2023-05-15T14:00:00 | PK | SK | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | Complaint1321 | metadata | custXYZ | Complaint1321 | investigating | 2023-05-10T15:58:00 | P2 | <descr_text> |

액세스 패턴 11(`getAllEscalatedComplaints`)에 대해 에스컬레이션된 모든 불만 사항을 가져오려면 이 GSI를 스캔하면 됩니다. 이 스캔은 GSI의 크기 때문에 성능과 비용 효율성이 뛰어납니다. 특정 에이전트에 대해 에스컬레이션된 불만 사항을 가져오려면(액세스 패턴 12(`getEscalatedComplaintsByAgentID`)) 파티션 키를 `escalated_to=agentID`로 지정하고 `ScanIndexForward`를 `False`로 설정하여 최신 항목부터 오래된 항목 순으로 정렬합니다.

7단계: 액세스 패턴 13(**`getCommentsByAgentID`**) 처리

마지막 액세스 패턴의 경우 새로운 차원인 `AgentID`로 조회를 수행해야 합니다. 또한 두 날짜 사이의 의견을 읽으려면 시간 기반 순서가 필요하므로 파티션 키로 `agent_id`를, 정렬 키로 `comm_date`를 사용하여 GSI를 생성합니다. 이 GSI의 데이터는 다음과 같습니다.

| Primary key | | Attributes | | | | | |
|---|---|---|---|---|---|---|---|
| Partition key: agentID | Sort key: comm_date | | | | | | |
| AgentA | 2023-04-30T12:00:24 | PK | SK | comm_id | complaint_state | comm_text | |
| | | Complaint123 | comm#2023-04-30T12:00:24#comm1 | comm1 | investigating | <comm text> | |
| | 2023-04-30T12:35:54 | PK | SK | comm_id | complaint_state | comm_text | attachments |
| | | Complaint123 | comm#2023-04-30T12:35:54#comm2 | comm2 | resolved | <comm text> | ["s3://URL_for_attachment1","s3://URL_for_attachment2"] |
| AgentB | 2023-05-10T16:00:00 | PK | SK | comm_id | complaint_state | comm_text | |
| | | Complaint1321 | comm#2023-05-10T16:00:00#comm3 | comm3 | investigating | <comm text> | |
| AgentC | 2022-12-31T19:40:00 | PK | SK | comm_id | complaint_state | comm_text | attachments |
| | | Complaint1444 | comm#2022-12-31T19:40:00#comm5 | comm5 | assigned | <comm text> | ["s3://URL_for_attachment1"] |

이 GSI에 대한 예시 쿼리는 `partition key agentID=AgentA` 및 `sort key=comm_date between (2023-04-30T12:30:00, 2023-05-01T09:00:00)`이며, 그 결과는 다음과 같습니다.

| Primary key | | Attributes | | | | | |
|---|---|---|---|---|---|---|---|
| Partition key: agentID | Sort key: comm_date | | | | | | |
| AgentA | 2023-04-30T12:00:24 | PK | SK | comm_id | complaint_state | comm_text | |
| | | Complaint123 | comm#2023-04-30T12:00:24#comm1 | comm1 | investigating | <comm text> | |
| | 2023-04-30T12:35:54 | PK | SK | comm_id | complaint_state | comm_text | attachments |
| | | Complaint123 | comm#2023-04-30T12:35:54#comm2 | comm2 | resolved | <comm text> | ["s3://URL_for_attachment1","s3://URL_for_attachment2"] |
| AgentB | 2023-05-10T16:00:00 | PK | SK | comm_id | complaint_state | comm_text | |
| | | Complaint1321 | comm#2023-05-10T16:00:00#comm3 | comm3 | investigating | <comm text> | |
| AgentC | 2022-12-31T19:40:00 | PK | SK | comm_id | complaint_state | comm_text | attachments |
| | | Complaint1444 | comm#2022-12-31T19:40:00#comm5 | comm5 | assigned | <comm text> | ["s3://URL_for_attachment1"] |

모든 액세스 패턴과 스키마 설계에서 이를 처리하는 방법이 아래 표에 요약되어 있습니다.

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 | 기타 조건/필터 |
|---|---|---|---|---|---|
| createComplaint | 기본 테이블 | PutItem | PK=complaint_id | SK=metadata | |
| updateComplaint | 기본 테이블 | UpdateItem | PK=complaint_id | SK=metadata | |
| updateSeveritybyComplaintID | 기본 테이블 | UpdateItem | PK=complaint_id | SK=metadata | |
| getComplaintByComplaintID | 기본 테이블 | GetItem | PK=complaint_id | SK=metadata | |
| addCommentByComplaintID | 기본 테이블 | TransactWriteItems | PK=complaint_id | SK=metadata, SK=comm#comm_date#comm_id | |

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 | 기타 조건/필터 |
|---|---|---|---|---|---|
| getAllCommentsByComplaintID | 기본 테이블 | 쿼리 | PK=complaint_id | SK begins_with "comm#" | |
| getLatestCommentByComplaintID | 기본 테이블 | 쿼리 | PK=complaint_id | SK begins_with "comm#" | scan_index_forward=False, Limit 1 |
| getAComplaintbyCustomerIDAndComplaintID | Customer_complaint_GSI | Query | customer_id=customer_id | complaint_id = complaint_id | |
| getAllComplaintsByCustomerID | Customer_complaint_GSI | Query | customer_id=customer_id | N/A | |
| escalateComplaintByComplaintID | 기본 테이블 | UpdateItem | PK=complaint_id | SK=metadata | |
| getAllEscalatedComplaints | Escalations_GSI | 스캔 | N/A | N/A | |
| getEscalatedComplaintsByAgentID(최신 것부터 가장 오래된 것 순) | Escalations_GSI | Query | escalated_to=agent_id | N/A | scan_index_forward=False |

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 | 기타 조건/필터 |
|---|---|---|---|---|---|
| getCommentsByAgentID(두 날짜 사이) | Agents_Comments_GSI | Query | agent_id= agent_id | SK between (date1, date2) | |

## 불만 관리 시스템 최종 스키마

다음은 최종 스키마 설계입니다. 이 스키마 설계를 JSON 파일로 다운로드하려면 GitHub의 DynamoDB 예제를 참조하세요.

### 기본 테이블

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | | | | | |
| Complaint123 | comm#2023-04-30T12:00:24#comm1 | comm_id | comm_date | complaint_state | comm_text | agentID | | | |
| | | comm1 | 2023-04-30T12:00:24 | investigating | <comm text> | AgentA | | | |
| | comm#2023-04-30T12:35:54#comm2 | comm_id | comm_date | complaint_state | comm_text | attachments | agentID | | |
| | | comm2 | 2023-04-30T12:35:54 | resolved | <comm text> | ["s3://URL_for_attachment1","s3://URL_for_attachment2"] | AgentA | | |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description | | |
| | | custABC | Complaint123 | resolved | 2023-04-30T12:00:00 | P2 | <description text> | | |
| Complaint1321 | comm#2023-05-10T16:00:00#comm3 | comm_id | comm_date | complaint_state | comm_text | agentID | | | |
| | | comm3 | 2023-05-10T16:00:00 | investigating | <comm text> | AgentB | | | |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description | escalated_to | escalation_time |
| | | custXYZ | Complaint1321 | investigating | 2023-05-10T15:58:00 | P2 | <descr_text> | AgentB | 2023-05-15T14:00:00 |
| Complaint1444 | comm#2022-12-31T19:32:00#comm4 | comm_id | comm_date | complaint_state | comm_text | | | | |
| | | comm4 | 2022-12-31T19:32:00 | waiting | <comm text> | | | | |
| | comm#2022-12-31T19:40:00#comm5 | comm_id | comm_date | complaint_state | comm_text | attachments | agentID | | |
| | | comm5 | 2022-12-31T19:40:00 | assigned | <comm text> | ["s3://URL_for_attachment1"] | AgentC | | |
| | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description | escalated_to | escalation_time |
| | | custXY32 | Complaint1444 | assigned | 2022-12-31T19:39:57 | P1 | <description text> | AgentB | 2023-01-03T04:00:07 |
| Complaint0987 | metadata | customer_id | complaint_id | current_state | creation_time | severity | complaint_description | | |
| | | custXYZ | Complaint0987 | assigned | 2023-06-10T12:30:08 | P3 | <description text> | | |

### Customer_Complaint_GSI

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Partition key: customer_id | Sort key: complaint_id | | | | | | | | |
| custABC | Complaint123 | PK | SK | current_state | creation_time | severity | complaint_description | | |
| | | Complaint123 | metadata | resolved | 2023-04-30T12:00:00 | P2 | <description text> | | |
| custXY32 | Complaint1444 | PK | SK | current_state | creation_time | severity | complaint_description | escalated_to | escalation_time |
| | | Complaint1444 | metadata | assigned | 2022-12-31T19:39:57 | P1 | <description text> | AgentB | 2023-01-03T04:00:07 |
| custXYZ | Complaint0987 | PK | SK | current_state | creation_time | severity | complaint_description | | |
| | | Complaint0987 | metadata | assigned | 2023-06-10T12:30:08 | P3 | <description text> | | |
| | Complaint1321 | PK | SK | current_state | creation_time | severity | complaint_description | escalated_to | escalation_time |
| | | Complaint1321 | metadata | investigating | 2023-05-10T15:58:00 | P2 | <descr_text> | AgentB | 2023-05-15T14:00:00 |

## Escalations_GSI

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Partition key: escalated_to | Sort key: escalation_time | | | | | | | | |
| AgentB | 2023-01-03T04:00:07 | PK | SK | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | Complaint1444 | metadata | custXY32 | Complaint1444 | assigned | 2022-12-31T19:39:57 | P1 | <description text> |
| | 2023-05-15T14:00:00 | PK | SK | customer_id | complaint_id | current_state | creation_time | severity | complaint_description |
| | | Complaint1321 | metadata | custXYZ | Complaint1321 | investigating | 2023-05-10T15:58:00 | P2 | <descr_text> |

## Agents_Comments_GSI

| Primary key | | Attributes | | | | | |
|---|---|---|---|---|---|---|---|
| Partition key: agentID | Sort key: comm_date | | | | | | |
| AgentA | 2023-04-30T12:00:24 | PK | SK | comm_id | complaint_state | comm_text | |
| | | Complaint123 | comm#2023-04-30T12:00:24#comm1 | comm1 | investigating | <comm text> | |
| | 2023-04-30T12:35:54 | PK | SK | comm_id | complaint_state | comm_text | attachments |
| | | Complaint123 | comm#2023-04-30T12:35:54#comm2 | comm2 | resolved | <comm text> | ["s3://URL_for_attachment1","s3://URL_for_attachment2"] |
| AgentB | 2023-05-10T16:00:00 | PK | SK | comm_id | complaint_state | comm_text | |
| | | Complaint1321 | comm#2023-05-10T16:00:00#comm3 | comm3 | investigating | <comm text> | |
| AgentC | 2022-12-31T19:40:00 | PK | SK | comm_id | complaint_state | comm_text | attachments |
| | | Complaint1444 | comm#2022-12-31T19:40:00#comm5 | comm5 | assigned | <comm text> | ["s3://URL_for_attachment1"] |

# 이 스키마 설계와 함께 NoSQL Workbench 사용

이 최종 스키마를 DynamoDB 데이터 모델링, 데이터 시각화, 쿼리 개발 기능을 제공하는 시각적 도구인 NoSQL Workbench로 가져와서 새 프로젝트를 추가로 탐색하고 편집할 수 있습니다. 시작하려면 다음 단계를 따릅니다.

1. NoSQL Workbench 다운로드 자세한 내용은 the section called “다운로드” 단원을 참조하십시오.

2. 위에 나열된 JSON 스키마 파일을 다운로드합니다. 이 파일은 이미 NoSQL Workbench 모델 형식으로 되어 있습니다.
3. JSON 스키마 파일을 NoSQL Workbench로 가져옵니다. 자세한 내용은 the section called “기존 모델 가져오기” 단원을 참조하십시오.
4. NOSQL Workbench로 가져온 후 데이터 모델을 편집할 수 있습니다. 자세한 내용은 the section called “기존 모델 편집” 단원을 참조하십시오.
5. 데이터 모델을 시각화하거나, 샘플 데이터를 추가하거나, CSV 파일에서 샘플 데이터를 가져오려면 NoSQL Workbench의 Data Visualizer 기능을 사용하세요.

# DynamoDB에서 반복 결제 스키마 설계

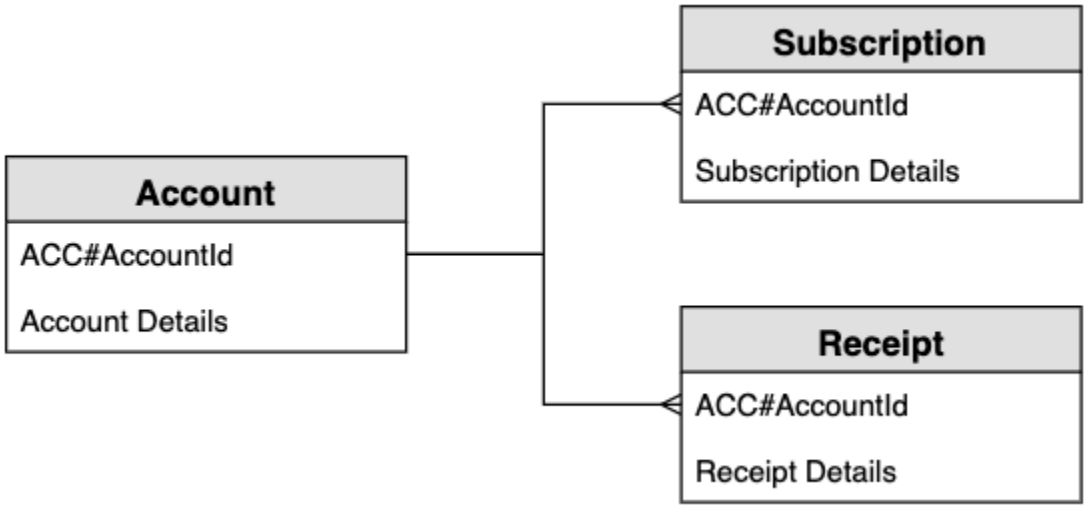
## 반복 결제 비즈니스 사용 사례

이 사용 사례에서는 DynamoDB를 사용하여 반복 결제 시스템을 구현하는 방법을 설명합니다. 데이터 모델에는 계정, 구독, 영수증과 같은 엔터티가 있습니다. 사용 사례의 구체적인 내용은 다음과 같습니다.

- 각 계정에는 여러 구독이 있을 수 있습니다.
- 구독에는 다음 결제를 처리해야 하는 `NextPaymentDate`가 있고 고객에게 이메일 미리 알림이 전송되는 `NextReminderDate`가 있습니다
- 결제가 처리될 때 저장되고 업데이트되는 구독 항목이 있습니다(평균 항목 크기는 약 1KB이며 처리량은 계정 및 구독 수에 따라 다름).
- 또한 결제 처리자는 테이블에 저장되고 TTL 속성을 사용하여 일정 기간 후에 만료되도록 설정된 프로세스의 일부로 영수증을 생성합니다

## 반복 결제 엔터티 관계 다이어그램

다음은 반복 결제 시스템 스키마 설계에 사용할 엔터티 관계 다이어그램(ERD)입니다.




## 반복 결제 시스템 액세스 패턴

다음은 반복 결제 시스템 스키마를 설계할 때 고려할 액세스 패턴입니다.

1. `createSubscription`
2. `createReceipt`
3. `updateSubscription`
4. `getDueRemindersByDate`
5. `getDuePaymentsByDate`
6. `getSubscriptionsByAccount`
7. `getReceiptsByAccount`

## 반복 결제 스키마 설계

일반 이름 PK 및 SK는 계정, 구독 및 영수증 엔터티와 같은 다양한 유형의 엔터티를 동일한 테이블에 저장할 수 있도록 키 속성에 사용됩니다. 사용자는 먼저 구독을 생성하여 제품에 대한 대금을 매달 같은 날에 지불하는 데 동의합니다. 매월 어느 날에 결제를 처리할지 사용자가 선택할 수 있습니다. 결제가 처리되기 전에 전송되는 미리 알림도 있습니다. 애플리케이션은 매일 실행되는 두 개의 배치 작업, 즉 해당 날짜 기한의 미리 알림을 전송하는 배치 작업과 해당 날짜 기한의 모든 결제를 처리하는 배치 작업을 통해 작동합니다.

1단계: 액세스 패턴 1(**`createSubscription`**) 처리

액세스 패턴 1(`createSubscription`)은 구독을 처음 만드는 데 사용되
며 SKU, `NextPaymentDate`, `NextReminderDate`, `PaymentDetails` 등의 세부 정보가 설정됩니

다. 이 단계에서는 구독이 하나인 계정 한 개에 대한 테이블 상태를 보여 줍니다. 항목 컬렉션에 여러 개의 구독이 있을 수 있으므로 이는 일대다 관계입니다.

| Primary key | | Attributes | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | Email | PaymentDay | PaymentAmount | LastPaymentDate | NextPaymentDate | LastReminderDate | NextReminderDate | SKU | PaymentDetails | CreatedDate |
| ACC#123 | SUB#123#SKU#999 | s@s.com | 28 | 12.99 | 1970-01-01T00:00:00.000Z | 2023-05-28 | 1970-01-01T00:00:00.000Z | 2023-05-21 | 999 | {"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}} | 2023-05-18T09:41:25.856Z |

2단계: 액세스 패턴 2(**createReceipt**) 및 3(**updateSubscription**) 처리

액세스 패턴 2(createReceipt)는 영수증 항목을 만드는 데 사용됩니다. 매월 결제가 처리되면 결제 처리자는 기본 테이블에 영수증을 다시 기록합니다. 항목 컬렉션에 여러 개의 영수증이 있을 수 있으므로 이는 일대다 관계입니다. 또한 결제 처리자는 구독 항목을 업데이트하여(액세스 패턴 3(updateSubscription)) 다음 달의 NextReminderDate 또는 NextPaymentDate를 업데이트합니다.

| Primary key | | Attributes | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | | | | | | | |
| ACC#123 | REC#12023-05-28T14:15:39.24#SKU#999 | Email | SKU | ProcessedDate | ProcessedAmount | TTL | | | | | |
| | | s@s.com | 999 | 2023-05-28T14:15:39.247Z | 12.99 | 1700318200 | | | | | |
| | SUB#123#SKU#999 | Email | PaymentDay | PaymentAmount | LastPaymentDate | NextPaymentDate | LastReminderDate | NextReminderDate | SKU | PaymentDetails | CreatedDate |
| | | s@s.com | 28 | 12.99 | 2023-05-18T14:15:39.247Z | 2023-06-28 | 2023-05-21T14:15:39.247Z | 2023-06-21 | 999 | {"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}} | 2023-05-18T09:41:25.856Z |

3단계: 액세스 패턴 4(**getDueRemindersByDate**) 처리

애플리케이션은 당일 결제 미리 알림을 배치로 처리합니다. 따라서 애플리케이션은 계정이 아닌 날짜라는 다른 차원에서 구독에 액세스해야 합니다. 이는 글로벌 보조 인덱스(GSI)의 좋은 사용 사례입니다. 이 단계에서는 NextReminderDate를 GSI 파티션 키로 사용하는 인덱스 GSI-1을 추가합니다. 모든 항목을 복제할 필요는 없습니다. 이 GSI는 희소 인덱스이며 영수증 항목은 복제되지 않습니다. 또한 모든 속성을 프로젝션할 필요는 없으며, 속성의 일부만 포함하면 됩니다. 아래 이미지는 GSI-1의 스키마를 보여주며 애플리케이션이 미리 알림 이메일을 보내는 데 필요한 정보를 제공합니다.

| Primary key | | Attributes | | | | |
|---|---|---|---|---|---|---|
| Partition key: NextReminderDate | Sort key: LastReminderDate | SK | PK | SKU | Email | NextPaymentDate |
| 2023-06-21 | 2023-05-21T14:15:39.247Z | SUB#123#SKU#999 | ACC#123 | 999 | s@s.com | 2023-06-28 |

4단계: 액세스 패턴 5(**getDuePaymentsByDate**) 처리

애플리케이션은 미리 알림과 마찬가지로 당일 결제를 배치로 처리합니다. 이 단계에서는 GSI-2를 추가하며, NextPaymentDate를 GSI 파티션 키로 사용합니다. 모든 항목을 복제할 필요는 없습니다. 영

수증 항목은 복제되지 않으므로 이 GSI는 희소 인덱스입니다. 아래 이미지는 GSI-2의 스키마를 보여줍니다.

| Primary key | | Attributes | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Partition key: NextPaymentDate | Sort key: LastPaymentDate | PK | SK | Email | PaymentDay | PaymentAmount | SKU | PaymentDetails |
| 2023-06-28 | 2023-05-18T14:15:39.247Z | ACC#123 | SUB#123#SKU#999 | s@s.com | 28 | 12.99 | 999 | {"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}} |

5단계: 액세스 패턴 6(**getSubscriptionsByAccount**) 및 7(**getReceiptsByAccount**) 처리

애플리케이션은 계정 식별자(PK)를 대상으로 하며 범위 연산자를 사용하여 SK가 'SUB#'로 시작하는 모든 항목을 가져오는 쿼리를 기본 테이블에서 사용하여 계정의 모든 구독을 검색할 수 있습니다. 또한 애플리케이션은 동일한 쿼리 구조로 범위 연산자를 사용하여 SK가 'REC#'로 시작하는 모든 항목을 가져옴으로써 모든 영수증을 검색할 수 있습니다. 이를 통해 액세스 패턴 6(getSubscriptionsByAccount) 및 7(getReceiptsByAccount)을 충족할 수 있습니다. 애플리케이션은 이러한 액세스 패턴을 사용하므로 사용자는 현재 구독과 지난 6개월 동안의 과거 영수증을 볼 수 있습니다. 이 단계에서 테이블 스키마는 변경되지 않으며, 액세스 패턴 6(getSubscriptionsByAccount)의 구독 항목만 대상으로 지정하는 방법을 아래에서 확인할 수 있습니다.

| Primary key | | Attributes | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | | | | | | | |
| | REC#12023-05-28T14:15:39.24#SKU#999 | Email | SKU | ProcessedDate | ProcessedAmount | TTL | | | | | |
| | | s@s.com | 999 | 2023-05-28T14:15:39.247Z | 12.99 | 1700318200 | | | | | |
| ACC#123 | SUB#123#SKU#999 | Email | PaymentDay | PaymentAmount | LastPaymentDate | NextPaymentDate | LastReminderDate | NextReminderDate | SKU | PaymentDetails | CreatedDate |
| | | s@s.com | 28 | 12.99 | 2023-05-18T14:15:39.247Z | 2023-06-28 | 2023-05-21T14:15:39.247Z | 2023-06-21 | 999 | {"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}} | 2023-05-18T09:41:25.856Z |

모든 액세스 패턴과 스키마 설계에서 이를 처리하는 방법이 아래 표에 요약되어 있습니다.

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 |
|---|---|---|---|---|
| createSubscription | 기본 테이블 | PutItem | ACC#account_id | SUB#<SUBID>#SKU<SKUID> |
| createReceipt | 기본 테이블 | PutItem | ACC#account_id | REC#<ReceiptDate>#SKU<SKUID> |

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 |
|---|---|---|---|---|
| updateSubscription | 기본 테이블 | UpdateItem | ACC#account_id | SUB#<SUBID>#SKU<SKUID> |
| getDueRemindersByDate | GSI-1 | Query | <NextReminderDate> | |
| getDuePaymentsByDate | GSI-2 | Query | <NextPaymentDate> | |
| getSubscriptionsByAccount | 기본 테이블 | 쿼리 | ACC#account_id | SK begins_with “SUB#” |
| getReceiptsByAccount | 기본 테이블 | 쿼리 | ACC#account_id | SK begins_with “REC#” |

## 반복 결제 최종 스키마

다음은 최종 스키마 설계입니다. 이 스키마 설계를 JSON 파일로 다운로드하려면 GitHub의 DynamoDB 예제를 참조하세요.

### 기본 테이블

| Primary key | | Attributes | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | | | | | | | | |
| ACC#123 | REC#12023-05-28T14:15:39.24#SKU#999 | Email | SKU | ProcessedDate | ProcessedAmount | TTL | | | | | | |
| | | s@s.com | 999 | 2023-05-28T14:15:39.247Z | 12.99 | 1700318200 | | | | | | |
| | SUB#123#SKU#999 | Email | PaymentDay | PaymentAmount | LastPaymentDate | NextPaymentDate | LastReminderDate | NextReminderDate | SKU | PaymentDetails | CreatedDate | |
| | | s@s.com | 28 | 12.99 | 2023-05-18T14:15:39.247Z | 2023-06-28 | 2023-05-21T14:15:39.247Z | 2023-06-21 | 999 | {"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}} | 2023-05-18T09:41:25.856Z | |

### GSI-1

| Primary key | | Attributes | | | | |
|---|---|---|---|---|---|---|
| Partition key: NextReminderDate | Sort key: LastReminderDate | | | | | |
| 2023-06-21 | 2023-05-21T14:15:39.247Z | SK | PK | SKU | Email | NextPaymentDate |
| | | SUB#123#SKU#999 | ACC#123 | 999 | s@s.com | 2023-06-28 |

### GSI-2

| Primary key | | Attributes | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Partition key: NextPaymentDate | Sort key: LastPaymentDate | PK | SK | Email | PaymentDay | PaymentAmount | SKU | PaymentDetails |
| 2023-06-28 | 2023-05-18T14:15:39.247Z | ACC#123 | SUB#123#SKU #999 | s@s.com | 28 | 12.99 | 999 | {"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}} |

## 이 스키마 설계와 함께 NoSQL Workbench 사용

이 최종 스키마를 DynamoDB 데이터 모델링, 데이터 시각화, 쿼리 개발 기능을 제공하는 시각적 도구인 NoSQL Workbench로 가져와서 새 프로젝트를 추가로 탐색하고 편집할 수 있습니다. 시작하려면 다음 단계를 따릅니다.

1. NoSQL Workbench 다운로드 자세한 내용은 the section called “다운로드” 단원을 참조하십시오.
2. 위에 나열된 JSON 스키마 파일을 다운로드합니다. 이 파일은 이미 NoSQL Workbench 모델 형식으로 되어 있습니다.
3. JSON 스키마 파일을 NoSQL Workbench로 가져옵니다. 자세한 내용은 the section called “기존 모델 가져오기” 단원을 참조하십시오.
4. NOSQL Workbench로 가져온 후 데이터 모델을 편집할 수 있습니다. 자세한 내용은 the section called “기존 모델 편집” 단원을 참조하십시오.
5. 데이터 모델을 시각화하거나, 샘플 데이터를 추가하거나, CSV 파일에서 샘플 데이터를 가져오려면 NoSQL Workbench의 Data Visualizer 기능을 사용하세요.

# DynamoDB에서 디바이스 상태 업데이트 모니터링

이 사용 사례에서는 DynamoDB를 사용하여 DynamoDB에서 디바이스 상태 업데이트(또는 디바이스 상태 변경)를 모니터링하는 방법을 설명합니다.

## 사용 사례

IoT 사용 사례(예: 스마트 팩토리)에서는 운영자가 많은 디바이스를 모니터링해야 하며 운영자는 주기적으로 상태 또는 로그를 모니터링 시스템에 전송합니다. 디바이스에 문제가 발생할 경우 디바이스의 상태가 정상에서 경고로 바뀝니다. 디바이스는 비정상적인 동작의 심각도 및 유형에 따라 다양한 로그 수준 또는 상태를 갖게 됩니다. 시스템에서는 운영자를 배정하여 디바이스를 점검하도록 하며, 필요한 경우 운영자가 감독자에게 문제를 에스컬레이션할 수 있습니다.

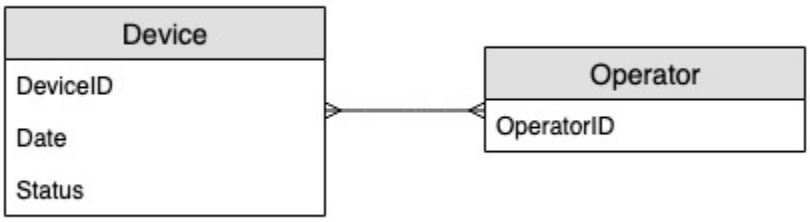
이 시스템의 몇 가지 일반적인 액세스 패턴은 다음과 같습니다.

- 디바이스에 대한 로그 항목 생성

- 특정 디바이스 상태에 대한 모든 로그 가져오기(최근 로그 먼저 표시)
- 두 날짜 사이의 지정된 운영자에 대한 모든 로그 가져오기
- 지정된 감독자에게 에스컬레이션된 모든 로그 가져오기
- 지정된 감독자에게 에스컬레이션되고 특정 디바이스 상태가 포함된 모든 로그 가져오기
- 특정 날짜에 지정된 감독자에게 에스컬레이션되고 특정 디바이스 상태가 포함된 모든 로그 가져오기

## 엔터티 관계 다이어그램

다음은 디바이스 상태 업데이트를 모니터링하는 데 사용할 엔터티 관계 다이어그램(ERD)입니다.




## 액세스 패턴

디바이스 상태 업데이트를 모니터링하는 데 고려할 액세스 패턴은 다음과 같습니다.

1. `createLogEntryForSpecificDevice`
2. `getLogsForSpecificDevice`
3. `getWarningLogsForSpecificDevice`
4. `getLogsForOperatorBetweenTwoDates`
5. `getEscalatedLogsForSupervisor`
6. `getEscalatedLogsWithSpecificStatusForSupervisor`
7. `getEscalatedLogsWithSpecificStatusForSupervisorForDate`

## 스키마 설계 진화

1단계: 액세스 패턴 1(**`createLogEntryForSpecificDevice`**) 및 2(**`getLogsForSpecificDevice`**) 처리

디바이스 추적 시스템의 규모 조정 단위는 개별 디바이스입니다. 이 시스템에서는 deviceID가 디바이스를 고유하게 식별합니다. 따라서 deviceID는 파티션 키로 적합한 후보입니다. 각 디바이스는 주기적으로(예: 5분마다) 추적 시스템에 정보를 전송합니다. 이렇게 정렬하면 날짜가 논리적인 정렬 기준이 되므로 정렬 키도 됩니다. 이 경우 샘플 데이터는 다음과 같습니다.

| Primary key | | Attributes |
|---|---|---|
| Partition key: DeviceID | Sort key: State#Date | |
| d#12345 | 2020-04-24T14:40:00 | State |
| | | WARNING1 |
| | 2020-04-24T14:45:00 | State |
| | | WARNING1 |
| | 2020-04-24T14:50:00 | State |
| | | WARNING1 |
| | 2020-04-24T14:55:00 | State |
| | | NORMAL |
| d#54321 | 2020-04-11T05:50:00 | State |
| | | WARNING3 |
| | 2020-04-11T05:55:00 | State |
| | | WARNING3 |
| | 2020-04-11T06:00:00 | State |
| | | NORMAL |
| | 2020-04-11T09:25:00 | State |
| | | WARNING2 |
| | 2020-04-11T09:30:00 | State |
| | | NORMAL |
| d#11223 | 2020-04-27T16:10:00 | State |
| | | WARNING4 |
| | 2020-04-27T16:15:00 | State |
| | | WARNING4 |

특정 디바이스에 대한 로그 항목을 가져오려는 경우 파티션 키 DeviceID="d#12345"를 사용하여 쿼리 작업을 수행할 수 있습니다.

2단계: 액세스 패턴 3(**getWarningLogsForSpecificDevice**) 처리

State는 키가 아닌 속성이므로 현재 스키마로 액세스 패턴 3을 처리하려면 필터 표현식이 필요합니다. DynamoDB에서는 키 조건 표현식을 사용하여 데이터를 읽은 후 필터 표현식이 적용됩니다. 예를 들어 d#12345에 대한 경고 로그를 가져오는 경우, 파티션 키 DeviceID="d#12345"를 사용한 쿼리

작업은 위 테이블에서 4개 항목을 읽은 다음 경고 상태가 없는 1개 항목을 필터링합니다. 이 접근 방식은 규모가 커지면 효율적이지 않습니다. 제외된 항목의 비율이 낮거나 쿼리가 자주 수행되지 않는 경우 필터 표현식은 쿼리되는 항목을 제외하는 좋은 방법이 될 수 있습니다. 하지만 테이블에서 많은 항목이 검색되고 대부분의 항목이 필터링되는 경우에는 테이블 설계를 계속 발전시켜 더 효율적으로 실행되도록 할 수 있습니다.

복합 정렬 키를 사용하여 이 액세스 패턴을 처리하는 방법을 변경해 보겠습니다. 정렬 키가 `State#Date`로 변경된 DeviceStateLog_3.json에서 샘플 데이터를 가져올 수 있습니다. 이 정렬 키는 `State`, `#` 및 `Date` 속성으로 구성됩니다. 이 예에서는 `#`이 구분 기호로 사용됩니다. 데이터는 이제 다음과 같습니다.

| Primary key | |
|---|---|
| **Partition key: DeviceID** | **Sort key: State#Date** |
| d#12345 | NORMAL#2020-04-24T14:55:00 |
| | WARNING1#2020-04-24T14:40:00 |
| | WARNING1#2020-04-24T14:45:00 |
| | WARNING1#2020-04-24T14:50:00 |

디바이스에 대한 경고 로그만 가져오려는 경우 이 스키마를 사용하면 쿼리의 대상이 더욱 명확해집니다. 쿼리의 키 조건은 파티션 키 `DeviceID="d#12345"` 및 정렬 키 `State#Date begins_with “WARNING”`을 사용합니다. 이 쿼리는 경고 상태의 관련 항목 3개만 읽습니다.

3단계: 액세스 패턴 4(**`getLogsForOperatorBetweenTwoDates`**) 처리

예제 데이터와 함께 `Operator` 속성이 `DeviceStateLog` 테이블에 추가된 DeviceStateLog_4.jsonD를 가져올 수 있습니다.

| Primary key | | Attributes | | |
|---|---|---|---|---|
| **Partition key: DeviceID** | **Sort key: State#Date** | | | |
| d#12345 | NORMAL#2020-04-24T14:55:00 | Operator | Date | State |
| | | Liz | 2020-04-24T14:55:00 | NORMAL |
| | WARNING1#2020-04-24T14:40:00 | Operator | Date | State |
| | | Liz | 2020-04-24T14:40:00 | WARNING1 |
| | WARNING1#2020-04-24T14:45:00 | Operator | Date | State |
| | | Liz | 2020-04-24T14:45:00 | WARNING1 |
| | WARNING1#2020-04-24T14:50:00 | Operator | Date | State |
| | | Liz | 2020-04-24T14:50:00 | WARNING1 |
| d#54321 | NORMAL#2020-04-11T06:00:00 | Operator | Date | State |
| | | Liz | 2020-04-11T06:00:00 | NORMAL |
| | NORMAL#2020-04-11T09:30:00 | Operator | Date | State |
| | | Sue | 2020-04-11T09:30:00 | NORMAL |
| | WARNING2#2020-04-11T09:25:00 | Operator | Date | State |
| | | Sue | 2020-04-11T09:25:00 | WARNING2 |
| | WARNING3#2020-04-11T05:50:00 | Operator | Date | State |
| | | Sue | 2020-04-11T05:50:00 | WARNING3 |
| | WARNING3#2020-04-11T05:55:00 | Operator | Date | State |
| | | Liz | 2020-04-11T05:55:00 | WARNING3 |
| d#11223 | WARNING4#2020-04-27T16:10:00 | Operator | Date | State |
| | | Sue | 2020-04-27T16:10:00 | WARNING4 |
| | WARNING4#2020-04-27T16:15:00 | Operator | Date | State |
| | | Sue | 2020-04-27T16:15:00 | WARNING4 |

`Operator`는 현재 파티션 키가 아니므로 `OperatorID`를 기반으로 이 테이블에서 직접 키-값 조회를 수행할 수 있는 방법이 없습니다. `OperatorID`에 대한 글로벌 보조 인덱스를 사용하여 새 항목 컬렉

션을 생성해야 합니다. 액세스 패턴에는 날짜를 기반으로 한 조회가 필요하므로 Date는 글로벌 보조 인덱스(GSI)의 정렬 키 속성입니다 현재 GSI는 다음과 같습니다.

| Primary key | | Attributes | | |
|---|---|---|---|---|
| **Partition key: Operator** | **Sort key: Date** | | | |
| Liz | 2020-04-11T05:55:00 | DeviceID | State#Date | State |
| | | d#54321 | WARNING3#2020-04-11T05:55:00 | WARNING3 |
| | 2020-04-11T06:00:00 | DeviceID | State#Date | State |
| | | d#54321 | NORMAL#2020-04-11T06:00:00 | NORMAL |
| | 2020-04-24T14:40:00 | DeviceID | State#Date | State |
| | | d#12345 | WARNING1#2020-04-24T14:40:00 | WARNING1 |
| | 2020-04-24T14:45:00 | DeviceID | State#Date | State |
| | | d#12345 | WARNING1#2020-04-24T14:45:00 | WARNING1 |
| | 2020-04-24T14:50:00 | DeviceID | State#Date | State |
| | | d#12345 | WARNING1#2020-04-24T14:50:00 | WARNING1 |
| | 2020-04-24T14:55:00 | DeviceID | State#Date | State |
| | | d#12345 | NORMAL#2020-04-24T14:55:00 | NORMAL |
| Sue | 2020-04-11T05:50:00 | DeviceID | State#Date | State |
| | | d#54321 | WARNING3#2020-04-11T05:50:00 | WARNING3 |
| | 2020-04-11T09:25:00 | DeviceID | State#Date | State |
| | | d#54321 | WARNING2#2020-04-11T09:25:00 | WARNING2 |
| | 2020-04-11T09:30:00 | DeviceID | State#Date | State |
| | | d#54321 | NORMAL#2020-04-11T09:30:00 | NORMAL |
| | 2020-04-27T16:10:00 | DeviceID | State#Date | State |
| | | d#11223 | WARNING4#2020-04-27T16:10:00 | WARNING4 |
| | 2020-04-27T16:15:00 | DeviceID | State#Date | State |
| | | d#11223 | WARNING4#2020-04-27T16:15:00 | WARNING4 |

액세스 패턴 4(`getLogsForOperatorBetweenTwoDates`)의 경우 `2020-04-11T05:58:00`과 `2020-04-24T14:50:00` 사이로 파티션 키 `OperatorID=Liz` 및 정렬 키 `Date`를 사용하여 이 GSI를 쿼리할 수 있습니다.

| Primary key | | Attributes | | |
|---|---|---|---|---|
| **Partition key: Operator** | **Sort key: Date** | | | |
| | 2020-04-11T05:55:00 | DeviceID | State#Date | State |
| | | d#54321 | WARNING3#2020-04-11T05:55:00 | WARNING3 |
| Liz | 2020-04-11T06:00:00 | DeviceID | State#Date | State |
| | | d#54321 | NORMAL#2020-04-11T06:00:00 | NORMAL |
| | 2020-04-24T14:40:00 | DeviceID | State#Date | State |
| | | d#12345 | WARNING1#2020-04-24T14:40:00 | WARNING1 |
| | 2020-04-24T14:45:00 | DeviceID | State#Date | State |
| | | d#12345 | WARNING1#2020-04-24T14:45:00 | WARNING1 |
| | 2020-04-24T14:50:00 | DeviceID | State#Date | State |
| | | d#12345 | WARNING1#2020-04-24T14:50:00 | WARNING1 |
| | 2020-04-24T14:55:00 | DeviceID | State#Date | State |
| | | d#12345 | NORMAL#2020-04-24T14:55:00 | NORMAL |
| Sue | 2020-04-11T05:50:00 | DeviceID | State#Date | State |
| | | d#54321 | WARNING3#2020-04-11T05:50:00 | WARNING3 |
| | 2020-04-11T09:25:00 | DeviceID | State#Date | State |
| | | d#54321 | WARNING2#2020-04-11T09:25:00 | WARNING2 |
| | 2020-04-11T09:30:00 | DeviceID | State#Date | State |
| | | d#54321 | NORMAL#2020-04-11T09:30:00 | NORMAL |
| | 2020-04-27T16:10:00 | DeviceID | State#Date | State |
| | | d#11223 | WARNING4#2020-04-27T16:10:00 | WARNING4 |
| | 2020-04-27T16:15:00 | DeviceID | State#Date | State |
| | | d#11223 | WARNING4#2020-04-27T16:15:00 | WARNING4 |

4단계: 액세스 패턴 5(**getEscalatedLogsForSupervisor**), 6(**getEscalatedLogsWithSpecificStatusForSupervisor**), 7(**getEscalatedLogsWithSpecificStatusForSupervisorForDate**) 처리

이러한 액세스 패턴을 처리하기 위해 희소 인덱스를 사용합니다.

글로벌 보조 인덱스는 기본적으로 희소 인덱스이므로 인덱스의 프라이머리 키 속성을 포함하는 기본 테이블의 항목만 실제로 인덱스에 표시됩니다. 이는 모델링되는 액세스 패턴과 관련이 없는 항목을 제외하는 또 다른 방법입니다.

예제 데이터와 함께 `EscalatedTo` 속성이 `DeviceStateLog` 테이블에 추가된 DeviceStateLog_6.json을 가져올 수 있습니다. 앞서 언급했듯이 모든 로그가 감독자에게 에스컬레이션되는 것은 아닙니다.

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| **Partition key: DeviceID** | **Sort key: State#Date** | | | | |
| d#12345 | NORMAL#2020-04-24T14:55:00 | Operator | Date | State | |
| | | Liz | 2020-04-24T14:55:00 | NORMAL | |
| | WARNING1#2020-04-24T14:40:00 | Operator | Date | State | |
| | | Liz | 2020-04-24T14:40:00 | WARNING1 | |
| | WARNING1#2020-04-24T14:45:00 | Operator | Date | State | |
| | | Liz | 2020-04-24T14:45:00 | WARNING1 | |
| | WARNING1#2020-04-24T14:50:00 | Operator | Date | State | |
| | | Liz | 2020-04-24T14:50:00 | WARNING1 | |
| d#54321 | NORMAL#2020-04-11T06:00:00 | Operator | Date | State | |
| | | Liz | 2020-04-11T06:00:00 | NORMAL | |
| | NORMAL#2020-04-11T09:30:00 | Operator | Date | State | |
| | | Sue | 2020-04-11T09:30:00 | NORMAL | |
| | WARNING2#2020-04-11T09:25:00 | Operator | Date | State | |
| | | Sue | 2020-04-11T09:25:00 | WARNING2 | |
| | WARNING3#2020-04-11T05:50:00 | Operator | Date | State | |
| | | Sue | 2020-04-11T05:50:00 | WARNING3 | |
| | WARNING3#2020-04-11T05:55:00 | Operator | Date | State | |
| | | Liz | 2020-04-11T05:55:00 | WARNING3 | |
| d#11223 | WARNING4#2020-04-27T16:10:00 | Operator | Date | State | |
| | | Sue | 2020-04-27T16:10:00 | WARNING4 | |
| | WARNING4#2020-04-27T16:15:00 | Operator | Date | State | EscalatedTo |
| | | Sue | 2020-04-27T16:15:00 | WARNING4 | Sara |

이제 파티션 키가 `EscalatedTo`이고 정렬 키가 `State#Date`인 새 GSI를 생성할 수 있습니다. `EscalatedTo`와 `State#Date` 속성이 모두 있는 항목만 인덱스에 표시됩니다.

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| Partition key: EscalatedTo | Sort key: State#Date | DeviceID | Operator | Date | State |
| Sara | WARNING4#2020-04-27T16:15:00 | d#11223 | Sue | 2020-04-27T16:15:00 | WARNING4 |

나머지 액세스 패턴은 다음과 같이 요약됩니다.

모든 액세스 패턴과 스키마 설계에서 이를 처리하는 방법이 아래 표에 요약되어 있습니다.

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 | 기타 조건/필터 |
|---|---|---|---|---|---|
| createLogEntryForSpecificDevice | 기본 테이블 | PutItem | DeviceID=deviceId | State#Date=state#date | |
| getLogsForSpecificDevice | 기본 테이블 | 쿼리 | DeviceID=deviceId | State#Date begins_with "state1#" | ScanIndexForward = False |
| getWarningLogsForSpecificDevice | 기본 테이블 | 쿼리 | DeviceID=deviceId | State#Date begins_with "WARNING" | |
| getLogsForOperatorBetweenTwoDates | GSI-1 | Query | Operator=operatorName | Date between date1 and date2 | |
| getEscalatedLogsForSupervisor | GSI-2 | Query | EscalatedTo=supervisorName | | |
| getEscalatedLogsWithSpecifi | GSI-2 | Query | EscalatedTo=supervisorName | State#Date begins_with "state1#" | |

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 | 기타 조건/필터 |
|---|---|---|---|---|---|
| cStatusFo rSupervisor | | | | | |
| getEscala tedLogsWi thSpecifi cStatusFo rSupervis orForDate | GSI-2 | Query | Escalated To=superv isorName | State#Dat e begins_wi th "state1#d ate1" | |

## 최종 스키마

다음은 최종 스키마 설계입니다. 이 스키마 설계를 JSON 파일로 다운로드하려면 GitHub의 DynamoDB 예제를 참조하세요.

기본 테이블

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| **Partition key: DeviceID** | **Sort key: State#Date** | | | | |
| d#12345 | NORMAL#2020-04-24T14:55:00 | Operator | Date | State | |
| | | Liz | 2020-04-24T14:55:00 | NORMAL | |
| | WARNING1#2020-04-24T14:40:00 | Operator | Date | State | |
| | | Liz | 2020-04-24T14:40:00 | WARNING1 | |
| | WARNING1#2020-04-24T14:45:00 | Operator | Date | State | |
| | | Liz | 2020-04-24T14:45:00 | WARNING1 | |
| | WARNING1#2020-04-24T14:50:00 | Operator | Date | State | |
| | | Liz | 2020-04-24T14:50:00 | WARNING1 | |
| d#54321 | NORMAL#2020-04-11T06:00:00 | Operator | Date | State | |
| | | Liz | 2020-04-11T06:00:00 | NORMAL | |
| | NORMAL#2020-04-11T09:30:00 | Operator | Date | State | |
| | | Sue | 2020-04-11T09:30:00 | NORMAL | |
| | WARNING2#2020-04-11T09:25:00 | Operator | Date | State | |
| | | Sue | 2020-04-11T09:25:00 | WARNING2 | |
| | WARNING3#2020-04-11T05:50:00 | Operator | Date | State | |
| | | Sue | 2020-04-11T05:50:00 | WARNING3 | |
| | WARNING3#2020-04-11T05:55:00 | Operator | Date | State | |
| | | Liz | 2020-04-11T05:55:00 | WARNING3 | |
| d#11223 | WARNING4#2020-04-27T16:10:00 | Operator | Date | State | |
| | | Sue | 2020-04-27T16:10:00 | WARNING4 | |
| | WARNING4#2020-04-27T16:15:00 | Operator | Date | State | EscalatedTo |
| | | Sue | 2020-04-27T16:15:00 | WARNING4 | Sara |

## GSI-1

| Primary key | | Attributes | | |
|---|---|---|---|---|
| **Partition key: Operator** | **Sort key: Date** | | | |
| Liz | 2020-04-11T05:55:00 | DeviceID | State#Date | State |
| | | d#54321 | WARNING3#2020-04-11T05:55:00 | WARNING3 |
| | 2020-04-11T06:00:00 | DeviceID | State#Date | State |
| | | d#54321 | NORMAL#2020-04-11T06:00:00 | NORMAL |
| | 2020-04-24T14:40:00 | DeviceID | State#Date | State |
| | | d#12345 | WARNING1#2020-04-24T14:40:00 | WARNING1 |
| | 2020-04-24T14:45:00 | DeviceID | State#Date | State |
| | | d#12345 | WARNING1#2020-04-24T14:45:00 | WARNING1 |
| | 2020-04-24T14:50:00 | DeviceID | State#Date | State |
| | | d#12345 | WARNING1#2020-04-24T14:50:00 | WARNING1 |
| | 2020-04-24T14:55:00 | DeviceID | State#Date | State |
| | | d#12345 | NORMAL#2020-04-24T14:55:00 | NORMAL |
| Sue | 2020-04-11T05:50:00 | DeviceID | State#Date | State |
| | | d#54321 | WARNING3#2020-04-11T05:50:00 | WARNING3 |
| | 2020-04-11T09:25:00 | DeviceID | State#Date | State |
| | | d#54321 | WARNING2#2020-04-11T09:25:00 | WARNING2 |
| | 2020-04-11T09:30:00 | DeviceID | State#Date | State |
| | | d#54321 | NORMAL#2020-04-11T09:30:00 | NORMAL |
| | 2020-04-27T16:10:00 | DeviceID | State#Date | State |
| | | d#11223 | WARNING4#2020-04-27T16:10:00 | WARNING4 |
| | 2020-04-27T16:15:00 | DeviceID | State#Date | State |
| | | d#11223 | WARNING4#2020-04-27T16:15:00 | WARNING4 |

## GSI-2

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| Partition key: EscalatedTo | Sort key: State#Date | | | | |
| | | DeviceID | Operator | Date | State |
| Sara | WARNING4#2020-04-27T16:15:00 | d#11223 | Sue | 2020-04-27T16:15:00 | WARNING4 |

### 이 스키마 설계와 함께 NoSQL Workbench 사용

이 최종 스키마를 DynamoDB 데이터 모델링, 데이터 시각화, 쿼리 개발 기능을 제공하는 시각적 도구인 NoSQL Workbench로 가져와서 새 프로젝트를 추가로 탐색하고 편집할 수 있습니다. 시작하려면 다음 단계를 따릅니다.

1. NoSQL Workbench 다운로드 자세한 내용은 the section called “다운로드” 섹션을 참조하세요.
2. 위에 나열된 JSON 스키마 파일을 다운로드합니다. 이 파일은 이미 NoSQL Workbench 모델 형식으로 되어 있습니다.
3. JSON 스키마 파일을 NoSQL Workbench로 가져옵니다. 자세한 내용은 the section called “기존 모델 가져오기” 섹션을 참조하세요.
4. NOSQL Workbench로 가져온 후 데이터 모델을 편집할 수 있습니다. 자세한 내용은 the section called “기존 모델 편집” 섹션을 참조하세요.
5. 데이터 모델을 시각화하거나, 샘플 데이터를 추가하거나, CSV 파일에서 샘플 데이터를 가져오려면 NoSQL Workbench의 Data Visualizer 기능을 사용하세요.

## DynamoDB를 온라인 상점용 데이터 스토어로 사용

이 사용 사례에서는 DynamoDB를 온라인 상점(또는 온라인 매장)의 데이터 스토어로 사용하는 방법을 설명합니다.

### 사용 사례

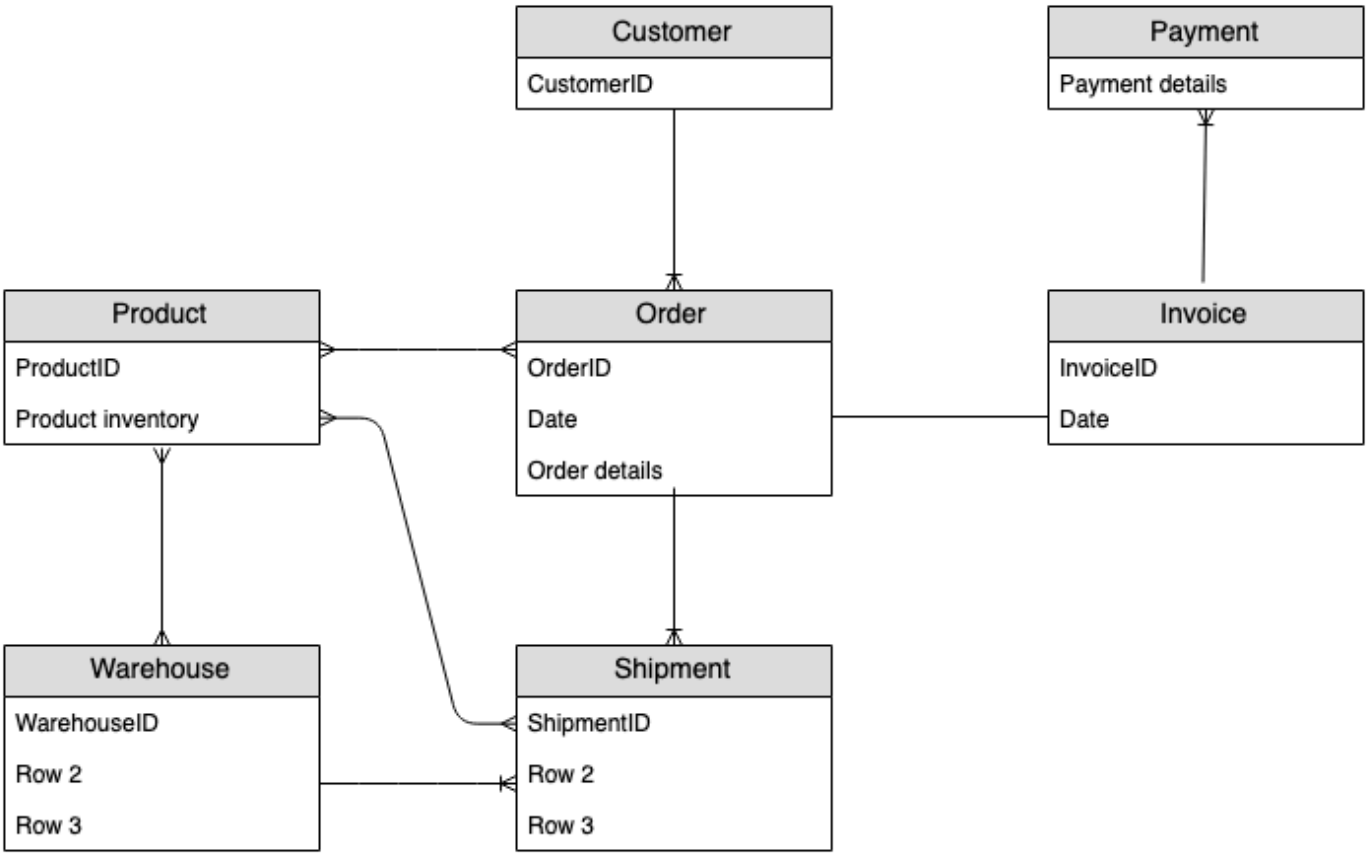
온라인 매장을 통해 사용자는 다양한 제품을 둘러보고 최종적으로 구매할 수 있습니다. 생성된 청구서를 기반으로 고객은 할인 코드 또는 기프트 카드를 사용하여 결제한 후 나머지 금액을 신용 카드로 결제할 수 있습니다. 구매한 제품은 여러 창고 중 한 곳에서 픽업되어 제공된 주소로 배송됩니다. 온라인 매장의 일반적인 액세스 패턴은 다음과 같습니다.

- 지정된 customerId의 고객 가져오기

- 지정된 productId의 제품 가져오기
- 지정된 warehouseId의 창고 가져오기
- productId를 기준으로 모든 창고의 제품 재고 가져오기
- 지정된 orderId의 주문 가져오기
- 지정된 orderId의 모든 제품 가져오기
- 지정된 orderId의 청구서 가져오기
- 지정된 orderId의 모든 배송물 가져오기
- 지정된 날짜 범위에서 지정된 productId의 모든 주문 가져오기
- 지정된 invoiceId의 청구서 가져오기
- 지정된 invoiceId에 대한 모든 결제 가져오기
- 지정된 shipmentId의 배송 세부 정보 가져오기
- 지정된 warehouseId의 모든 배송물 가져오기
- 지정된 warehouseId의 모든 제품 재고 가져오기
- 지정된 날짜 범위의 특정 customerId에 대한 모든 청구서 가져오기
- 지정된 날짜 범위의 특정 customerId가 주문한 모든 제품 가져오기

## 엔터티 관계 다이어그램

다음은 DynamoDB를 온라인 상점용 데이터 스토어로 모델링하는 데 사용할 엔터티 관계 다이어그램(ERD)입니다.




## 액세스 패턴

DynamoDB를 온라인 상점의 데이터 스토어로 사용할 때 고려할 액세스 패턴은 다음과 같습니다.

1. `getCustomerByCustomerId`
2. `getProductByProductId`
3. `getWarehouseByWarehouseId`
4. `getProductInventoryByProductId`
5. `getOrderDetailsByOrderId`
6. `getProductByOrderId`
7. `getInvoiceByOrderId`
8. `getShipmentByOrderId`
9. `getOrderByProductIdForDateRange`
10. `getInvoiceByInvoiceId`

11getPaymentByInvoiceId

12getShipmentDetailsByShipmentId

13getShipmentByWarehouseId

14getProductInventoryByWarehouseId

15getInvoiceByCustomerIdForDateRange

16getProductsByCustomerIdForDateRange

## 스키마 설계 진화

DynamoDB용 NoSQL Workbench를 사용하여 AnOnlineShop_1.json을 가져와 AnOnlineShop이라는 새 데이터 모델과 OnlineShop이라는 새 테이블을 생성합니다. 파티션 키와 정렬 키에는 일반 이름 PK 및 SK를 사용합니다. 이는 동일한 테이블에 다양한 유형의 엔터티를 저장하기 위해 사용되는 방법입니다.

1단계: 액세스 패턴 1(**getCustomerByCustomerId**) 처리

AnOnlineShop_2.json을 가져와 액세스 패턴 1(getCustomerByCustomerId)을 처리합니다. 일부 엔터티는 다른 엔터티와 관계가 없으므로 해당 엔터티에 대해 동일한 PK 및 SK 값을 사용합니다. 예제 데이터에서 키는 나중에 추가될 다른 엔터티와 customerId를 구별하기 위해 접두사 c#을 사용합니다. 이 방법은 다른 엔티티에 대해서도 반복됩니다.

이 액세스 패턴을 처리하기 위해 GetItem 작업을 PK=customerId 및 SK=customerId와 함께 사용할 수 있습니다.

2단계: 액세스 패턴 2(**getProductByProductId**) 처리

AnOnlineShop_3.json을 가져와 product 엔터티에 대한 액세스 패턴 2(getProductByProductId)를 처리합니다. 제품 엔터티에는 접두사 p#이 붙고 동일한 정렬 키 속성이 customerID와 productID를 저장하는 데 사용되었습니다. 일반 이름 지정 및 수직 파티셔닝을 통해 효과적인 단일 테이블 설계를 위한 항목 컬렉션을 생성할 수 있습니다.

이 액세스 패턴을 처리하기 위해 GetItem 작업을 PK=productId 및 SK=productId와 함께 사용할 수 있습니다.

3단계: 액세스 패턴 3(**getWarehouseByWarehouseId**) 처리

AnOnlineShop_4.json을 가져와 warehouse 엔터티에 대한 액세스 패턴 3(getWarehouseByWarehouseId)을 처리합니다. 현재 동일한 테이블에 customer, product 및

warehouse 엔터티가 추가되어 있습니다. 이들은 접두사와 EntityType 속성을 사용하여 구별됩니다. 유형 속성(또는 접두사 이름 지정)은 모델의 가독성을 향상시킵니다. 동일한 속성에 서로 다른 엔터티에 대한 영숫자 ID를 저장하면 가독성에 영향을 미칩니다. 이러한 식별자가 없으면 한 엔터티와 다른 엔터티를 구별하기가 어려울 수 있습니다.

이 액세스 패턴을 처리하기 위해 GetItem 작업을 PK=warehouseId 및 SK=warehouseId와 함께 사용할 수 있습니다.

기본 테이블:

| Primary key | | Attributes | | |
|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | |
| c#12345 | c#12345 | EntityType | Email | Name |
| | | customer | samaneh@example.com | Samaneh Utter |
| p#12345 | p#12345 | EntityType | Detail | Price |
| | | product | {"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}} | 100 |
| w#12345 | w#12345 | EntityType | Address | |
| | | warehouse | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}} | |

4단계: 액세스 패턴 4(**getProductInventoryByProductId**) 처리

AnOnlineShop_5.json을 가져와 액세스 패턴 4(getProductInventoryByProductId)를 처리합니다. warehouseItem 엔터티는 각 창고의 제품 수를 추적하는 데 사용됩니다. 이 항목은 일반적으로 제품이 창고에서 추가되거나 제거될 때 업데이트됩니다. ERD에서 볼 수 있듯이 product와 warehouse 간에는 다대다 관계가 있습니다. 여기서는 product에서 warehouse로의 일대다 관계를 warehouseItem으로 모델링합니다. 나중에 warehouse에서 product로의 일대다 관계도 모델링합니다.

액세스 패턴 4는 PK=ProductId 및 SK begins_with “w#“에 대한 쿼리를 통해 처리될 수 있습니다.

begins_with() 및 정렬 키에 적용할 수 있는 기타 표현식에 대한 자세한 내용은 키 조건 표현식을 참조하세요.

기본 테이블:

| Primary key | | Attributes | | |
|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | |
| c#12345 | c#12345 | EntityType | Email | Name |
| | | customer | samaneh@example.com | Samaneh Utter |
| | p#12345 | EntityType | Detail | Price |
| | | product | {"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}} | 100 |
| p#12345 | w#12345 | EntityType | Quantity | |
| | | warehouseItem | 50 | |
| w#12345 | w#12345 | EntityType | Address | |
| | | warehouse | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}} | |

5단계: 액세스 패턴 5(**`getOrderDetailsByOrderId`**) 및 6(**`getProductByOrderId`**) 처리

AnOnlineShop_6.json을 가져와 테이블에 더 많은 `customer`, `product` 및 `warehouse`를 추가합니다. 그런 다음 AnOnlineShop_7.json을 가져와 액세스 패턴 5(`getOrderDetailsByOrderId`) 및 6(`getProductByOrderId`)을 처리할 수 있도록 `order`에 대한 항목 컬렉션을 생성합니다. orderItem 엔터티로 모델링된 `order`와 `product` 간의 일대다 관계를 볼 수 있습니다.

액세스 패턴 5(`getOrderDetailsByOrderId`)를 처리하기 위해 `PK=orderId`를 사용하여 테이블을 쿼리합니다. 그러면 `customerId` 및 주문한 제품을 포함하여 주문에 대한 모든 정보가 제공됩니다.

기본 테이블:

| Primary key | | Attributes | | |
|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | |
| o#12345 | c#12345 | EntityType | Date | |
| | | order | 2020-06-21T19:10:00 | |
| | p#12345 | EntityType | Price | Quantity |
| | | orderItem | 100 | 2 |
| | p#99887 | EntityType | Price | Quantity |
| | | orderItem | 40 | 5 |

액세스 패턴 6(`getProductByOrderId`)을 해결하려면 `order`의 제품만 읽어야 합니다. 이를 위해 `PK=orderId` 및 `SK begins_with "p#"`을 사용하여 테이블을 쿼리합니다.

기본 테이블:

| Primary key | | Attributes | | |
|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | |
| | c#12345 | EntityType | Date | |
| | | order | 2020-06-21T19:10:00 | |
| o#12345 | p#12345 | EntityType | Price | Quantity |
| | | orderItem | 100 | 2 |
| | p#99887 | EntityType | Price | Quantity |
| | | orderItem | 40 | 5 |

6단계: 액세스 패턴 7(**`getInvoiceByOrderId`**) 처리

AnOnlineShop_8.json을 가져와 주문 항목 컬렉션에 `invoice` 엔터티를 추가하고 액세스 패턴 7(`getInvoiceByOrderId`)을 처리합니다. 이 액세스 패턴을 처리하기 위해 쿼리 작업을 `PK=orderId` 및 `SK begins_with "i#"`과 함께 사용할 수 있습니다.

기본 테이블:

| Primary key | | Attributes | | |
|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | |
| | c#12345 | EntityType | Date | |
| | | order | 2020-06-21T19:10:00 | |
| o#12345 | i#55443 | EntityType | Amount | Date |
| | | invoice | 400 | 2020-06-21T19:18:00 |
| | p#12345 | EntityType | Price | Quantity |
| | | orderItem | 100 | 2 |
| | p#99887 | EntityType | Price | Quantity |
| | | orderItem | 40 | 5 |

7단계: 액세스 패턴 8(**`getShipmentByOrderId`**) 처리

AnOnlineShop_9.json을 가져와 주문 항목 컬렉션에 shipment 엔터티를 추가하여 액세스 패턴 8(getShipmentByOrderId)을 처리합니다. 단일 테이블 설계에 더 많은 유형의 엔터티를 추가하여 동일한 수직 파티셔닝 모델을 확장하고 있습니다. 주문 항목 컬렉션에 포함된 order 엔터티가 shipment, orderItem 및 invoice 엔터티와 갖는 다양한 관계를 확인해 보세요.

orderId를 기준으로 배송물을 가져오기 위해 PK=orderId 및 SK begins_with "sh#"을 사용하여 쿼리 작업을 수행할 수 있습니다.

기본 테이블:

| Primary key | | Attributes | | | | |
|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | | |
| o#12345 | c#12345 | EntityType | Date | | | |
| | | order | 2020-06-21T19:10:00 | | | |
| | i#55443 | EntityType | Amount | Date | | |
| | | invoice | 400 | 2020-06-21T19:18:00 | | |
| | p#12345 | EntityType | Price | Quantity | | |
| | | orderItem | 100 | 2 | | |
| | p#99887 | EntityType | Price | Quantity | | |
| | | orderItem | 40 | 5 | | |
| | sh#88899 | EntityType | Address | Type | Date | Warehouseld |
| | | shipment | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}} | Express | 2020-06-22T08:20:00 | w#12376 |
| | sh#98765 | EntityType | Address | Type | Date | Warehouseld |
| | | shipment | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}} | Express | 2020-06-22T10:20:00 | w#12345 |

8단계: 액세스 패턴 9(**getOrderByProductIdForDateRange**) 처리

이전 단계에서 주문 항목 컬렉션을 만들었습니다. 이 액세스 패턴에는 전체 테이블을 스캔하고 관련 레코드를 필터링하여 대상 항목을 가져와야 하는 새로운 조회 차원(ProductID 및 Date)이 있습니다.

이 액세스 패턴을 처리하려면 글로벌 보조 인덱스(GSI)를 생성해야 합니다. AnOnlineShop_10.json을 가져와 여러 주문 항목 컬렉션에서 `orderItem` 데이터를 검색할 수 있는 GSI를 사용하여 새 항목 컬렉션을 만듭니다. 이제 데이터에는 각각 `GSI1`의 파티션 키와 정렬 키가 될 `GSI1-PK` 및 `GSI1-SK`가 있습니다.

DynamoDB는 GSI의 주요 속성이 포함된 항목을 테이블에서 GSI에 자동으로 채웁니다. GSI에 추가 삽입을 수동으로 수행할 필요가 없습니다.

액세스 패턴 9를 처리하려면 `GSI1-PK=productId` 및 `GSI1SK between (date1, date2)`를 사용하여 `GSI1`에 대해 쿼리를 수행합니다.

기본 테이블:

| Primary key | | Attributes | | | | |
|---|---|---|---|---|---|---|
| **Partition key: PK** | **Sort key: SK** | | | | | |
| o#12345 | p#12345 | EntityType | GSI1-PK | GSI1-SK | Price | Quantity |
| | | orderItem | p#12345 | 2020-06-21T19:18:00 | 100 | 2 |
| | p#99887 | EntityType | GSI1-PK | GSI1-SK | Price | Quantity |
| | | orderItem | p#99887 | 2020-06-21T19:20:00 | 40 | 5 |

GSI1:

| Primary key | | Attributes | | | | |
|---|---|---|---|---|---|---|
| **Partition key: GSI1-PK** | **Sort key: GSI1-SK** | | | | | |
| p#12345 | 2020-06-21T19:18:00 | PK | SK | EntityType | Quantity | Price |
| | | o#12345 | p#12345 | orderItem | 2 | 100 |
| p#99887 | 2020-06-21T19:20:00 | PK | SK | EntityType | Quantity | Price |
| | | o#12345 | p#99887 | orderItem | 5 | 40 |

9단계: 액세스 패턴 10(**`getInvoiceByInvoiceId`**) 및 11(**`getPaymentByInvoiceId`**) 처리

AnOnlineShop_11.json을 가져와서 `invoice`와 관련된 액세스 패턴 10(`getInvoiceByInvoiceId`) 및 11(`getPaymentByInvoiceId`)을 처리합니다. 이는 서로 다른 두 가지 액세스 패턴이지만 동일한 키 조건을 사용하여 구현됩니다. `Payments`는 `invoice` 엔터티에 대한 맵 데이터 유형의 속성으로 정의됩니다.

> **Note**
>
> GSI1-PK 및 GSI1-SK는 여러 엔터티에 대한 정보를 저장하도록 오버로드되므로 동일한 GSI에서 여러 액세스 패턴을 제공할 수 있습니다. GSI 오버로딩에 대한 자세한 내용은 DynamoDB의 글로벌 보조 인덱스 오버로딩 섹션을 참조하세요.

액세스 패턴 10 및 11을 처리하려면 GSI1-PK=invoiceId 및 GSI1-SK=invoiceId를 사용하여 GSI1에 대해 쿼리를 수행합니다.

GSI1:

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Partition key: GSI1-PK | Sort key: GSI1-SK | PK | SK | EntityType | GSI2-PK | GSI2-SK | Detail | Amount | Date |
| i#55443 | i#55443 | o#12345 | i#55443 | invoice | c#12345 | i#2020-06-21T19:18:00 | {"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}} | 400 | 2020-06-21T19:18:00 |

10단계: 액세스 패턴 12(**getShipmentDetailsByShipmentId**) 및 13(**getShipmentByWarehouseId**) 처리

AnOnlineShop_12.json을 가져와 액세스 패턴 12(getShipmentDetailsByShipmentId) 및 13(getShipmentByWarehouseId)을 처리합니다.

단일 쿼리 작업으로 주문에 대한 모든 세부 정보를 검색할 수 있도록 shipmentItem 엔터티가 기본 테이블의 주문 항목 컬렉션에 추가됩니다.

기본 테이블:

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Partition key: PK | Sort key: SK | | | | | | | | |
| o#12345 | sh#88899 | EntityType | GSI1-PK | GSI1-SK | GSI2-PK | GSI2-SK | Address | Type | Date |
| | | shipment | sh#88899 | sh#88899 | w#12376 | sh#88899 | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}} | Express | 2020-06-22T08:20:00 |
| | sh#98765 | EntityType | GSI1-PK | GSI1-SK | GSI2-PK | GSI2-SK | Address | Type | Date |
| | | shipment | sh#98765 | sh#98765 | w#12345 | sh#98765 | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}} | Express | 2020-06-22T10:20:00 |
| | shp#12345 | EntityType | GSI1-PK | GSI1-SK | Quantity | | | | |
| | | shipmentItem | sh#98765 | p#99887 | 3 | | | | |
| | shp#54321 | EntityType | GSI1-PK | GSI1-SK | Quantity | | | | |
| | | shipmentItem | sh#88899 | p#99887 | 2 | | | | |
| | shp#55555 | EntityType | GSI1-PK | GSI1-SK | Quantity | | | | |
| | | shipmentItem | sh#98765 | p#12345 | 2 | | | | |

GSI1 파티션 및 정렬 키는 이미 shipment와 shipmentItem 간의 일대다 관계를 모델링하는 데 사용되었습니다. 액세스 패턴 12(getShipmentDetailsByShipmentId)를 처리하려면 GSI1-PK=shipmentId 및 GSI1-SK=shipmentId를 사용하여 GSI1에 대해 쿼리를 수행합니다.

GSI1:

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Partition key: GSI1-PK | Sort key: GSI1-SK | | | | | | | | |
| | p#99887 | PK | SK | EntityType | Quantity | | | | |
| | | o#12345 | shp#54321 | shipmentItem | 2 | | | | |
| sh#88899 | sh#88899 | PK | SK | EntityType | GSI2-PK | GSI2-SK | Address | Type | Date |
| | | o#12345 | sh#88899 | shipment | w#12376 | sh#88899 | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}} | Express | 2020-06-22T08:20:00 |
| sh#98765 | p#12345 | PK | SK | EntityType | Quantity | | | | |
| | | o#12345 | shp#55555 | shipmentItem | 2 | | | | |
| | p#99887 | PK | SK | EntityType | Quantity | | | | |
| | | o#12345 | shp#12345 | shipmentItem | 3 | | | | |
| | sh#98765 | PK | SK | EntityType | GSI2-PK | GSI2-SK | Address | Type | Date |
| | | o#12345 | sh#98765 | shipment | w#12345 | sh#98765 | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}} | Express | 2020-06-22T10:20:00 |

액세스 패턴 13(`getShipmentByWarehouseId`)에 대해 `warehouse`와 `shipment` 간의 새로운 일대다 관계를 모델링하려면 또 다른 GSI(`GSI2`)를 만들어야 합니다. 이 액세스 패턴을 처리하려면 `GSI2-PK=warehouseId` 및 `GSI2-SK begins_with “sh#”`를 사용하여 `GSI2`에 대해 쿼리를 수행합니다.

GSI2:

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Partition key: GSI2-PK | Sort key: GSI2-SK | | | | | | | | |
| | | PK | SK | EntityType | GSI1-PK | GSI1-SK | Address | Type | Date |
| w#12376 | sh#88899 | o#12345 | sh#88899 | shipment | sh#88899 | sh#88899 | {"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}} | Express | 2020-06-22T08:20:00 |
| | | PK | SK | EntityType | GSI1-PK | GSI1-SK | Address | Type | Date |
| w#12345 | sh#98765 | o#12345 | sh#98765 | shipment | sh#98765 | sh#98765 | {"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}} | Express | 2020-06-22T10:20:00 |

11: 액세스 패턴 14(**`getProductInventoryByWarehouseId`**)
15(**`getInvoiceByCustomerIdForDateRange`**) 및
16(**`getProductsByCustomerIdForDateRange`**) 처리

AnOnlineShop_13.json을 가져와서 다음 액세스 패턴 세트와 관련된 데이터를 추가합니다. 액세스 패턴 14(`getProductInventoryByWarehouseId`)를 처리하려면 `GSI2-PK=warehouseId` 및 `GSI2-SK begins_with "p#"`를 사용하여 `GSI2`에 대해 쿼리를 수행합니다.

GSI2:

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Partition key: GSI2-PK | Sort key: GSI2-SK | | | | | | | | |
| w#12345 | p#12345 | PK | SK | EntityType | Quantity | | | | |
| | | p#12345 | w#12345 | warehouseItem | 50 | | | | |
| | p#99887 | PK | SK | EntityType | Quantity | | | | |
| | | p#99887 | w#12345 | warehouseItem | 4 | | | | |
| c#12345 | i#2020-06-21T19:18:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Detail | Amount | Date |
| | | o#12345 | i#55443 | invoice | i#55443 | i#55443 | {"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}} | 400 | 2020-06-21T19:18:00 |
| | p#2020-06-21T19:18:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Price | Quantity | |
| | | o#12345 | p#12345 | orderItem | p#12345 | 2020-06-21T19:18:00 | 100 | 2 | |
| | p#2020-06-21T19:20:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Price | Quantity | |
| | | o#12345 | p#99887 | orderItem | p#99887 | 2020-06-21T19:20:00 | 40 | 5 | |

액세스 패턴 15(`getInvoiceByCustomerIdForDateRange`)를 처리하려면 `GSI2-PK=customerId` 및 `GSI2-SK between (i#date1, i#date2)`를 사용하여 GSI2에 대해 쿼리를 수행합니다.

GSI2:

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Partition key: GSI2-PK | Sort key: GSI2-SK | | | | | | | | |
| w#12345 | p#12345 | PK | SK | EntityType | Quantity | | | | |
| | | p#12345 | w#12345 | warehouseItem | 50 | | | | |
| | p#99887 | PK | SK | EntityType | Quantity | | | | |
| | | p#99887 | w#12345 | warehouseItem | 4 | | | | |
| c#12345 | i#2020-06-21T19:18:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Detail | Amount | Date |
| | | o#12345 | i#55443 | invoice | i#55443 | i#55443 | {"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}} | 400 | 2020-06-21T19:18:00 |
| | p#2020-06-21T19:18:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Price | Quantity | |
| | | o#12345 | p#12345 | orderItem | p#12345 | 2020-06-21T19:18:00 | 100 | 2 | |
| | p#2020-06-21T19:20:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Price | Quantity | |
| | | o#12345 | p#99887 | orderItem | p#99887 | 2020-06-21T19:20:00 | 40 | 5 | |

액세스 패턴 16(`getProductsByCustomerIdForDateRange`)을 처리하려면 `GSI2-PK=customerId` 및 `GSI2-SK between (p#date1, p#date2)`를 사용하여 GSI2에 대해 쿼리를 수행합니다.

GSI2:

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Partition key: GSI2-PK | Sort key: GSI2-SK | | | | | | | | |
| w#12345 | p#12345 | PK | SK | EntityType | Quantity | | | | |
| | | p#12345 | w#12345 | warehouseItem | 50 | | | | |
| | p#99887 | PK | SK | EntityType | Quantity | | | | |
| | | p#99887 | w#12345 | warehouseItem | 4 | | | | |
| c#12345 | i#2020-06-21T19:18:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Detail | Amount | Date |
| | | o#12345 | i#55443 | invoice | i#55443 | i#55443 | {"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}} | 400 | 2020-06-21T19:18:00 |
| | p#2020-06-21T19:18:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Price | Quantity | |
| | | o#12345 | p#12345 | orderItem | p#12345 | 2020-06-21T19:18:00 | 100 | 2 | |
| | p#2020-06-21T19:20:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Price | Quantity | |
| | | o#12345 | p#99887 | orderItem | p#99887 | 2020-06-21T19:20:00 | 40 | 5 | |

**Note**

NoSQL Workbench에서 패싯은 애플리케이션의 서로 다른 DynamoDB 데이터 액세스 패턴을 나타냅니다. 패싯을 사용하면 패싯의 제약 조건을 충족하지 않는 레코드를 확인할 필요 없이 테이블에서 데이터의 하위 집합을 볼 수 있습니다. 패싯은 시각적 데이터 모델링 도구로 간주되며 DynamoDB에서 사용 가능한 구성체로 존재하지 않습니다. 패싯은 순전히 액세스 패턴 모델링에 도움이 되기 때문입니다.

AnOnlineShop_facets.json을 가져와 이 사용 사례의 패싯을 확인합니다.

모든 액세스 패턴과 스키마 설계에서 이를 처리하는 방법이 아래 표에 요약되어 있습니다.

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 |
|---|---|---|---|---|
| getCustomerByCustomerId | 기본 테이블 | GetItem | PK=customerId | SK=customerId |
| getProductByProductId | 기본 테이블 | GetItem | PK=productId | SK=productId |
| getWarehouseByWarehouseId | 기본 테이블 | GetItem | PK=warehouseId | SK=warehouseId |
| getProductInventoryByProductId | 기본 테이블 | 쿼리 | PK=productId | SK begins_with "w#" |
| getOrderDetailsByOrderId | 기본 테이블 | 쿼리 | PK=orderId | |
| getProductByOrderId | 기본 테이블 | 쿼리 | PK=orderId | SK begins_with "p#" |
| getInvoiceByOrderId | 기본 테이블 | 쿼리 | PK=orderId | SK begins_with "i#" |
| getShipmentByOrderId | 기본 테이블 | 쿼리 | PK=orderId | SK begins_with "sh#" |
| getOrderByProductIdForDateRange | GSI1 | Query | PK=productId | SK between date1 and date2 |
| getInvoiceByInvoiceId | GSI1 | Query | PK=invoiceId | SK=invoiceId |

| 액세스 패턴 | 기본 테이블/GSI/LSI | Operation | 파티션 키 값 | 정렬 키 값 |
|---|---|---|---|---|
| getPaymentByInvoiceId | GSI1 | Query | PK=invoiceId | SK=invoiceId |
| getShipmentDetailsByShipmentId | GSI1 | Query | PK=shipmentId | SK=shipmentId |
| getShipmentByWarehouseId | GSI2 | Query | PK=warehouseId | SK begins_with "sh#" |
| getProductInventoryByWarehouseId | GSI2 | Query | PK=warehouseId | SK begins_with "p#" |
| getInvoiceByCustomerIdForDateRange | GSI2 | Query | PK=customerId | SK between i#date1 and i#date2 |
| getProductsByCustomerIdForDateRange | GSI2 | Query | PK=customerId | SK between p#date1 and p#date2 |

온라인 상점 최종 스키마

다음은 최종 스키마 설계입니다. 이 스키마 설계를 JSON 파일로 다운로드하려면 GitHub의 DynamoDB 설계 패턴을 참조하세요.

기본 테이블

| Primary key | | Attributes | | | |
|---|---|---|---|---|---|
| **Partition key: PK** | **Sort key: SK** | | | | |
| c#12345 | c#12345 | EntityType | Email | Name | |
| | | customer | samaneh@example.com | Samaneh | |
| c#23456 | c#23456 | EntityType | Email | Name | |
| | | customer | kathleen@example.com | Kathleen | |
| c#54321 | c#54321 | EntityType | Email | Name | |
| | | customer | henrik@example.com | Henrik | |
| p#12345 | p#12345 | EntityType | Detail | Price | |
| | | product | {"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}} | 100 | |
| | w#12345 | EntityType | GSI2-PK | GSI2-SK | Quantity |
| | | warehouseItem | w#12345 | p#12345 | 50 |
| p#99887 | p#99887 | EntityType | Detail | Price | |
| | | product | {"Name":{"S":"The Book"},"Description":{"S":"The best book ever"}} | 40 | |
| | w#12345 | EntityType | GSI2-PK | GSI2-SK | Quantity |
| | | warehouseItem | w#12345 | p#99887 | 4 |
| | w#12376 | EntityType | Quantity | | |
| | | warehouseItem | 4 | | |
| w#12345 | w#12345 | EntityType | Address | | |
| | | warehouse | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}} | | |
| w#12376 | w#12376 | EntityType | Address | | |
| | | warehouse | {"Country":{"S":"Sweden"},"County": | | |

GSI1

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Partition key: GSI1-PK** | **Sort key: GSI1-SK** | | | | | | | | |
| p#12345 | 2020-06-21T19:18:00 | PK | SK | EntityType | GSI2-PK | GSI2-SK | Price | Quantity | |
| | | o#12345 | p#12345 | orderItem | c#12345 | 2020-06-21T19:18:00 | 100 | 2 | |
| p#99887 | 2020-06-21T19:20:00 | PK | SK | EntityType | GSI2-PK | GSI2-SK | Price | Quantity | |
| | | o#12345 | p#99887 | orderItem | c#12345 | 2020-06-21T19:20:00 | 40 | 5 | |
| i#55443 | i#55443 | PK | SK | EntityType | GSI2-PK | GSI2-SK | Detail | Amount | Date |
| | | o#12345 | i#55443 | invoice | c#12345 | 2020-06-21T19:18:00 | {"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}} | 400 | 2020-06-21T19:18:00 |
| sh#88899 | p#99887 | PK | SK | EntityType | Quantity | | | | |
| | | o#12345 | shp#54321 | shipmentItem | 2 | | | | |
| | sh#88899 | PK | SK | EntityType | GSI2-PK | GSI2-SK | Address | Type | Date |
| | | o#12345 | sh#88899 | shipment | w#12376 | sh#88899 | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}} | Express | 2020-06-22T08:20:00 |
| sh#98765 | p#12345 | PK | SK | EntityType | Quantity | | | | |
| | | o#12345 | shp#55555 | shipmentItem | 2 | | | | |
| | p#99887 | PK | SK | EntityType | Quantity | | | | |
| | | o#12345 | shp#12345 | shipmentItem | 3 | | | | |
| | sh#98765 | PK | SK | EntityType | GSI2-PK | GSI2-SK | Address | Type | Date |
| | | o#12345 | sh#98765 | shipment | w#12345 | sh#98765 | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}} | Express | 2020-06-22T10:20:00 |

GSI2

| Primary key | | Attributes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Partition key: GSI2-PK** | **Sort key: GSI2-SK** | | | | | | | | |
| w#12345 | p#12345 | PK | SK | EntityType | Quantity | | | | |
| | | p#12345 | w#12345 | warehouseItem | 50 | | | | |
| | p#99887 | PK | SK | EntityType | Quantity | | | | |
| | | p#99887 | w#12345 | warehouseItem | 4 | | | | |
| | sh#98765 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Address | Type | Date |
| | | o#12345 | sh#98765 | shipment | sh#98765 | sh#98765 | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}} | Express | 2020-06-22T10:20:00 |
| c#12345 | 2020-06-21T19:18:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Price | Quantity | |
| | | o#12345 | p#12345 | orderItem | p#12345 | 2020-06-21T19:18:00 | 100 | 2 | |
| | 2020-06-21T19:18:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Detail | Amount | Date |
| | | o#12345 | i#55443 | invoice | i#55443 | i#55443 | {"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}} | 400 | 2020-06-21T19:18:00 |
| | 2020-06-21T19:20:00 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Price | Quantity | |
| | | o#12345 | p#99887 | orderItem | p#99887 | 2020-06-21T19:20:00 | 40 | 5 | |
| w#12376 | sh#88899 | PK | SK | EntityType | GSI1-PK | GSI1-SK | Address | Type | Date |
| | | o#12345 | sh#88899 | shipment | sh#88899 | sh#88899 | {"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}} | Express | 2020-06-22T08:20:00 |

### 이 스키마 설계와 함께 NoSQL Workbench 사용

이 최종 스키마를 DynamoDB 데이터 모델링, 데이터 시각화, 쿼리 개발 기능을 제공하는 시각적 도구인 NoSQL Workbench로 가져와서 새 프로젝트를 추가로 탐색하고 편집할 수 있습니다. 시작하려면 다음 단계를 따릅니다.

1. NoSQL Workbench 다운로드 자세한 내용은 the section called “다운로드” 단원을 참조하십시오.
2. 위에 나열된 JSON 스키마 파일을 다운로드합니다. 이 파일은 이미 NoSQL Workbench 모델 형식으로 되어 있습니다.
3. JSON 스키마 파일을 NoSQL Workbench로 가져옵니다. 자세한 내용은 the section called “기존 모델 가져오기” 단원을 참조하십시오.
4. NOSQL Workbench로 가져온 후 데이터 모델을 편집할 수 있습니다. 자세한 내용은 the section called “기존 모델 편집” 단원을 참조하십시오.
5. 데이터 모델을 시각화하거나, 샘플 데이터를 추가하거나, CSV 파일에서 샘플 데이터를 가져오려면 NoSQL Workbench의 Data Visualizer 기능을 사용하세요.

## DynamoDB의 관계형 데이터 모델링 모범 사례

이 섹션에서는 Amazon DynamoDB의 관계형 데이터 모델링 모범 사례를 제공합니다. 먼저 기존 데이터 모델링 개념을 소개합니다. 그런 다음 기존 관계형 데이터베이스 관리 시스템에 비해 DynamoDB를 사용할 때의 이점, 즉 JOIN 작업의 필요성이 없어지고 오버헤드가 줄어드는 것에 대해 설명합니다.

또한 효율적으로 확장되는 DynamoDB 테이블을 설계하는 방법을 설명합니다. 마지막으로 DynamoDB에서 관계형 데이터를 모델링하는 방법에 대한 예를 제공합니다.

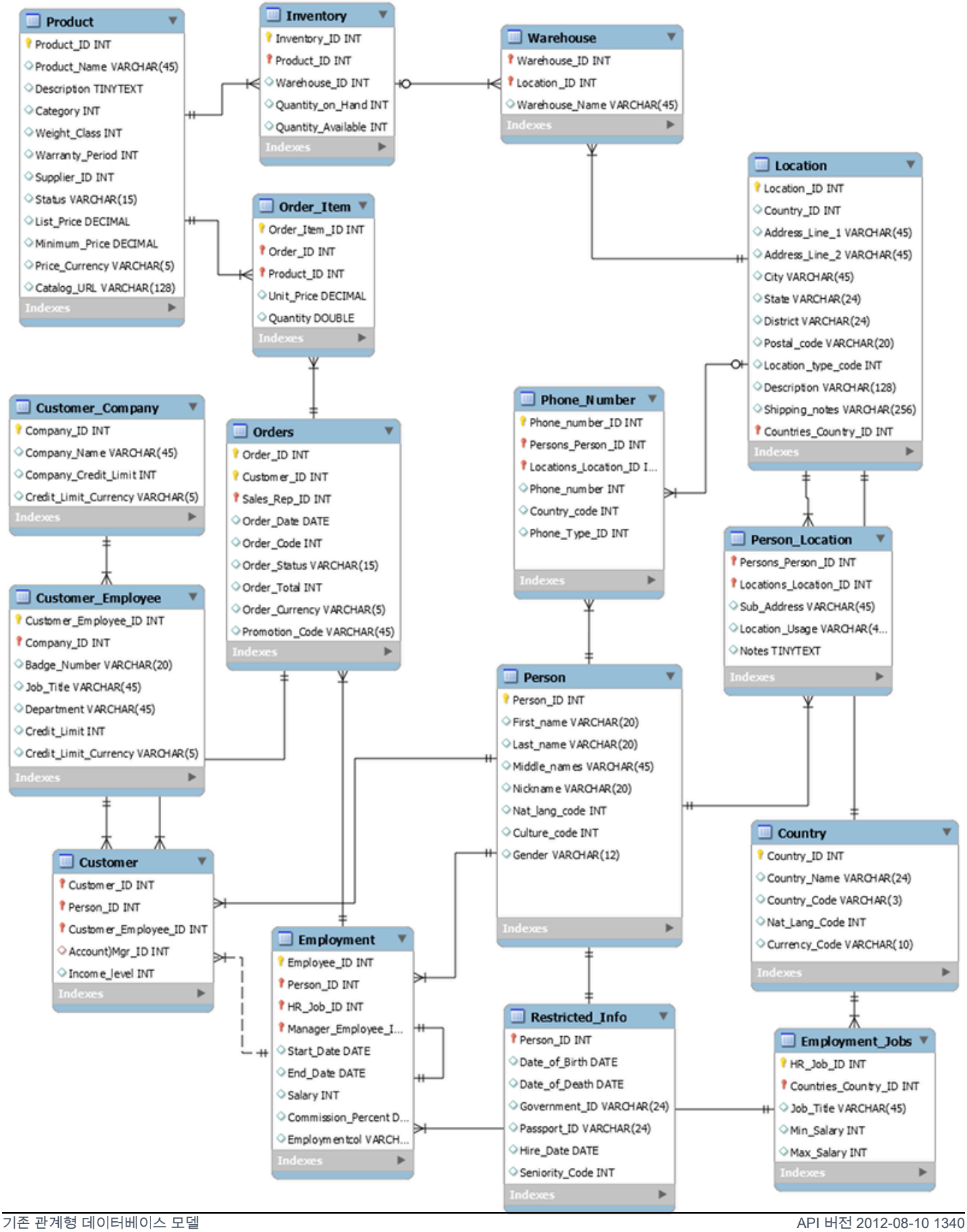
주제

- 기존 관계형 데이터베이스 모델
- DynamoDB를 통해 JOIN 작업의 필요성을 없애는 방법
- DynamoDB 트랜잭션이 쓰기 프로세스의 오버헤드를 제거하는 방법
- DynamoDB의 관계형 데이터를 모델링하는 첫 번째 단계
- DynamoDB의 관계형 데이터 모델링에 대한 예

# 기존 관계형 데이터베이스 모델

기존의 관계형 데이터베이스 관리 시스템(RDBMS)은 데이터를 정규화된 관계형 구조로 저장합니다. 관계형 데이터 모델의 목적은 정규화를 통해 데이터 중복을 줄여 참조 무결성을 지원하고 데이터 이상 현상을 줄이는 것입니다.

다음 스키마는 일반 주문 입력 애플리케이션을 위한 관계형 데이터 모델의 예입니다. 이 애플리케이션은 가상의 제조업체의 운영 및 비즈니스 지원 시스템을 뒷받침하는 HR 스키마를 지원합니다.


Product
Product_ID INT
Product_Name VARCHAR(45)
Description TINYTEXT
Category INT
Weight_Class INT
Warranty_Period INT
Supplier_ID INT
Status VARCHAR(15)
List_Price DECIMAL
Minimum_Price DECIMAL
Price_Currency VARCHAR(5)
Catalog_URL VARCHAR(128)
Indexes
Inventory
Inventory_ID INT
Product_ID INT
Warehouse_ID INT
Quantity_on_Hand INT
Quantity_Available INT
Indexes
Warehouse
Warehouse_ID INT
Location_ID INT
Warehouse_Name VARCHAR(45)
Indexes
Location
Location_ID INT
Country_ID INT
Address_Line_1 VARCHAR(45)
Address_Line_2 VARCHAR(45)
City VARCHAR(45)
State VARCHAR(24)
District VARCHAR(24)
Postal_code VARCHAR(20)
Location_type_code INT
Description VARCHAR(128)
Shipping_notes VARCHAR(256)
Countries_Country_ID INT
Indexes
Order_Item
Order_Item_ID INT
Order_ID INT
Product_ID INT
Unit_Price DECIMAL
Quantity DOUBLE
Indexes
Customer_Company
Company_ID INT
Company_Name VARCHAR(45)
Company_Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Orders
Order_ID INT
Customer_ID INT
Sales_Rep_ID INT
Order_Date DATE
Order_Code INT
Order_Status VARCHAR(15)
Order_Total INT
Order_Currency VARCHAR(5)
Promotion_Code VARCHAR(45)
Indexes
Phone_Number
Phone_number_ID INT
Persons_Person_ID INT
Locations_Location_ID I...
Phone_number INT
Country_code INT
Phone_Type_ID INT
Indexes
Person_Location
Persons_Person_ID INT
Locations_Location_ID INT
Sub_Address VARCHAR(45)
Location_Usage VARCHAR(4...
Notes TINYTEXT
Indexes
Customer_Employee
Customer_Employee_ID INT
Company_ID INT
Badge_Number VARCHAR(20)
Job_Title VARCHAR(45)
Department VARCHAR(45)
Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Person
Person_ID INT
First_name VARCHAR(20)
Last_name VARCHAR(20)
Middle_names VARCHAR(45)
Nickname VARCHAR(20)
Nat_lang_code INT
Culture_code INT
Gender VARCHAR(12)
Indexes
Country
Country_ID INT
Country_Name VARCHAR(24)
Country_Code VARCHAR(3)
Nat_Lang_Code INT
Currency_Code VARCHAR(10)
Indexes
Customer
Customer_ID INT
Person_ID INT
Customer_Employee_ID INT
Account)Mgr_ID INT
Income_level INT
Indexes
Employment
Employee_ID INT
Person_ID INT
HR_Job_ID INT
Manager_Employee_I...
Start_Date DATE
End_Date DATE
Salary INT
Commission_Percent D...
Employmentcol VARCH...
Indexes
Restricted_Info
Person_ID INT
Date_of_Birth DATE
Date_of_Death DATE
Government_ID VARCHAR(24)
Passport_ID VARCHAR(24)
Hire_Date DATE
Seniority_Code INT
Indexes
Employment_Jobs
HR_Job_ID INT
Countries_Country_ID INT
Job_Title VARCHAR(45)
Min_Salary INT
Max_Salary INT
Indexes

비관계형 데이터베이스 서비스인 DynamoDB는 기존의 관계형 데이터베이스 관리 시스템에 비해 많은 이점을 제공합니다.

## DynamoDB를 통해 JOIN 작업의 필요성을 없애는 방법

RDBMS는 구조 쿼리 언어(SQL)를 사용하여 데이터를 애플리케이션에 반환합니다. 데이터 모델의 정규화로 인해 이러한 쿼리에서는 일반적으로 `JOIN` 연산자를 사용하여 하나 이상의 테이블에서 데이터를 결합해야 합니다.

예를 들어, 각 항목을 출하할 수 있는 모든 창고의 재고량을 기준으로 정렬된 구매 주문 항목 목록을 생성하려면, 앞서 스키마에 대해 다음 SQL 쿼리를 발행할 수 있습니다.

```
SELECT * FROM Orders
  INNER JOIN Order_Items ON Orders.Order_ID = Order_Items.Order_ID
  INNER JOIN Products ON Products.Product_ID = Order_Items.Product_ID
  INNER JOIN Inventories ON Products.Product_ID = Inventories.Product_ID
  ORDER BY Quantity_on_Hand DESC
```

이런 유형의 SQL 쿼리는 데이터에 유연하게 액세스 할 수 있는 API를 제공할 수 있지만, 상당히 많은 처리량이 필요합니다. 쿼리의 각 조인은 각 테이블의 데이터를 스테이징한 다음 조합하여 결과 집합을 반환해야 하므로 쿼리의 런타임 복잡성이 증가합니다.

쿼리를 실행하는 데 걸리는 시간에 영향을 줄 수 있는 추가 요소로는 테이블 크기 및 조인되는 열의 인덱스 여부가 있습니다. 앞의 쿼리는 여러 테이블을 대상으로 복잡한 쿼리를 시작한 후, 결과 세트를 분류합니다.

`JOINs`에 대한 필요를 없애는 것이 NoSQL 데이터 모델링의 핵심입니다. 이것이 바로 Amazon.com을 지원하기 위해 DynamoDB를 구축한 이유이자 DynamoDB가 모든 규모에서 일관된 성능을 제공할 수 있는 이유입니다. SQL 쿼리 및 `JOINs`의 런타임 복잡성을 고려할 때 RBDMS 성능은 규모에 따라 일정하지 않으므로 고객 애플리케이션이 확장됨에 따라 성능 문제가 발생합니다.

데이터를 정규화하면 디스크에 저장되는 데이터의 양이 줄어들지만 성능에 영향을 미치는 가장 제한적인 리소스는 CPU 시간과 네트워크 지연 시간인 경우가 많습니다.

DynamoDB는 항목에 대한 단일 요청으로 애플리케이션 쿼리에 완전히 응답하도록 `JOINs`을 제거(및 데이터 비정규화를 촉진)하고 데이터베이스 아키텍처를 최적화하고 두 제약 조건을 모두 최소화하도록 구축되었습니다. 이러한 특성 덕분에 DynamoDB는 어떤 규모에서든 한 자릿수 밀리초 수준의 성능을 제공할 수 있습니다. DynamoDB 작업의 런타임 복잡성은 데이터 크기와 관계없이 일반적인 액세스 패턴에서 일정하기 때문입니다.

# DynamoDB 트랜잭션이 쓰기 프로세스의 오버헤드를 제거하는 방법

RDBMS를 느리게 할 수 있는 또 다른 요인은 트랜잭션을 사용하여 정규화된 스키마에 쓰는 것입니다. 예시에서 볼 수 있는 것처럼 대부분의 온라인 트랜잭션 처리(OLTP) 애플리케이션이 사용하는 관계형 데이터 구조는 구분한 후 RDBMS에 저장할 때 여러 논리적 테이블로 배포해야 합니다.

즉 ACID를 준수하는 트랜잭션 프레임워크로 애플리케이션이 쓰기 처리 중에 객체 읽기를 시도할 때 발생할 수 있는 교착 상태와 데이터 무결성 문제를 방지해야 합니다. 이러한 트랜잭션 프레임워크는 관계형 스키마와 결합될 때 쓰기 프로세스에 상당한 오버헤드를 추가할 수 있습니다.

DynamoDB에서 트랜잭션을 구현하면 RDBMS에서 발생하는 일반적인 규모 조정 문제를 방지할 수 있습니다. 트랜잭션이 단일 API 호출로 실행되고 해당 단일 트랜잭션에서 액세스할 수 있는 항목 수가 제한되기 때문입니다. 장기 실행 트랜잭션은 트랜잭션이 닫히지 않기 때문에 장기간 또는 영구적으로 데이터 잠금을 유지하여 운영상의 문제를 일으킬 수 있습니다.

DynamoDB에서 이러한 문제를 방지하기 위해 트랜잭션은 두 개의 고유한 API 작업인 `TransactWriteItems` 및 `TransactGetItems`를 사용하여 구현되었습니다. 이러한 API 작업에는 RDBMS에서 흔히 볼 수 있는 시작 및 종료 시맨틱이 없습니다. 또한 DynamoDB는 트랜잭션 내에 100개의 항목 액세스 제한을 두어 장기 실행 트랜잭션을 유사하게 방지합니다. DynamoDB 트랜잭션에 대해 자세히 알아보려면 트랜잭션 작업을 참조하세요.

이런 이유로 비즈니스에서 트래픽이 많은 쿼리에 지연 시간이 낮은 응답이 요구되는 경우, NoSQL 시스템을 활용하는 것이 기술적 및 경제적으로 합리적입니다. Amazon DynamoDB는 이를 방지해 관계형 시스템의 확장성을 제한하는 문제를 해결하는 데 도움을 줍니다.

RDBMS의 성능은 일반적으로 다음과 같은 이유로 제대로 확장되지 않습니다.

- 고가의 조인을 사용해 필요한 쿼리 결과 보기를 재수집해야 합니다.
- 데이터를 정규화해서, 여러 테이블에 저장을 하는 데, 디스크 쓰기 작업에 여러 쿼리가 요구됩니다.
- ACID 준수 트랜잭션 시스템의 경우 일반적으로 성능과 관련된 '비용'이 발생합니다.

DynamoDB는 다음 이유 때문에 효과적으로 조정이 됩니다.

- 스키마 유연성 덕분에 DynamoDB가 복잡한 계층적 데이터를 단일 항목으로 저장할 수 있습니다.
- 복합 키 설계 덕분에 관련 항목을 동일한 테이블에 함께 가까이 저장할 수 있습니다.
- 트랜잭션은 한 번의 작업으로 수행되며 장시간 실행되는 작업을 피하기 위해 액세스할 수 있는 항목 수를 100개로 제한합니다.

데이터 스토어에 대한 쿼리가 훨씬 더 간단합니다(아래 형식인 경우가 많음).

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

DynamoDB는 이전 예의 RDBMS보다 요청 데이터 반환이 효과적입니다.

## DynamoDB의 관계형 데이터를 모델링하는 첫 번째 단계

> **Important**
>
> NoSQL 설계에는 RDBMS 설계와 다른 사고 방식이 요구됩니다. RDBMS의 경우, 액세스 패턴을 생각하지 않고 정규화된 데이터 모델을 생성할 수 있습니다. 그런 후 나중에 새로운 질문과 쿼리에 대한 요구 사항이 생길 때 이를 확장할 수 있습니다. 대조적으로 Amazon DynamoDB의 경우 대답해야 할 질문을 모르기 전까지는 스키마 설계를 시작할 수 없습니다. 사전에 비즈니스 문제와 애플리케이션 사용 사례를 이해해야 합니다.

효율적으로 확장되는 DynamoDB 테이블 설계를 시작하려면, 몇 단계를 거쳐야 하는 데 먼저 지원해야 하는 OSS/BSS(운영 지원 시스템 및 비즈니스 지원 시스템)에서 요구되는 액세스 패턴을 파악해야 합니다.

- 새 애플리케이션의 경우, 활동과 목표에 대한 사용자 사례를 검토합니다. 파악한 다양한 사용 사례를 문서화하고, 여기에 필요한 액세스 패턴을 분석합니다.
- 기존 애플리케이션의 경우, 쿼리 로그를 분석해 현재 시스템을 사용하고 있는 사람의 수와 핵심 액세스 패턴을 파악해야 합니다.

이런 프로세스를 끝내면 다음과 같은 형태를 가진 목록이 준비되어야 합니다.

| | Most Common/Import Access Patterns in Our Organization |
|---|---|
| 1 | Look up employee details by employee ID |
| 2 | Query employee details by employee name |
| 3 | Find an employee's phone number(s) |
| 4 | Find a customer's phone number(s) |
| 5 | Get orders for a given customer within a given date range |
| 6 | Show all open orders within a given date range across all customers |
| 7 | See all employees hired recently |
| 8 | Find all employees working in a given warehouse |
| 9 | Get all items on order for a given product |
| 10 | Get current inventories for a given product at all warehouses |
| 11 | Get customers by account representative |
| 12 | Get orders by account representative and date |
| 13 | Get all employees with a given job title |
| 14 | Get inventory by product and warehouse |
| 15 | Get total product inventory |
| 16 | Get account representatives ranked by order total and sales period |

실제 사용하는 애플리케이션의 경우, 목록이 훨씬 더 길 것입니다. 그러나 이는 프로덕션 환경에서 찾을 수 있는 다양하고 복잡한 쿼리 패턴을 보여줍니다.

DynamoDB 스키마 설계에 대한 일반적인 접근 방식은 애플리케이션 계층의 엔터티를 파악하고, 비정규화와 복합 키 집계로 쿼리의 복잡성을 줄이는 것입니다.

DynamoDB에서는 복합 정렬 키, 오버로드된 글로벌 보조 인덱스, 분할된 테이블/인덱스 및 기타 설계 패턴을 사용하는 것을 의미합니다. 이런 요소들을 사용해 데이터를 구조화, 애플리케이션이 테이블이나 인덱스에 대한 한 번의 쿼리로 특정 액세스 패턴에 필요한 것을 검색하도록 만들 수 있습니다. 관계형 모델링에서 소개하고 있는 정규화된 스키마 모델링에 사용할 수 있는 기본 패턴은 인접 목록 패턴입니다. 기타 글로벌 보조 인덱스 쓰기 샤딩, 글로벌 보조 인덱스 오버로딩, 복합 키, 구체화된 집계 등의 패턴을 이런 설계에 사용합니다.

> **Important**
>
> 대체로 DynamoDB 애플리케이션에서는 가능한 적은 수의 테이블을 유지해야 합니다. 단 볼륨이 많은 시계열 데이터가 관여된 경우나 액세스 패턴이 아주 다른 데이터세트는 해당되지 않습니다. 통상 반전된 인덱스의 단일 테이블로 간단한 쿼리를 활성화시켜 사용자의 애플리케이션에 필요한 복잡한 계층적 데이터 구조를 생성 및 검색할 수 있습니다.

DynamoDB용 NoSQL Workbench를 사용하여 파티션 키 설계를 시각화하려면 NoSQL Workbench로 데이터 모델 빌드 섹션을 참조하세요.

# DynamoDB의 관계형 데이터 모델링에 대한 예

이 예는 Amazon DynamoDB에서 관계형 데이터를 모델링하는 방법을 설명합니다. DynamoDB 테이블 설계는 관계형 모델링에 표시된 관계형 주문 입력 스키마에 해당합니다. 이는 DynamoDB에서 관계형 데이터 구조를 표시하는 범용적인 방식인 인접 목록 설계 패턴을 따릅니다.

이 설계 패턴의 경우, 사용자가 관계형 스키마의 다양한 테이블과 연결되는 항목 유형 세트를 정의해야 합니다. 그런 후 복합(파티션 및 정렬) 기본 키를 사용하여 테이블에 개체 항목을 추가합니다. 이런 개체 항목의 파티션 키는 항목을 고유하게 식별하는 속성이며, 대체로 PK인 모든 항목을 가리킵니다. 정렬 키 속성에는 반전 인덱스나 글로벌 보조 인덱스에 사용할 수 있는 속성 값이 포함되어 있습니다. 대체로 SK입니다.

관계형 주문 입력 스키마를 지원하는 다음 개체를 정의합니다.

1. HR-Employee - PK: EmployeeID, SK: Employee Name
2. HR-Region - PK: RegionID, SK: Region Name
3. HR-Country - PK: CountryId, SK: Country Name
4. HR-Location - PK: LocationID, SK: Country Name
5. HR-Job - PK: JobID, SK: Job Title
6. HR-Department - PK: DepartmentID, SK: DepartmentName
7. OE-Customer - PK: CustomerID, SK: AccountRepID
8. OE-Order - PK OrderID, SK: CustomerID
9. OE-Product - PK: ProductID, SK: Product Name
10. OE-Warehouse - PK: WarehouseID, SK: Region Name

테이블에 이런 개체 항목을 추가하고, 개체 항목 파티션에 엣지 항목을 추가해 항목 간 관계를 정의할 수 있습니다. 다음은 이 단계에 대해 설명하고 있는 테이블입니다.

이 예에서 테이블의 `Employee`, `Order` 및 `Product Entity` 파티션에는 테이블의 다른 개체 항목에 대한 포인터가 포함되어 있는 추가 엣지 항목이 있습니다. 이제 앞서 정의한 모든 액세스 패턴을 지원하는 몇 개의 글로벌 보조 인덱스(GSI)를 정의합니다. 개체 항목들의 기본 키나 정렬 키 속성에 사용되는 값의 유형이 다릅니다. 기본 키와 정렬 키 속성을 테이블에 삽입할 수 있도록 준비하는 것만 요구됩니다.

적절한 이름을 사용하고 있는 개체가 있는 반면 다른 개체의 ID를 정렬 키 값으로 사용하는 개체도 있습니다. 이를 통해 동일한 글로벌 보조 인덱스로 여러 유형의 쿼리를 지원할 수 있습니다. 이 기법을

GSI 오버로딩이라고 합니다. 이 방법을 사용하면 여러 항목 유형이 포함된 테이블에 대해 20개의 글로벌 보조 인덱스라는 기본 제한이 사실상 제거됩니다. GSI 1이라는 다이어그램에 설명되어 있습니다.

GSI 2는 특정 상태를 가지고 있는 표의 모든 항목을 가져오는 비교적 범용적인 애플리케이션 액세스 패턴을 지원하도록 설계되어 있습니다. 가용한 상태에 대한 항목의 분포가 고르지 않은 라지 테이블의 경우, 항목을 병렬로 쿼리할 수 있도록 하나 이상의 논리적 파티션에 분포하지 않는 경우 '핫' 키가 초래될 수 있습니다. 이런 설계 패턴을 `write sharding`라고 합니다.

GSI 2에 이 작업을 수행하기 위해 애플리케이션은 모든 Order 항목에 GSI 2 기본 키 속성을 추가합니다. 0~N 범위의 난수로 채워 넣습니다. N은 일반적으로 다음 공식을 사용해 계산할 수 있습니다(이렇게 하지 않아야 하는 이유가 있는 경우 제외).

```
ItemsPerRCU = 4KB / AvgItemSize

PartitionMaxReadRate = 3K * ItemsPerRCU

N = MaxRequiredIO / PartitionMaxReadRate
```

예를 들어, 다음을 기대한다고 가정하겠습니다.

- 시스템의 주문은 최대 200만이고, 5년 내에 300만으로 증가할 것입니다.
- 특정 시점에서 열려 있는 상태의 주문은 최대 20%입니다.
- 평균적으로 주문 레코드는 약 100바이트이고, 주문 파티션의 3개 `OrderItem` 레코드는 각각 약 50바이트이므로 평균적으로 주문 엔터티의 크기는 250바이트입니다.

이 테이블에서 N 팩터는 다음과 같이 계산합니다.

```
ItemsPerRCU = 4KB / 250B = 16

PartitionMaxReadRate = 3K * 16 = 48K

N = (0.2 * 3M) / 48K = 13
```

이 경우, GSI 2의 논리적 파티션 13개 이상에 모든 주문을 배포해 `OPEN` 상태인 모든 `Order` 항목의 읽기 작업이 물리적 스토리지 계층에서 핫 파티션을 초래하지 않도록 만들어야 합니다. 이 숫자를 덧대 데이터세트에서 이상(이례)을 허용하는 것이 좋습니다. 즉 `N = 15`을 사용하는 모델이 아마 좋을 것입니다. 앞서 언급했듯이 테이블에 삽입된 각 `Order` 및 `OrderItem` 레코드의 GSI 2 PK 속성에 0~N 사이의 임의의 값을 추가해 이렇게 할 수 있습니다.

이는 모든 OPEN 인보이스 수집을 요구하는 액세스 패턴이 비교적 적게 발생해, 버스트 용량으로 요청을 충족할 수 있다는 가정 아래 분석을 한 것입니다. State 및 Date Range Sort Key 조건을 사용하는 다음의 글로벌 보조 인덱스를 쿼리해 필요한 특정 상태의 모든 Orders의 하위집합을 생성할 수 있습니다.

이 예에서는 15개의 논리적 파티션을 대상으로 항목을 임의 배포합니다. 이 구조는 액세스 패턴에 검색할 항목의 수가 많기 때문에 작동을 합니다. 즉 15개 스레드 중 어느 하나가 낭비된 용량을 대표할 수 있는 비어있는 결과 세트를 반환할 확률이 낮습니다. 반환이 없거나, 작성된 데이터가 없는 경우에도, 쿼리는 항상 1RCU(읽기 용량 단위)나 1WCU(쓰기 용량 단위)을 사용합니다.

액세스 패턴이 희소한 결과 세트를 반환하는 이러한 글로벌 보조 인덱스에 요구하는 쿼리 속도가 빨라야 하는 경우, 임의 패턴보다는 항목을 분산시키는 해시 알고리즘을 사용하는 것이 더 좋을 수 있습니다. 이 경우, 쿼리가 실행 시간에 실행될 때 알려진 속성을 선택하고 이 속성을 항목이 삽입될 때 0~14의 키 공간에 해시합니다. 그러면 글로벌 보조 인덱스에서 효율적으로 읽기 작업을 수행할 수 있습니다.

마지막으로 앞서 정의한 액세스 패턴을 다시 살펴보겠습니다. 다음은 수용하는 애플리케이션의 새 DynamoDB 버전과 함께 사용할 수 있는 액세스 패턴과 쿼리 조건의 목록입니다.

| 일련 번호 | 액세스 패턴 | 쿼리 조건 |
|---|---|---|
| 1 | 직원 ID별로 직원 세부 정보 조회 | 테이블의 프라이머리 키, ID='HR-EMPLOYEE' |
| 2 | 직원 이름별 직원 세부 정보 쿼리 | GSI-1, PK = 'Employee Name' 사용 |
| 3 | 직원의 현재 작업 세부 정보만 가져오기 | 테이블의 프라이머리 키, PK=HR-EMPLOYEE-1, SK는 'JH'로 시작 |
| 4 | 특정 고객의 특정 날짜 범위 주문 가져오기 | 각 StatusCode에 대해 GSI-1, PK=CUSTOMER1, SK='STATUS-DATE' 사용 |
| 5 | 모든 고객에 걸쳐 특정 날짜 범위에 OPEN 상태인 모든 주문 표시 | OPEN-Date1과 OPEN-Date2 사이의 범위 [0..N], SK에 대해 병렬로 GSI-2, PK=query 사용 |

| 일련 번호 | 액세스 패턴 | 쿼리 조건 |
| --- | --- | --- |
| 6 | 최근 채용된 모든 직원 | GSI-1, PK='HR-CONFIDENTIAL', SK > date1 사용 |
| 7 | 특정 창고의 모든 직원 찾기 | GSI-1, PK=WAREHOUSE1 사용 |
| 8 | 창고 위치 재고를 포함하여 제품에 대한 모든 주문 품목 가져오기 | GSI-1, PK=PRODUCT1 사용 |
| 9 | 계정 담당자별로 고객 가져오기 | GSI-1, PK=ACCOUNT-REP 사용 |
| 10 | 계정 담당자 및 날짜별로 주문 가져오기 | 각 StatusCode에 대해 GSI-1, PK=ACCOUNT-REP, SK='STATUS-DATE' 사용 |
| 11 | 특정 직책의 모든 직원 가져오기 | GSI-1, PK=JOBTITLE 사용 |
| 12 | 제품 및 창고별로 재고 가져오기 | 테이블의 프라이머리 키, PK=OE-PRODUCT1,SK=PRODUCT1 |
| 13 | 전체 제품 재고 가져오기 | 테이블의 프라이머리 키, PK=OE-PRODUCT1,SK=PRODUCT1 |
| 14 | 주문 총계 및 판매 기간별로 순위가 지정된 계정 담당자 가져오기 | GSI-1, PK=YYYY-Q1, scanIndexForward=False 사용 |

# 관계형 데이터베이스에서 DynamoDB로 마이그레이션

관계형 데이터베이스를 DynamoDB로 마이그레이션하려면 성공적인 결과를 보장하기 위해 신중한 계획이 필요합니다. 이 안내서는 이 프로세스의 작동 방식, 사용 가능한 도구, 잠재적 마이그레이션 전략을 평가하고 요구 사항에 맞는 전략을 선택하는 방법을 이해하는 데 도움이 됩니다.

주제


- DynamoDB로 마이그레이션하는 이유
- 관계형 데이터베이스를 DynamoDB로 마이그레이션하는 경우 고려 사항
- DynamoDB로의 마이그레이션 작동 방식 이해
- DynamoDB로 마이그레이션하는 데 도움이 되는 도구
- DynamoDB로 마이그레이션하기 위한 적절한 전략 선택
- DynamoDB로 오프라인 마이그레이션 수행
- DynamoDB로 하이브리드 마이그레이션 수행
- 각 테이블을 일대일로 마이그레이션하여 DynamoDB로 온라인 마이그레이션 수행
- 사용자 지정 스테이징 테이블을 사용하여 DynamoDB로 온라인 마이그레이션 수행


## DynamoDB로 마이그레이션하는 이유

Amazon DynamoDB로 마이그레이션하면 기업과 조직은 다양하고 매력적인 이점을 누릴 수 있습니다. 다음은 DynamoDB가 데이터베이스 마이그레이션에 적합한 이유라고 할 수 있는 주요 이점입니다.

- 확장성: DynamoDB는 대규모 워크로드를 처리하고 증가하는 데이터 볼륨 및 트래픽을 수용하기 위해 원활하게 확장할 수 있도록 설계되었습니다. DynamoDB를 사용하면 수요에 따라 데이터베이스를 쉽게 확장하거나 축소할 수 있으므로 애플리케이션에서 성능 저하 없이 갑작스러운 트래픽 급증을 처리할 수 있습니다.
- 성능: DynamoDB는 지연 시간이 짧은 데이터 액세스를 제공하므로 애플리케이션이 뛰어난 속도로 데이터를 검색하고 처리할 수 있습니다. 분산 아키텍처는 읽기 및 쓰기 작업이 여러 노드에 분산되도록 하여 요청 속도가 높아도 일관되게 10밀리초 미만의 응답 시간을 제공합니다.
- 완전관리형: DynamoDB는 AWS에서 제공하는 완전관리형 서비스입니다. 즉, 프로비저닝, 구성, 패치 적용, 백업, 규모 조정 등 데이터베이스 관리의 운영 측면을 AWS가 처리합니다. 따라서 개발자는 데이터베이스 관리 작업보다는 애플리케이션 개발에 더 집중할 수 있습니다.
- 서버리스 아키텍처: DynamoDB는 DynamoDB 온디맨드라고 하는 서버리스 모델을 지원합니다. 이 모델에서는 사전 용량 프로비저닝이 필요 없이 애플리케이션이 수행한 실제 읽기 및 쓰기 요청에 대

해서만 비용을 지불합니다. 이 요청당 지불 모델은 용량을 프로비저닝하고 모니터링할 필요 없이 사용한 리소스에 대해서만 비용을 지불하므로 비용 효율성이 높고 운영 오버헤드를 최소화합니다.

- NoSQL 유연성: 기존 관계형 데이터베이스와 달리 DynamoDB는 NoSQL 데이터 모델을 따르므로 스키마 설계에 유연성을 제공합니다. DynamoDB를 사용하면 정형, 반정형 및 비정형 데이터를 저장할 수 있으므로 다양하고 변화하는 데이터 유형을 처리하는 데 적합합니다. 이러한 유연성 덕분에 개발 주기를 단축하고 변화하는 비즈니스 요구 사항에 쉽게 적응할 수 있습니다.
- 고가용성 및 내구성: DynamoDB는 리전 내 여러 가용 영역에 데이터를 복제하여 고가용성과 데이터 내구성을 보장합니다. 복제, 장애 조치 및 복구를 자동으로 처리하여 데이터 손실이나 서비스 중단의 위험을 최소화합니다. DynamoDB는 최대 99.999%의 가용성 SLA를 제공합니다.
- 보안 및 규정 준수: DynamoDB는 AWS Identity and Access Management와 통합되어 세분화된 액세스 제어를 제공합니다. 저장 데이터 암호화 및 전송 중 암호화 기능을 제공하여 데이터의 보안을 보장합니다. 또한 DynamoDB는 HIPAA, PCI DSS, GDPR을 비롯한 다양한 규정 준수 표준을 준수하므로 규제 요구 사항을 충족할 수 있습니다.
- AWS 에코시스템과의 통합: DynamoDB는 AWS 에코시스템의 일부로서 AWS Lambda, AWS CloudFormation, AWS AppSync 등의 다른 AWS 서비스와 원활하게 통합됩니다. 이러한 통합을 통해 서버리스 아키텍처를 구축하고, 코드형 인프라를 활용하고, 실시간 데이터 기반 애플리케이션을 만들 수 있습니다.

# 관계형 데이터베이스를 DynamoDB로 마이그레이션하는 경우 고려 사항

관계형 데이터베이스 시스템과 NoSQL 데이터베이스는 각기 다른 장단점을 갖고 있습니다. 이런 차이가 두 시스템의 데이터베이스 설계를 다르게 만듭니다.

| 작업 유형 | 관계형 데이터베이스 | NoSQL 데이터베이스 |
| --- | --- | --- |
| 데이터베이스 쿼리 | 관계형 데이터베이스에서는 데이터를 유연하게 쿼리할 수 있지만, 쿼리 비용이 상대적으로 높으며 트래픽이 많은 상황에서는 확장성이 떨어집니다 (DynamoDB의 관계형 데이터를 모델링하는 첫 번째 단계 참조). 관계형 데이터베이스 애플리케이션은 저장 프로시저, | DynamoDB와 같은 NoSQL 데이터베이스에서는 몇 가지 방법으로 데이터를 효율적으로 쿼리할 수 있지만, 그 외에는 쿼리 비용이 높고 속도가 느립니다. DynamoDB에 대한 쓰기는 singleton입니다. 이전에 저장 프로시저에서 실행되었던 애플리케이션 비즈니스 로 |

| 작업 유형 | 관계형 데이터베이스 | NoSQL 데이터베이스 |
| --- | --- | --- |
| | SQL 하위 쿼리, 대량 업데이트 쿼리 및 집계 쿼리에서 비즈니스 로직을 구현할 수 있습니다. | 직은 Amazon EC2 또는 AWS Lambda 같은 호스트에서 실행되는 사용자 지정 코드로 DynamoDB 외부에서 실행되도록 리팩터링해야 합니다. |
| 데이터베이스 설계 | 세부적인 구현이나 성능을 걱정하지 않고 유연성을 목적으로 설계합니다. 일반적으로 쿼리 최적화가 스키마 설계에 영향을 미치지 않지만, 정규화가 중요합니다. | 가장 중요하고 범용적인 쿼리를 가능한 빠르고 저렴하게 수행할 수 있도록 스키마를 설계합니다. 사용자의 데이터 구조는 사용자 비즈니스 사용 사례의 특정 요구 사항에 적합하도록 만듭니다. |

NoSQL 데이터베이스를 설계할 때는 관계형 데이터베이스 관리 시스템(RDBMS)과는 다른 사고 방식이 필요합니다. RDBMS의 경우, 액세스 패턴을 생각하지 않고 정규화된 데이터 모델을 생성할 수 있습니다. 그런 후 나중에 새로운 질문과 쿼리에 대한 요구 사항이 생길 때 이를 확장할 수 있습니다. 각 데이터 유형을 고유의 테이블에 구성할 수 있습니다.

NoSQL 설계와는 달리, 대답해야 할 질문을 알았을 때 DynamoDB에 대한 스키마를 설계해야 합니다. 비즈니스 문제와 애플리케이션 읽기 및 쓰기 패턴을 이해하는 것이 필수입니다. 또한 DynamoDB 애플리케이션에서는 가능한 적은 수의 테이블을 유지하는 것을 목표로 해야 합니다. 테이블 수가 적을수록 확장성이 향상되고 권한 관리가 줄어들며 DynamoDB 애플리케이션의 오버헤드가 감소합니다. 백업 비용도 전반적으로 낮게 유지할 수 있습니다.

DynamoDB의 관계형 데이터를 모델링하고 프런트엔드 애플리케이션의 새 버전을 구축하는 작업은 별도의 주제입니다. 이 안내서에서는 DynamoDB를 사용하도록 새 버전의 애플리케이션을 구축했다고 가정하지만, 전환 중에 기록 데이터를 마이그레이션하고 동기화하는 최선의 방법을 결정해야 합니다.

크기 조정 고려 사항

DynamoDB 테이블에 저장하는 각 항목(행)의 최대 크기는 400KB입니다. 자세한 내용은 the section called “할당량” 단원을 참조하십시오. 항목 크기는 항목에 있는 모든 속성 이름 및 속성 값의 총 크기에 따라 결정됩니다. 자세한 내용은 the section called “항목 크기 및 형식” 단원을 참조하십시오.

애플리케이션이 DynamoDB 크기 제한이 허용하는 것보다 많은 데이터를 항목에 저장해야 하는 경우, 항목을 항목 모음으로 나누거나, 항목 데이터를 압축하거나, Amazon Simple Storage Service(S3)

에 항목을 객체로 저장하는 동시에 DynamoDB 항목에 Amazon S3 객체 식별자를 저장합니다. the section called “큰 항목” 섹션을 참조하세요. 항목 업데이트 비용은 항목의 전체 크기를 기준으로 합니다. 기존 항목을 자주 업데이트해야 하는 워크로드의 경우 1~2KB의 작은 항목이 있으면 큰 항목보다 업데이트 비용이 적게 듭니다. 항목 모음에 대한 자세한 내용은 the section called “항목 컬렉션 작업” 섹션을 참조하세요.

파티션 및 정렬 키 속성, 기타 테이블 설정, 항목 크기 및 구조, 보조 인덱스 생성 여부를 선택할 때는 DynamoDB 모델링 설명서와 the section called “비용 최적화”에 대한 안내서를 검토하세요. DynamoDB 솔루션이 비용 효율적이고 DynamoDB의 기능 및 제한 사항을 벗어나지 않는지 마이그레이션 계획을 테스트해야 합니다.

## DynamoDB로의 마이그레이션 작동 방식 이해

사용 가능한 마이그레이션 도구를 검토하기 전에 DynamoDB에서 쓰기를 처리하는 방법을 고려해 보세요.

기본적이고 가장 일반적인 쓰기 작업은 단일 `PutItem` API 작업입니다. 루프에서 `PutItem` 작업을 수행하여 데이터세트를 처리할 수 있습니다. DynamoDB는 사실상 무제한의 동시 연결을 지원하므로 MapReduce 또는 Spark와 같은 대규모 다중 스레드 로드 루틴을 구성하고 실행할 수 있다고 가정하면 쓰기 속도는 대상 테이블의 용량에 의해서만 제한됩니다. 대상 테이블의 용량도 일반적으로 무제한입니다.

DynamoDB로 데이터를 로드할 때는 로더의 쓰기 속도를 이해하는 것이 중요합니다. 로드하는 항목(행)의 크기가 1KB 이하인 경우 이 속도는 단순히 초당 항목 수입니다. 그러면 대상 테이블에 이 속도를 처리할 수 있는 충분한 쓰기 용량 단위(WCU)를 프로비저닝할 수 있습니다. 로더가 어느 시점에든 프로비저닝된 용량을 초과할 경우 추가 요청이 제한되거나 아예 거부될 수 있습니다. DynamoDB 콘솔 모니터링 탭에 있는 CloudWatch 차트에서 제한을 확인할 수 있습니다.

두 번째로 수행할 수 있는 작업은 `BatchWriteItem`이라는 관련 API를 사용하는 것입니다. `BatchWriteItem`은 최대 25개의 쓰기 요청을 하나의 API 직접 호출로 결합할 수 있습니다. 이러한 요청은 서비스에서 수신되며 테이블에 대한 개별 `PutItem` 요청으로 처리됩니다. 현재는 `BatchWriteItem`을 선택하면 `PutItem`을 사용하여 singleton을 직접적으로 호출할 때 AWS SDK에 포함된 자동 재시도 기능을 이용할 수 없습니다. 따라서 오류(예: 제한 예외)가 있는 경우 `BatchWriteItem`에 대한 응답 직접 호출에서 실패한 쓰기 목록을 찾아야 합니다. CloudWatch 제한 차트에서 제한 경고가 감지된 경우 이를 처리하는 방법에 대한 자세한 내용은 the section called “제한(Throttling) 문제” 섹션을 참조하세요.

세 번째 유형의 데이터 가져오기는 S3에서 DynamoDB로 가져오기 기능을 사용하는 것입니다. 이 기능을 사용하면 Amazon S3에 대규모 데이터세트를 스테이징하고 DynamoDB에 데이터를 새 테이블로 자동으로 가져오도록 요청할 수 있습니다. 가져오기는 즉시 이루어지지 않으며, 데이터세트의 크기에 비례하여 시간이 걸립니다. 하지만 ETL 플랫폼을 사용하거나 사용자 지정 DynamoDB 코드를 작성할 필요가 없으므로 편리합니다. DynamoDB는 가져오기를 통해 생성된 새 테이블에 데이터를 로드합니다. 현재는 기존 테이블에 데이터를 로드할 수 없습니다. DynamoDB는 변환 없이 데이터를 있는 그대로 가져옵니다. `PutItem`과 마찬가지로 업스트림 프로세스가 필요하며 선택한 형식의 데이터를 Amazon S3 버킷에 씁니다.

## DynamoDB로 마이그레이션하는 데 도움이 되는 도구

데이터를 DynamoDB로 마이그레이션하는 데 사용할 수 있는 몇 가지 일반적인 마이그레이션 및 ETL 도구가 있습니다.

Amazon은 AWS Database Migration Service(DMS), AWS Glue, Amazon EMR, Amazon Managed Streaming for Apache Kafka 등 마이그레이션에 사용할 수 있는 다양한 데이터 도구를 제공합니다. 이러한 도구를 모두 사용하여 가동 중지 시간 마이그레이션을 수행할 수 있으며, 관계형 데이터베이스 변경 데이터 캡처(CDC) 기능을 활용하여 온라인 마이그레이션도 지원할 수 있습니다. 도구를 선택할 때는 각 도구의 기능, 성능 및 비용과 함께 조직이 각 도구에 대해 보유하고 있는 기술 역량 및 경험을 고려하는 것이 도움이 됩니다.

많은 고객이 마이그레이션 프로세스를 위한 사용자 지정 데이터 변환을 구축하기 위해 자체 마이그레이션 스크립트와 작업을 작성합니다. 쓰기 트래픽이 많거나 정기적인 대량 로드 작업이 있는 대용량 DynamoDB 테이블을 운영하려는 경우 쓰기 트래픽이 많을 때 DynamoDB의 동작에 더 익숙해지도록 마이그레이션 코드를 직접 작성하는 것이 좋습니다. 마이그레이션을 연습할 때 제한 처리 및 효율적인 테이블 프로비저닝과 같은 시나리오를 프로젝트 초기에 경험할 수 있습니다.

## DynamoDB로 마이그레이션하기 위한 적절한 전략 선택

대규모 관계형 데이터베이스 애플리케이션은 테이블이 100개 이상일 수 있으며 다양한 애플리케이션 함수를 지원할 수 있습니다. 대규모 마이그레이션을 진행할 때는 애플리케이션을 더 작은 구성 요소나 마이크로서비스로 나누고 한 번에 작은 테이블 집합을 마이그레이션하는 것을 고려해 보세요. 그런 다음 추가 구성 요소를 DynamoDB로 여러 차례에 걸쳐 마이그레이션할 수 있습니다.

마이그레이션 전략을 선택할 때 다양한 요인에 따라 솔루션을 선택하게 될 수 있습니다. 요구 사항과 사용 가능한 리소스를 고려하여 사용 가능한 옵션을 단순화하기 위해 의사 결정 트리로 이러한 옵션을 시각화할 수 있습니다. 개념은 여기에 간략하게 설명되어 있습니다(이 안내서의 뒷부분에서 더 자세히 다룰 예정).

- 오프라인 마이그레이션: 마이그레이션 중에 애플리케이션이 가동 중지 시간을 어느 정도 감수할 수 있다면 오프라인 마이그레이션은 마이그레이션 프로세스를 단순화합니다.
- 하이브리드 마이그레이션: 이 접근 방식을 사용하면 마이그레이션 중에 부분적인 가동 시간을 허용할 수 있습니다. 예를 들어 읽기는 허용하지만 쓰기는 허용하지 않거나, 읽기 및 삽입은 허용하지만 업데이트 및 삭제는 허용하지 않습니다.
- 온라인 마이그레이션: 마이그레이션 중에 가동 중지 시간이 전혀 없어야 하는 애플리케이션은 마이그레이션하기가 상대적으로 쉽지 않으며 상당한 계획과 맞춤형 개발이 필요할 수 있습니다. 결정해야 하는 한 가지 중요한 사항은 사용자 지정 마이그레이션 프로세스를 구축하는 데 드는 비용과 전환 중에 가동 중지 시간이 발생하는 데 드는 비즈니스 비용을 추정하여 평가하는 것입니다.

| If | And | Then |
|---|---|---|
| 데이터 마이그레이션을 수행하기 위해 유지 관리 기간 동안 애플리케이션을 잠시 중단해도 괜찮습니다. 이 방식은 오프라인 마이그레이션입니다. | | AWS DMS를 사용하고 전체 로드 작업을 사용하여 오프라인 마이그레이션을 수행합니다. 원하는 경우 SQL `VIEW`를 사용하여 소스 데이터를 미리 구성합니다. |
| 마이그레이션 중에 애플리케이션을 읽기 전용 모드로 실행해도 됩니다. 이 방식은 하이브리드 마이그레이션입니다. | | 애플리케이션 또는 소스 데이터베이스 내에서 쓰기를 비활성화합니다. AWS DMS를 사용하고 전체 로드 작업을 사용하여 오프라인 마이그레이션을 수행합니다. |
| 마이그레이션 중에 읽기 및 새 레코드 삽입을 허용한 상태로 애플리케이션을 실행해도 괜찮습니다. 단, 업데이트나 삭제는 허용하지 않습니다. 이 방식은 하이브리드 마이그레이션입니다. | 애플리케이션 개발 기술 역량을 보유하고 있으며 모든 새 레코드에 대해 DynamoDB를 포함하여 이중 쓰기를 수행하도록 기존 관계형 앱을 업데이트할 수 있습니다. | AWS DMS를 사용하고 전체 로드 작업을 사용하여 오프라인 마이그레이션을 수행합니다. 동시에, 읽기를 허용하고 이중 쓰기를 수행하는 기존 앱 버전을 배포합니다. |
| 가동 중지 시간을 최소화하면서 마이그레이션해야 합니다. | • 주요 스키마 변경 없이 소스 테이블을 DynamoDB로 일대 | AWS DMS를 사용하여 온라인 데이터 마이그레이션을 수행합 |

| If | And | Then |
| --- | --- | --- |
| 이 방식은 온라인 마이그레이션입니다. | 일로 마이그레이션하고 있습니다. | 니다. 대량 로드 작업을 실행한 다음 CDC 동기화 작업을 실행합니다. |
| 가동 중지 시간을 최소화하면서 마이그레이션해야 합니다. 이 방식은 온라인 마이그레이션입니다. | • 누적 스키마 또는 단일 테이블 철학에 따라 소스 테이블을 더 적은 수의 DynamoDB 테이블로 결합하고 있습니다.<br>  • 백엔드 데이터베이스 개발 기술 역량이 있으며 SQL 호스트에 예비 용량이 있습니다. | SQL 데이터베이스 내에 NoSQL 지원 테이블을 생성합니다. JOIN, UNION, VIEW, 트리거, 저장 프로시저를 사용하여 테이블을 채우고 동기화합니다. |
| 가동 중지 시간을 최소화하면서 마이그레이션해야 합니다. 이 방식은 온라인 마이그레이션입니다. | • 단일 테이블 철학에 따라 소스 테이블을 더 적은 수의 DynamoDB 테이블로 결합하고 있습니다. 예:<br>  • 백엔드 데이터베이스 개발 기술 역량이 없으며 SQL 호스트에 예비 용량이 없습니다. | 하이브리드 또는 오프라인 마이그레이션 접근 방식을 고려해 보세요. |
| 가동 중지 시간을 최소화하면서 마이그레이션해야 합니다. 이 방식은 온라인 마이그레이션입니다. | 이전 트랜잭션 데이터를 마이그레이션하지 않아도 되거나 마이그레이션하는 대신 Amazon S3에 아카이브할 수 있습니다. 몇 개의 작은 정적 테이블만 마이그레이션하면 됩니다. | 스크립트를 작성하거나 ETL 도구를 사용하여 테이블을 마이그레이션합니다. 원하는 경우 SQL `VIEW`를 사용하여 소스 데이터를 미리 구성합니다. |

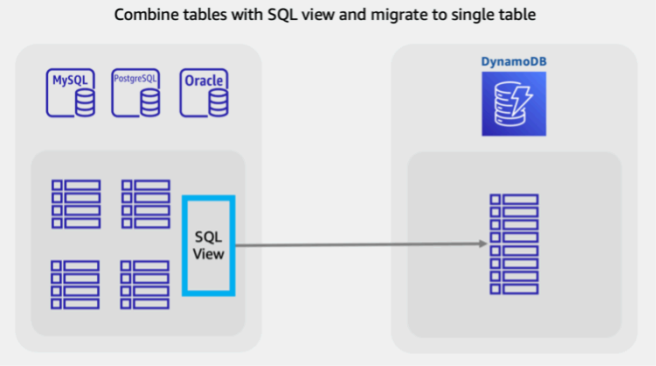
# DynamoDB로 오프라인 마이그레이션 수행

오프라인 마이그레이션은 마이그레이션을 수행할 때 가동 중지 시간을 허용할 수 있는 경우에 적합합니다. 관계형 데이터베이스는 일반적으로 유지 관리 및 패치 적용으로 인해 매달 최소 어느 정도의 가

동 중지 시간이 발생하며 하드웨어 업그레이드 또는 메이저 릴리스 업그레이드의 경우 가동 중지 시간이 더 길어질 수 있습니다.

Amazon S3는 마이그레이션 중에 스테이징 영역으로 사용할 수 있습니다. S3에서 DynamoDB로 가져오기 기능을 사용하여 쉼표로 구분된 값(CSV) 또는 DynamoDB JSON 형식으로 저장된 데이터를 새 DynamoDB 테이블로 자동으로 가져올 수 있습니다.

고유한 NoSQL 액세스 패턴을 활용하기 위해 테이블을 결합할 수 있습니다(예: 4개의 레거시 테이블을 단일 DynamoDB 테이블로 변환). 단일 키-값 문서 요청 또는 사전 그룹화된 항목 모음에 대한 쿼리는 일반적으로 다중 테이블 조인을 수행하는 SQL 데이터베이스보다 지연 시간이 더 짧습니다. 하지만 이로 인해 마이그레이션 작업이 더 어려워집니다. SQL 뷰는 소스 데이터베이스 내에서 작업을 수행하여 테이블 4개 모두를 하나의 집합으로 나타내는 단일 데이터세트를 준비할 수 있습니다.




이 뷰는 테이블을 비정규화된 형태로 `JOIN`할 수도 있고, 엔터티를 정규화한 상태로 유지하고 SQL `UNION`을 사용하여 테이블을 스태킹할 수도 있습니다. 관계형 데이터 재구성과 관련된 주요 결정 사항은 이 동영상에서 다룹니다. 오프라인 마이그레이션의 경우 뷰를 사용하여 테이블을 결합하는 것은 DynamoDB 단일 테이블 스키마의 데이터를 구성하는 좋은 방법입니다.

계획

Amazon S3를 사용하여 오프라인 마이그레이션 수행

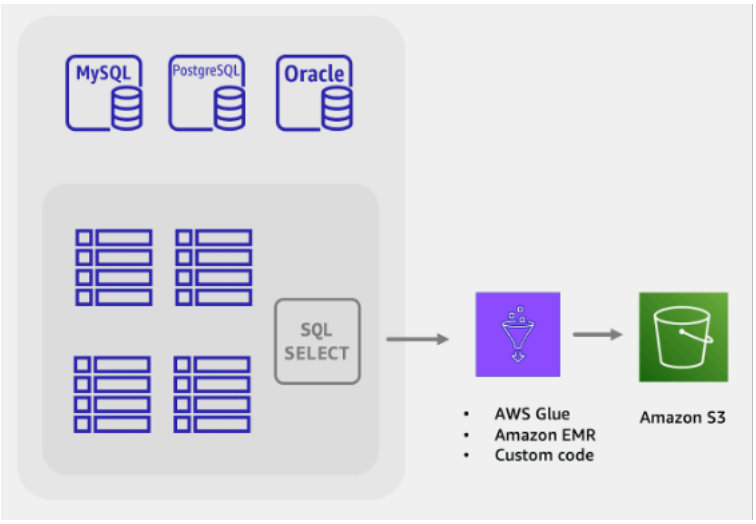
도구

- SQL 데이터를 추출 및 변환하여 다음과 같은 S3 버킷에 저장하는 ETL 작업:
  - 기록 데이터를 대량으로 로드하고 변경 데이터 캡처(CDC) 레코드를 처리하여 소스 테이블과 대상 테이블을 동기화할 수 있는 서비스인 AWS Database Migration Service.
  - AWS Glue
  - Amazon EMR
  - 자체 사용자 지정 코드

- S3에서 DynamoDB로 가져오기 기능

오프라인 마이그레이션 단계:

1. SQL 데이터베이스를 쿼리하고, 테이블 데이터를 DynamoDB JSON 또는 CSV 형식으로 변환하고, S3 버킷에 저장할 수 있는 ETL 작업을 빌드합니다.




2. S3에서 DynamoDB로 가져오기 기능이 간접 호출되어 새 테이블을 생성하고 S3 버킷에서 데이터를 자동으로 로드합니다.

완전 오프라인 마이그레이션은 단순하고 간단하지만 애플리케이션 소유자와 사용자에게는 인기가 없을 수 있습니다. 마이그레이션 중에 애플리케이션이 서비스를 전혀 제공하지 않는 대신 축소된 서비스를 제공할 수 있다면 사용자에게 도움이 될 것입니다.

오프라인 마이그레이션 중에 쓰기를 비활성화하는 기능을 추가하고 읽기는 정상적으로 계속되도록 할 수 있습니다. 관계형 데이터를 마이그레이션하는 동안에도 애플리케이션 사용자는 기존 데이터를 안전하게 찾아보고 쿼리할 수 있습니다. 이 내용을 찾고 있었다면 계속 읽으면서 하이브리드 마이그레이션에 대해 알아보세요.

## DynamoDB로 하이브리드 마이그레이션 수행

모든 데이터베이스 애플리케이션이 읽기 및 쓰기 작업을 수행하지만 하이브리드 또는 온라인 마이그레이션을 계획할 때는 수행 중인 쓰기 작업 유형을 고려해야 합니다. 데이터베이스 쓰기는 삽입, 업데이트 및 삭제의 세 가지로 분류할 수 있습니다. 일부 애플리케이션의 경우 삭제를 즉시 처리하지 않아도 될 수 있습니다. 예를 들어 이러한 애플리케이션에서는 월말에 일괄 정리 프로세스를 수행하는 방식

으로 삭제를 연기할 수 있습니다. 이러한 유형의 애플리케이션은 부분적인 가동 시간을 허용하면서 더 간단하게 마이그레이션할 수 있습니다.

계획

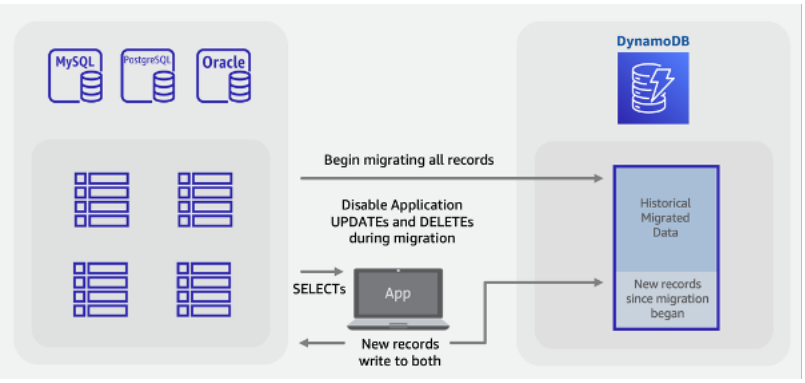
애플리케이션 이중 쓰기를 허용하면서 하이브리드 온라인/오프라인 마이그레이션 수행

도구

- SQL 데이터를 추출 및 변환하여 다음과 같은 S3 버킷에 저장하는 ETL 작업:
  - AWS DMS
  - AWS Glue
  - Amazon EMR
  - 자체 사용자 지정 코드

하이브리드 마이그레이션 단계:

1. 대상 DynamoDB 테이블을 생성합니다. 이 테이블은 과거 대량 데이터와 새로운 라이브 데이터를 모두 수신합니다.
2. SQL 데이터베이스와 DynamoDB 모두에 대해 모든 삽입을 이중 쓰기로 수행하면서 삭제 및 업데이트가 비활성화된 레거시 애플리케이션 버전을 생성합니다.
3. ETL 작업 또는 AWS DMS 태스크를 시작하여 기존 데이터를 채우는 동시에 새 애플리케이션 버전을 배포합니다.
4. 채우기 작업이 완료되면 DynamoDB는 기존 레코드와 새 레코드를 모두 보유하고 애플리케이션 전환을 수행할 준비가 됩니다.



> **Note**
>
> 채우기 작업은 SQL에서 DynamoDB로 직접 씁니다. 오프라인 마이그레이션 예제에서와 같이 S3 가져오기 기능을 사용할 수 없는데, 이 기능은 DynamoDB가 데이터를 로드할 때까지 라이브가 아닌 새 테이블을 생성하기 때문입니다.

# 각 테이블을 일대일로 마이그레이션하여 DynamoDB로 온라인 마이그레이션 수행

많은 관계형 데이터베이스에는 변경 데이터 캡처(CDC)라는 기능이 있습니다. 이 기능을 사용하면 사용자가 데이터베이스에서 특정 시점 전 또는 후에 발생한 테이블 변경 사항 목록을 요청할 수 있습니다. CDC는 내부 로그를 사용하여 이 기능을 활성화하며 테이블에 타임스탬프 열이 없어도 작동합니다.

SQL 테이블의 스키마를 NoSQL 데이터베이스로 마이그레이션할 때 데이터를 결합하여 더 적은 수의 테이블로 재구성해야 할 수 있습니다. 이렇게 하면 한 곳에서 데이터를 수집할 수 있고 여러 단계의 읽기 작업에서 관련 데이터를 수동으로 조인하지 않아도 됩니다. 하지만 단일 테이블 데이터 구성이 항상 필요한 것은 아니며 테이블을 일대일로 DynamoDB로 마이그레이션하는 경우도 있습니다. 이러한 일대일 테이블 마이그레이션은 이러한 유형의 마이그레이션을 지원하는 일반 ETL 도구를 사용하여 소스 데이터베이스 CDC 기능을 활용할 수 있으므로 덜 복잡합니다. 각 행의 데이터는 여전히 새 형식으로 변환될 수 있지만 각 테이블의 범위는 변하지 않습니다.

DynamoDB는 서버 측 조인을 지원하지 않는다는 점을 염두에 두고 SQL 테이블을 DynamoDB로 일대일로 마이그레이션하는 것을 고려해 보세요. 여러 테이블의 데이터를 결합하려면 애플리케이션에 로직을 추가해야 합니다.

계획

AWS DMS를 사용하여 각 테이블을 DynamoDB로 온라인 마이그레이션 수행

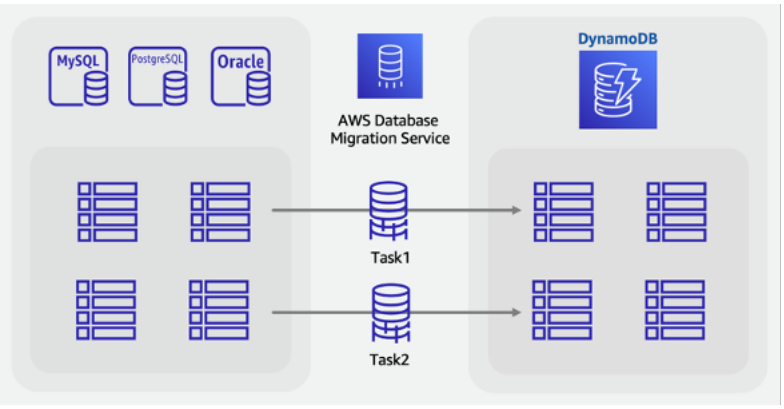
도구

- AWS DMS(DMS)

온라인 마이그레이션 단계:

1. 소스 스키마에서 마이그레이션할 테이블을 식별합니다.

2. 소스와 동일한 키 구조를 사용하여 DynamoDB에 동일한 수의 테이블을 생성합니다.
3. AWS DMS에서 복제 서버를 생성하고 소스 및 대상 엔드포인트를 구성합니다.
4. 필요한 행별 변환(예: 열을 연결하거나 날짜를 ISO-8601 문자열 형식으로 변환)을 정의합니다.
5. 전체 로드 및 변경 데이터 캡처를 위해 각 테이블에 대해 마이그레이션 작업을 생성합니다.
6. 진행 중인 복제 단계가 시작될 때까지 이러한 작업을 모니터링합니다.
7. 이 시점에서 검증 감사를 수행한 다음 사용자를 DynamoDB에 읽고 쓰는 애플리케이션으로 전환할 수 있습니다.




# 사용자 지정 스테이징 테이블을 사용하여 DynamoDB로 온라인 마이그레이션 수행

위의 오프라인 마이그레이션 시나리오와 마찬가지로 고유한 NoSQL 액세스 패턴을 활용하기 위해 테이블을 결합할 수 있습니다(예: 4개의 레거시 테이블을 단일 DynamoDB 테이블로 변환). SQL `VIEW`는 소스 데이터베이스 내에서 작업을 수행하여 테이블 4개 모두를 하나의 집합으로 나타내는 단일 데이터 세트를 준비할 수 있습니다.

하지만 변경되는 라이브 데이터가 포함된 온라인 마이그레이션의 경우 CDC 기능은 `VIEW`에서 지원되지 않으므로 CDC 기능을 활용할 수 없습니다. 테이블에 마지막으로 업데이트된 타임스탬프 열이 포함되어 있고 이 열이 `VIEW`에 통합되어 있는 경우 이를 사용하여 동기화를 통해 대량 로드를 처리하는 사용자 지정 ETL 작업을 만들 수 있습니다.

이 문제에 대한 새로운 접근 방식은 뷰, 저장 프로시저, 트리거와 같은 표준 SQL 기능을 사용하여 최종적으로 원하는 DynamoDB NoSQL 형식의 새 SQL 테이블을 생성하는 것입니다.

데이터베이스 서버에 여유 용량이 있는 경우 마이그레이션이 시작되기 전에 이 단일 스테이징 테이블을 생성할 수 있습니다. 이를 위해서는 기존 테이블에서 읽고, 필요에 따라 데이터를 변환하고, 새 스테

이징 테이블에 쓰는 저장 프로시저를 작성하면 됩니다. 일련의 트리거를 추가하여 테이블의 변경 사항을 스테이징 테이블에 실시간으로 복제할 수 있습니다. 회사 정책에 따라 트리거가 허용되지 않는 경우 저장 프로시저를 변경해도 동일한 결과를 얻을 수 있습니다. 데이터를 쓰는 프로시저에 몇 줄의 코드를 추가하여 동일한 변경 내용을 스테이징 테이블에 추가로 쓸 수 있습니다.

이 스테이징 테이블을 레거시 애플리케이션 테이블과 완전히 동기화하면 실시간 마이그레이션의 훌륭한 출발점이 됩니다. 이제 데이터베이스 CDC를 사용하여 라이브 마이그레이션을 수행하는 도구(예: AWS DMS)를 이 테이블에 사용할 수 있습니다. 이 접근 방식의 장점은 관계형 데이터베이스 엔진에서 사용할 수 있는 잘 알려진 SQL 기술 역량과 기능을 사용한다는 것입니다.

계획

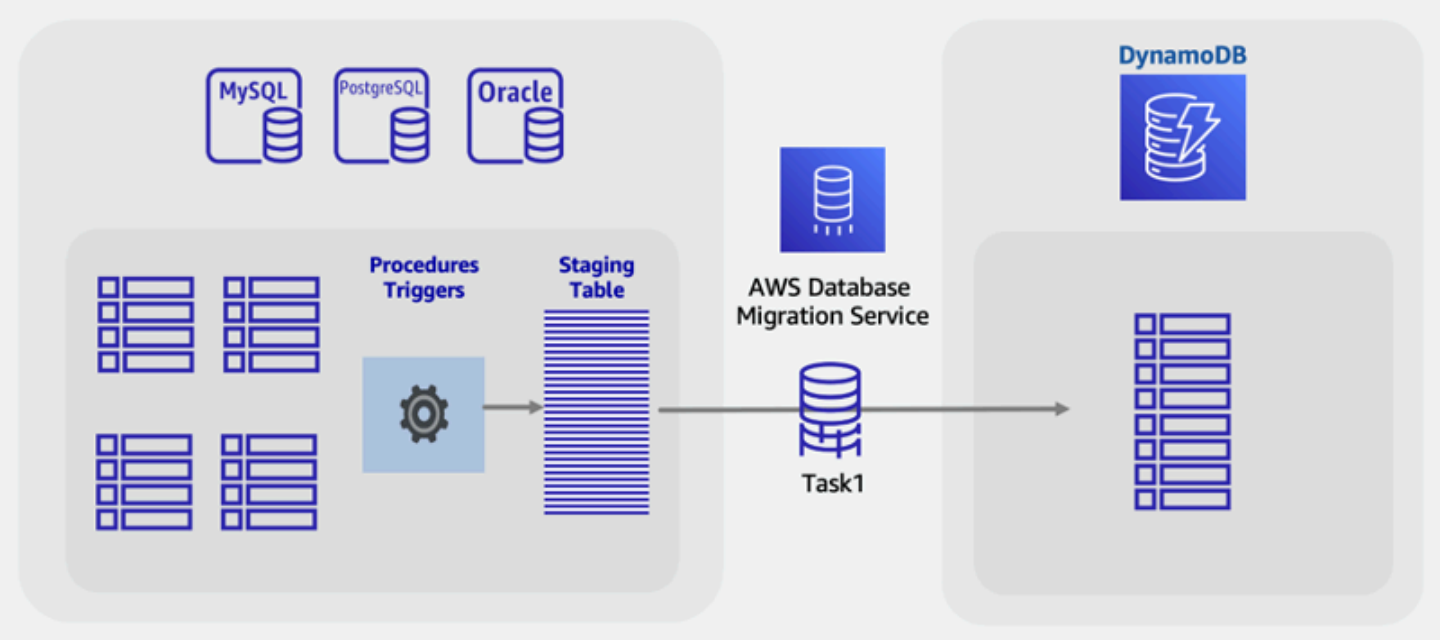
AWS DMS를 사용하여 SQL 스테이징 테이블을 활용하여 온라인 마이그레이션 수행

도구

- 사용자 지정 SQL 저장 프로시저 또는 트리거
- AWS DMS

온라인 마이그레이션 단계:

1. 소스 관계형 데이터베이스 엔진 내에 추가 디스크 공간과 처리 용량이 있는지 확인합니다.
2. 타임스탬프 또는 CDC 기능을 활성화하여 SQL 데이터베이스에 새 스테이징 테이블을 생성합니다.
3. 저장 프로시저를 작성하고 실행하여 기존 관계형 테이블 데이터를 스테이징 테이블에 복사합니다.
4. 트리거를 배포하거나 기존 프로시저를 수정하여 기존 테이블에 일반 쓰기를 수행하면서 새 스테이징 테이블에 이중 쓰기를 수행합니다.
5. AWS DMS를 실행하여 이 소스 테이블을 대상 DynamoDB 테이블로 마이그레이션하고 동기화합니다.




이 안내서에서는 가동 중지 시간을 최소화하고 일반적인 데이터베이스 도구 및 기법을 사용하는 데 중점을 두고 관계형 데이터베이스 데이터를 DynamoDB로 마이그레이션하기 위한 몇 가지 고려 사항과 접근 방식을 제시했습니다. 자세한 내용은 다음 자료를 참조하세요.

- AWS DMS 사용 설명서
- AWS Glue 사용 설명서
- Best Practices for Migrating from RDBMS to DynamoDB

# DynamoDB용 NoSQL Workbench

Amazon DynamoDB용 NoSQL Workbench는 최신 데이터베이스 개발 및 작업에 사용할 수 있는 플랫폼 간, 클라이언트 측 GUI 애플리케이션입니다. Windows, macOS 및 Linux에서 사용할 수 있습니다. NoSQL Workbench는 DynamoDB 테이블 설계, 생성, 쿼리 및 관리에 도움이 되는 데이터 모델링, 데이터 시각화 및 쿼리 개발 기능을 제공하는 시각적 개발 도구입니다. NoSQL Workbench는 이제 설치 프로세스의 선택적 부분으로 DynamoDB Local을 포함하므로 DynamoDB Local에서 데이터를 더 쉽게 모델링할 수 있습니다. DynamoDB Local 및 그 요구 사항에 대한 자세한 내용은 DynamoDB local 설정(다운로드 가능 버전) 섹션을 참조하세요.

데이터 모델링

DynamoDB용 NoSQL Workbench를 사용하여 새로운 데이터 모델을 구축하거나 애플리케이션 데이터 액세스 패턴을 충족하는 기존 데이터 모델을 기반으로 모델을 설계할 수 있습니다. 그리고 프로세스를 종료할 때 설계된 데이터 모델을 가져오고 내보낼 수도 있습니다. 자세한 내용은 NoSQL Workbench로 데이터 모델 빌드 단원을 참조하십시오.

데이터 시각화

데이터 모델 시각화 도우미에는 애플리케이션 코드를 쓰지 않고 쿼리를 매핑하고 액세스 패턴(패싯)을 시각화할 수 있는 캔버스가 있습니다. 각 패싯은 DynamoDB의 서로 다른 각 액세스 패턴에 해당합니다. 데이터 모델에서 사용할 샘플 데이터를 자동 생성할 수 있습니다. 자세한 내용은 데이터 액세스 패턴 시각화 단원을 참조하십시오.

작업 빌드

NoSQL Workbench는 쿼리 개발 및 테스트용 그래픽 사용자 인터페이스를 제공합니다. 작업 빌더를 사용하여 실시간 데이터 세트를 보고, 탐색하고, 쿼리할 수 있습니다. 구조화된 작업 빌더는 프로젝션 표현식, 조건식을 지원하고 여러 언어로 샘플 코드를 생성합니다. 한 Amazon DynamoDB 계정에서 다른 리전의 다른 계정으로 테이블을 직접 복제할 수 있습니다. 또한 DynamoDB 로컬과 Amazon DynamoDB 계정 간에 테이블을 직접 복제하여 개발 환경 간에 테이블의 키 스키마(및 선택적으로 GSI 스키마와 항목)를 더 빠르게 복사할 수 있습니다. 자세한 내용은 NoSQL Workbench로 데이터 세트 탐색 및 작업 빌드 단원을 참조하십시오.

아래 동영상에서는 NoSQL Workbench를 사용한 데이터 모델링의 개념을 자세히 설명합니다.

주제

- DynamoDB용 NoSQL Workbench 다운로드
- DynamoDB용 NoSQL Workbench 설치

- NoSQL Workbench로 데이터 모델 빌드
- 데이터 액세스 패턴 시각화
- NoSQL Workbench로 데이터 세트 탐색 및 작업 빌드
- NoSQL Workbench용 샘플 데이터 모델
- NoSQL Workbench 릴리스 기록

# DynamoDB용 NoSQL Workbench 다운로드

다음 지침에 따라 Amazon DynamoDB용 NoSQL Workbench 및 DynamoDB Local*을 다운로드합니다.

사전 조건

Ubuntu 설치에는 libfuse2와 curl이라는 두 가지 필수 소프트웨어가 필요합니다.

libfuse2

Ubuntu 22.04부터는 libfuse2가 기본적으로 설치되지 않습니다. 이 문제를 해결하려면 `sudo add-apt-repository universe && sudo apt install libfuse2`를 실행하여 최신 Ubuntu 버전을 설치하세요.

curl

Ubuntu를 업데이트합니다. `sudo apt update && sudo apt upgrade`를 실행합니다.

다음으로, cURL을 설치합니다. `sudo apt install curl`을 실행합니다.

NoSQL Workbench 및 DynamoDB Local 다운로드

1. 운영 체제에 적합한 NoSQL Workbench 버전을 다운로드합니다.

| 운영 체제 | 다운로드 링크 |
| --- | --- |
| macOS(Intel)** | macOS용 다운로드(Intel) |
| macOS(Apple 실리콘) | macOS용 다운로드(Apple 실리콘) |
| Windows | Windows용 다운로드 |

| 운영 체제 | 다운로드 링크 |
| --- | --- |
| Linux*** | Linux용 다운로드 |

* NoSQL Workbench에는 설치 프로세스의 선택적 부분으로 DynamoDB Local이 포함되어 있습니다.

** NoSQL Workbench를 열려고 할 때 식별된 개발자가 앱을 Apple에 등록하지 않았다는 경고 메시지가 나타나면 다음과 작업을 수행하세요.

1. 앱을 찾아 엽니다.
2. Control을 누른 상태로 앱 아이콘을 클릭한 다음 단축키 메뉴에서 열기를 선택합니다.

   이렇게 하면 앱이 보안 설정의 예외로 저장됩니다. 등록된 다른 앱을 열 때처럼 앱을 두 번 클릭하여 엽니다.

*** NoSQL Workbench는 Ubuntu 12.04, Fedora 21, Debian 8 또는 이러한 Linux 배포의 최신 버전을 지원합니다.

2. 다운로드한 애플리케이션을 시작하고, 단계에 따라 NoSQL Workbench를 설치합니다.

> **Note**
>
> DynamoDB Local을 실행하려면 Java 런타임 환경(JRE) 버전 11.x 이상이 필요합니다.

# DynamoDB용 NoSQL Workbench 설치

지원되는 플랫폼에 NoSQL Workbench 및 DynamoDB Local을 설치하려면 다음 단계를 따르세요.

Windows

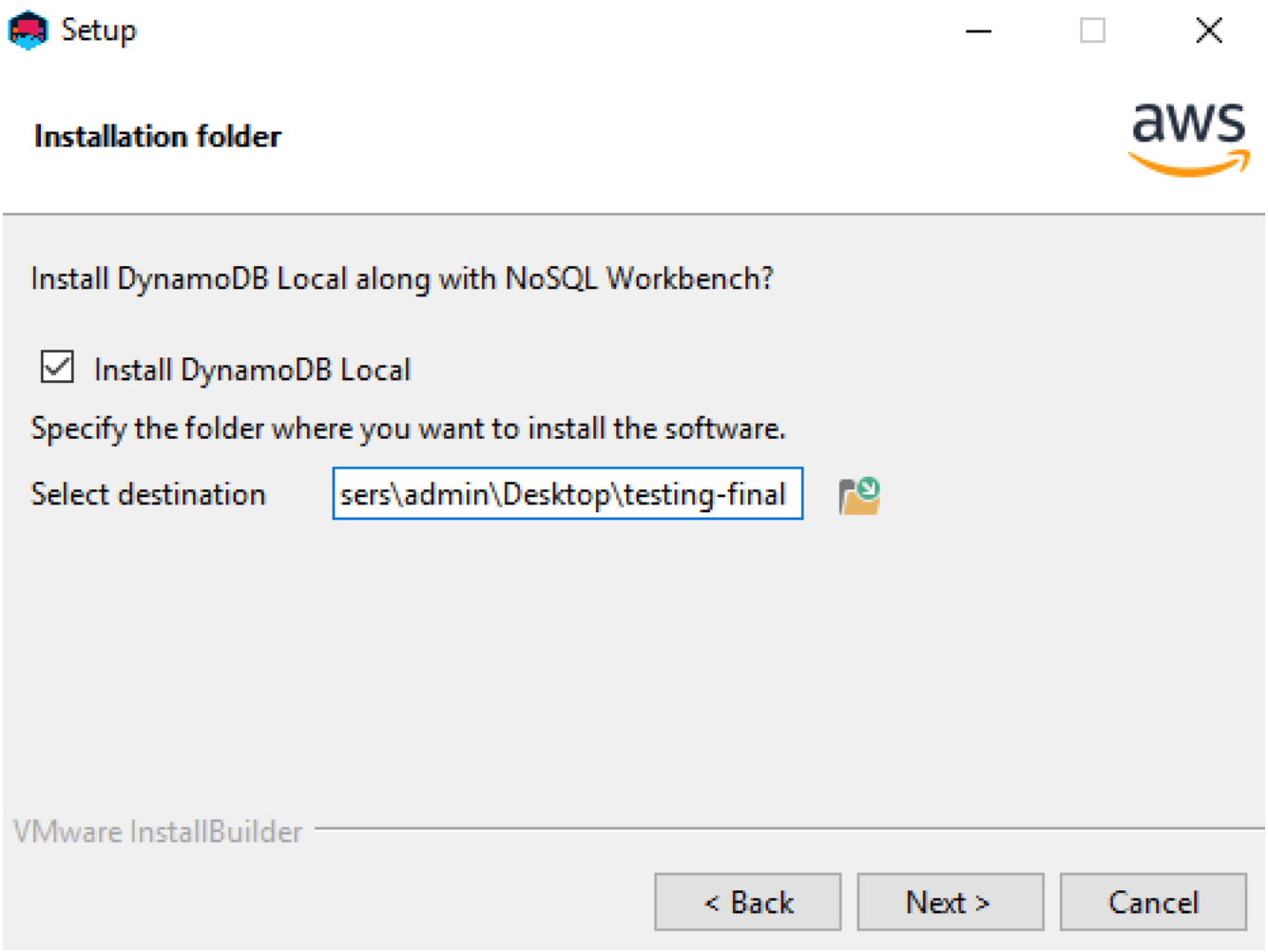
Windows에 NoSQL Workbench 설치

1. NoSQL Workbench 설치 프로그램 애플리케이션을 실행하고 설치 언어를 선택합니다. 그런 다음 OK(확인)를 선택하여 설정을 시작합니다. NoSQL Workbench 다운로드에 대한 자세한 내용은 DynamoDB용 NoSQL Workbench 다운로드 섹션을 참조하세요.
2. Next(다음)를 선택하여 설정을 계속한 후 다음 화면에서 Next(다음)를 선택합니다.

3. 기본적으로 DynamoDB Local 설치 확인란은 설치의 일부로 DynamoDB Local을 포함하도록 선택되어 있습니다. 이 옵션을 선택된 상태로 두면 DynamoDB Local이 설치되고, 대상 경로는 NoSQL Workbench의 설치 경로와 같습니다. 이 옵션의 확인란을 선택 취소하면 DynamoDB Local 설치가 생략되고 설치 경로에는 NoSQL Workbench만 설치됩니다.

   소프트웨어를 설치할 대상 위치를 선택하고 Next(다음)를 선택합니다.

   > **Note**
   >
   > 설치의 일부로 DynamoDB Local을 포함하지 않기로 선택한 경우 DynamoDB Local 설치 확인란의 선택을 취소하고 다음을 선택한 후 6단계로 건너뛰세요. 나중에 DynamoDB Local을 독립 실행형 설치로 별도로 다운로드할 수 있습니다. 자세한 내용은 DynamoDB local 설정(다운로드 가능 버전) 단원을 참조하십시오.



4. 사용할 DynamoDB Local의 포트 번호를 선택합니다. 기본 포트는 8000입니다. 포트 번호를 입력한 후 Next(다음)를 선택합니다.
5. Next(다음)를 선택하여 설정을 시작합니다.
6. 설정이 완료되면 Finish(마침)를 선택하여 설정 화면을 닫습니다.
7. 설치 경로(예: /programs/DynamoDBWorkbench)에서 애플리케이션을 엽니다.

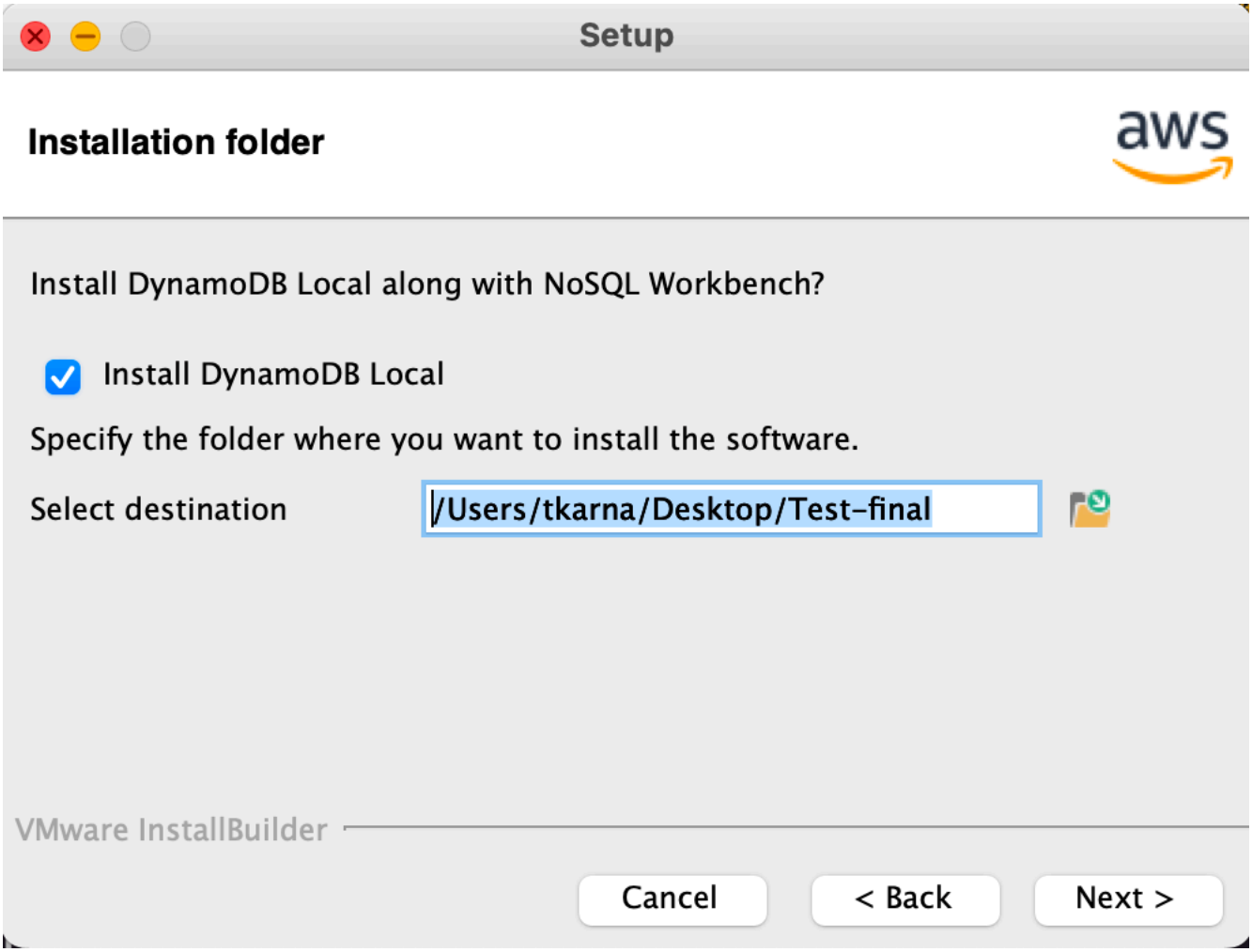
macOS

macOS에 NoSQL Workbench 설치

1. NoSQL Workbench 설치 프로그램 애플리케이션을 실행하고 설치 언어를 선택합니다. 그런 다음 OK(확인)를 선택하여 설정을 시작합니다. NoSQL Workbench 다운로드에 대한 자세한 내용은 DynamoDB용 NoSQL Workbench 다운로드 섹션을 참조하세요.
2. Next(다음)를 선택하여 설정을 계속한 후 다음 화면에서 Next(다음)를 선택합니다.
3. 기본적으로 DynamoDB Local 설치 확인란은 설치의 일부로 DynamoDB Local을 포함하도록 선택되어 있습니다. 이 옵션을 선택된 상태로 두면 DynamoDB Local이 설치되고, 대상 경로는 NoSQL Workbench의 설치 경로와 같습니다. 이 옵션을 선택 취소하면 DynamoDB Local 설치가 생략되고 설치 경로에는 NoSQL Workbench만 설치됩니다.

   소프트웨어를 설치할 대상 위치를 선택하고 Next(다음)를 선택합니다.

   > **Note**
   >
   > 설치의 일부로 DynamoDB Local을 포함하지 않기로 선택한 경우 DynamoDB Local 설치 확인란의 선택을 취소하고 다음을 선택한 후 6단계로 건너뛰세요. 나중에 DynamoDB Local을 독립 실행형 설치로 별도로 다운로드할 수 있습니다. 자세한 내용은 DynamoDB local 설정(다운로드 가능 버전) 단원을 참조하십시오.




4. 사용할 DynamoDB Local의 포트 번호를 선택합니다. 기본 포트는 8000입니다. 포트 번호를 입력한 후 Next(다음)를 선택합니다.
5. Next(다음)를 선택하여 설정을 시작합니다.
6. 설정이 완료되면 Finish(마침)를 선택하여 설정 화면을 닫습니다.
7. 설치 경로(예: /Applications/DynamoDBWorkbench)에서 애플리케이션을 엽니다.

> **Note**
>
> macOS용 NoSQL Workbench는 자동 업데이트를 수행합니다. 업데이트에 대한 알림을 받으려면 System Preferences(시스템 환경설정) > Notifications(알림)에서 NoSQL Workbench에 대한 알림 액세스를 활성화하세요.

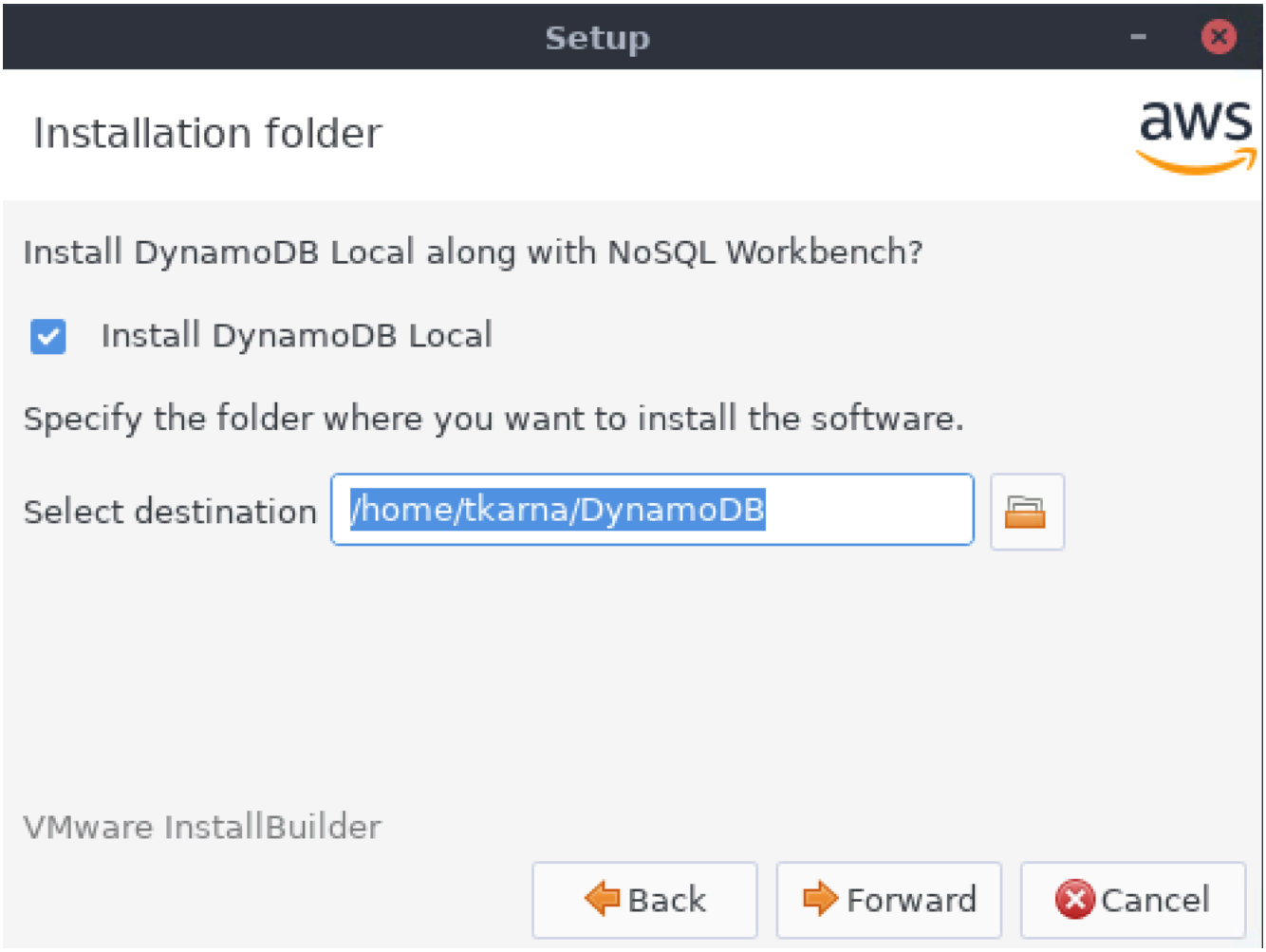
## Linux

### Linux에 NoSQL Workbench 설치

1. NoSQL Workbench 설치 프로그램 애플리케이션을 실행하고 설치 언어를 선택합니다. 그런 다음 OK(확인)를 선택하여 설정을 시작합니다. NoSQL Workbench 다운로드에 대한 자세한 내용은 DynamoDB용 NoSQL Workbench 다운로드 섹션을 참조하세요.
2. 설정을 계속하려면 Forward(다음)를 선택하고 다음 화면에서 Forward(다음)를 선택합니다.
3. 기본적으로 DynamoDB Local 설치 확인란은 설치의 일부로 DynamoDB Local을 포함하도록 선택되어 있습니다. 이 옵션을 선택된 상태로 두면 DynamoDB Local이 설치되고, 대상 경로는 NoSQL Workbench의 설치 경로와 같습니다. 이 옵션을 선택 취소하면 DynamoDB Local 설치가 생략되고 설치 경로에는 NoSQL Workbench만 설치됩니다.

   소프트웨어를 설치할 대상 위치를 선택하고 Forward(다음)를 선택합니다.

> **Note**
>
> 설치의 일부로 DynamoDB Local을 포함하지 않기로 선택한 경우 DynamoDB Local 설치 확인란의 선택을 취소하고 다음을 선택한 후 6단계로 건너뛰세요. 나중에 DynamoDB Local을 독립 실행형 설치로 별도로 다운로드할 수 있습니다. 자세한 내용은 DynamoDB local 설정(다운로드 가능 버전) 단원을 참조하십시오.




4. 사용할 DynamoDB Local의 포트 번호를 선택합니다. 기본 포트는 8000입니다. 포트 번호를 입력한 후 Forward(다음)를 선택합니다.
5. Forward(다음)를 선택하여 설치를 시작합니다.
6. 설정이 완료되면 Finish(마침)를 선택하여 설정 화면을 닫습니다.
7. 설치 경로(예: /usr/local/programs/DynamoDBWorkbench/)에서 애플리케이션을 엽니다.

Linux에서 AppImage를 시작하려면

1. AppImage 실행 파일 만들기:

   chmod +x noSQL-workbench-linux.AppImage

   noSQL-workbench-linux.AppImage를 다운로드한 AppImage의 실제 파일 이름으로 바꿉니다.
2. 다음 AppImage를 실행합니다.

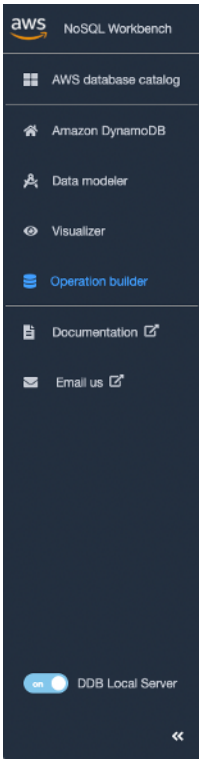
./noSQL-workbench-linux.AppImage

그러면 NoSQL Workbench 애플리케이션이 시작됩니다.

> **Note**
>
> Linux 배포판에 따라 AppImage가 제대로 실행되도록 추가 종속성을 설치해야 할 수 있습니다. 문제가 발생하면 AppImage 개발자가 제공한 설명서를 참조하거나 커뮤니티의 지원을 요청하세요.

> **Note**
>
> NoSQL Workbench 설치의 일부로 DynamoDB Local을 설치하기로 선택한 경우 DynamoDB Local이 기본 옵션으로 사전 구성됩니다. 기본 옵션을 편집하려면 /resources/DDBlocal_scripts/ 디렉터리에 있는 DDBlocalStart 스크립트를 수정합니다. 설치 중에 제공한 경로에서 찾을 수 있습니다. DynamoDB Local 옵션에 대한 자세한 내용은 DynamoDB local 사용 참고 사항 섹션을 참조하세요.

NoSQL Workbench 설치의 일부로 DynamoDB Local을 설치하기로 선택한 경우 다음 이미지와 같이 DynamoDB Local을 활성화 및 비활성화할 수 있는 토글에 액세스할 수 있습니다.




# NoSQL Workbench로 데이터 모델 빌드

Amazon DynamoDB용 NoSQL Workbench의 데이터 모델 제작자 도구를 사용하여 새로운 데이터 모델을 빌드하거나 애플리케이션 데이터 액세스 패턴을 충족하는 기존 데이터 모델을 기반으로 모델을 설계할 수 있습니다. 데이터 모델 제작자에는 시작하는 데 도움이 되는 몇 가지 샘플 데이터 모델이 포함되어 있습니다.

주제

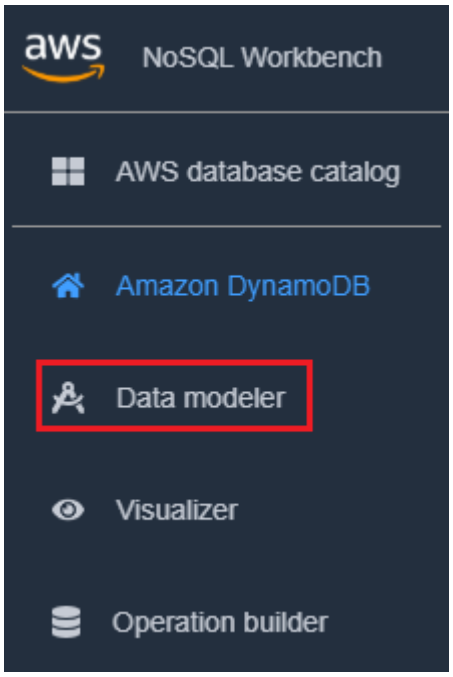
- 새 데이터 모델 생성
- 기존 데이터 모델 가져오기
- 데이터 모델 내보내기
- 기존 데이터 모델 편집

## 새 데이터 모델 생성

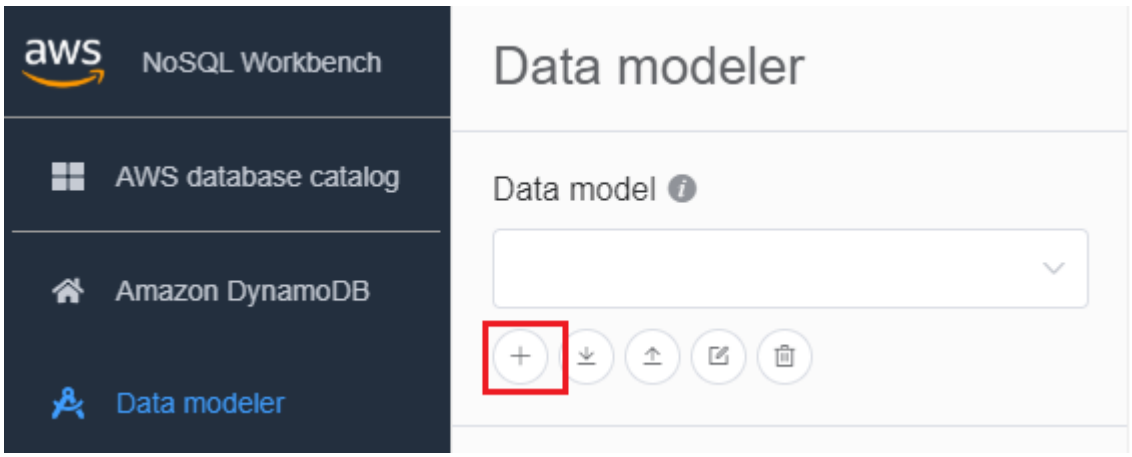
다음 단계에 따라 NoSQL Workbench를 사용하여 Amazon DynamoDB에 새 데이터 모델을 생성합니다.

## 새 데이터 모델 생성

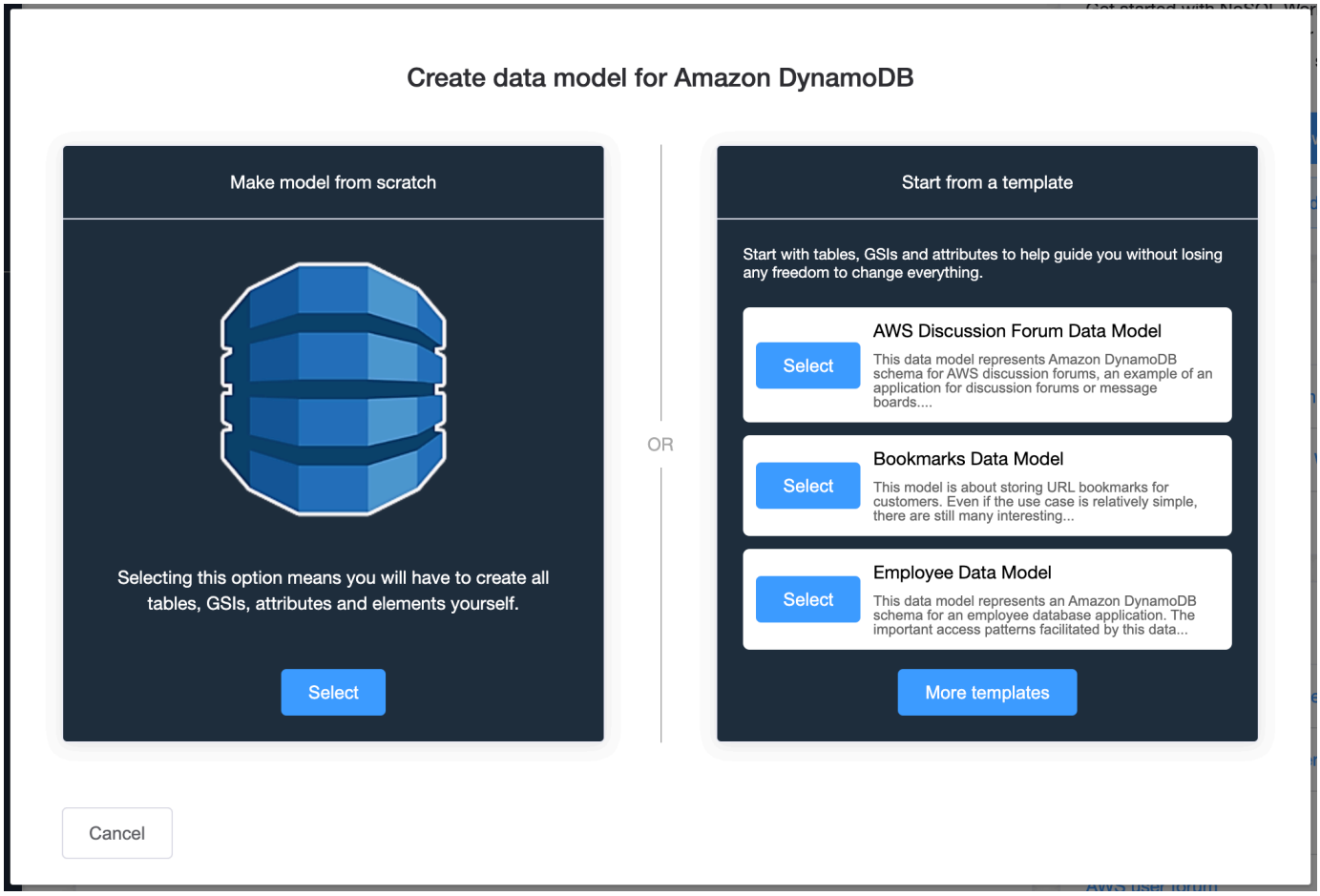
1. NoSQL Workbench를 열고 왼쪽 탐색 창에서 데이터 모델 제작자 아이콘을 선택합니다.




2. Create data model(데이터 모델 생성)을 선택합니다.

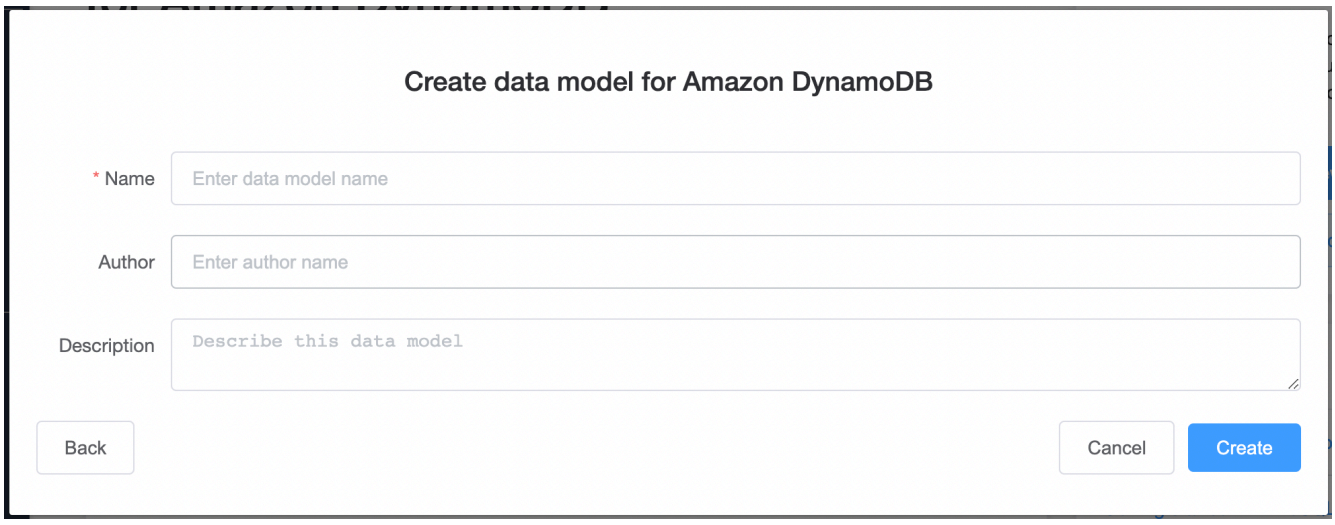



데이터 모델 생성(Create data model)에는 모델을 처음부터 만들거나 템플릿으로 시작하는 두 가지 옵션이 있습니다.




Make model from scratch

모델을 처음부터 만들려면 데이터 모델의 이름, 작성자 및 설명을 입력합니다. 마친 후에는 Create(생성)를 선택합니다.



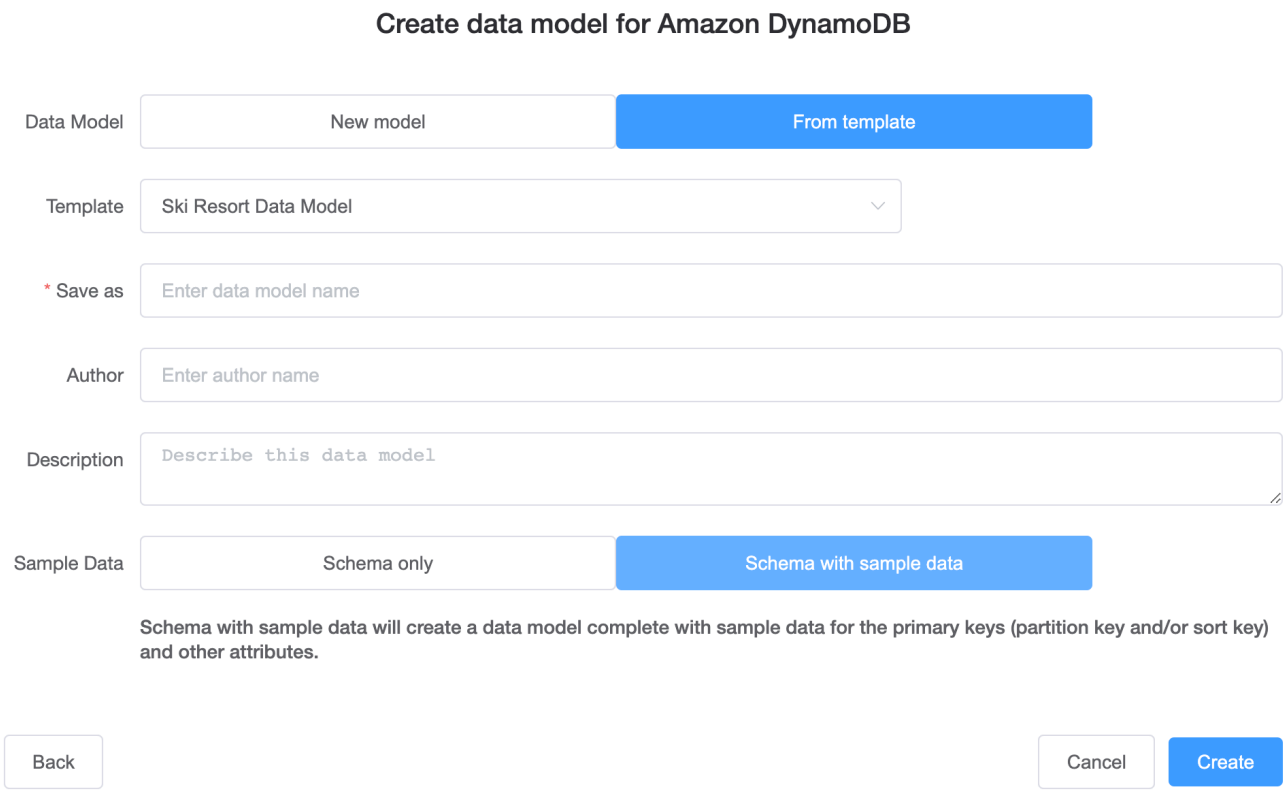
Start from a template

템플릿으로 시작하면 시작하는 데 사용할 샘플 모델을 선택할 수 있습니다. 더 많은 템플릿 옵션을 보려면 More templates(추가 템플릿)를 선택합니다. 사용할 템플릿에서 Select(선택)를 선택합니다.

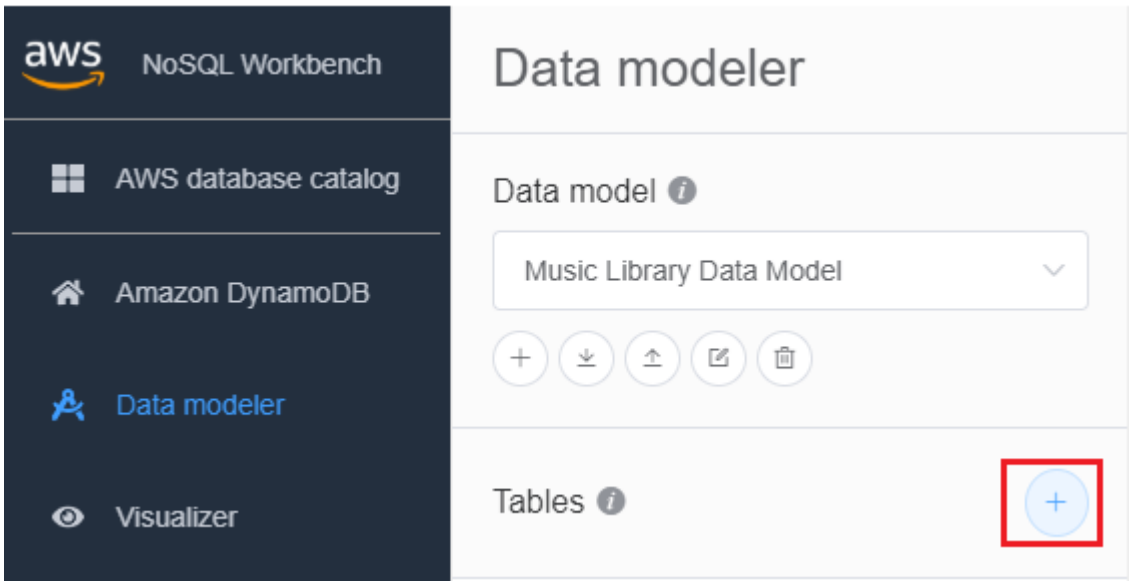
선택한 템플릿의 데이터 모델 이름, 작성자 및 설명을 입력합니다. Schema only(스키마 전용) 및 Schema with sample data(샘플 데이터가 포함된 스키마) 중에서 선택할 수 있습니다.

- Schema only(스키마 전용)를 선택하면 프라이머리 키(파티션 및 정렬 키) 및 기타 속성을 사용하여 빈 데이터 모델이 생성됩니다.
- Schema with sample data(샘플 데이터가 포함된 스키마)를 선택하면 프라이머리 키(파티션 및 정렬 키) 및 기타 속성에 대한 샘플 데이터가 포함된 완전한 데이터 모델이 생성됩니다.

이 정보를 모두 입력했으면 Create(생성)를 선택하여 모델을 생성합니다.




3. 모델을 생성한 후 Add table(테이블 추가)을 선택합니다.




테이블에 대한 자세한 내용은 DynamoDB의 테이블 작업을 참조하세요.

4. 다음을 지정합니다.

   - 테이블 이름 – 고유한 테이블 이름을 입력합니다.
   - 파티션 키 – 파티션 키 이름을 입력하고 그 형식을 지정합니다. 필요에 따라 샘플 데이터 생성을 위해 보다 세분화된 데이터 유형 형식을 선택할 수도 있습니다.
   - 정렬 키를 추가할 경우

     1. [Add sort key]를 선택합니다.
     2. 정렬 키 이름과 그 형식을 지정합니다. 필요에 따라 샘플 데이터 생성을 위해 보다 세분화된 데이터 유형 형식을 선택할 수 있습니다.

   > **Note**
   >
   > 프라이머리 키 설계, 효과적인 파티션 키 설계 및 사용, 정렬 키 사용에 대한 자세한 내용은 다음을 참조하세요.
   >
   > - 프라이머리 키
   > - DynamoDB에서 효과적으로 파티션 키를 설계해 사용하는 모범 사례입니다.
   > - 정렬 키를 사용하여 DynamoDB의 데이터를 정리하는 모범 사례

5. 기타 속성을 추가할 때는 각 속성에 다음을 수행합니다.

   1. 속성 추가를 선택합니다.

2. 속성 이름과 형식을 지정합니다. 필요에 따라 샘플 데이터 생성을 위해 보다 세분화된 데이터 유형 형식을 선택할 수 있습니다.

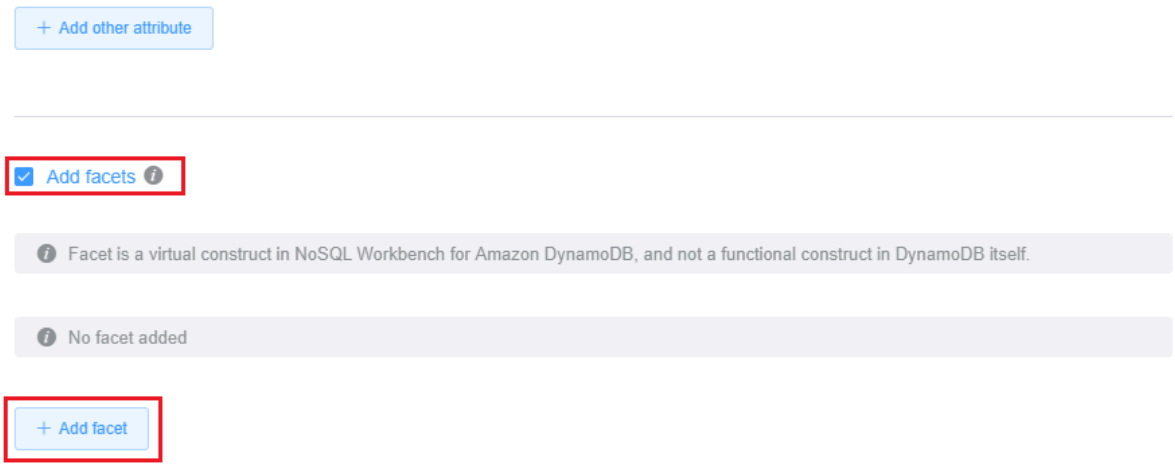
6. 패싯 추가:

필요에 따라 패싯을 추가할 수 있습니다. 패싯은 NoSQL Workbench의 가상 구성입니다. DynamoDB 자체에서는 함수 구성이 아닙니다.

> **Note**
>
> NoSQL Workbench의 패싯은 테이블의 일부 데이터만 사용하여 Amazon DynamoDB에 대한 애플리케이션의 다양한 데이터 액세스 패턴을 시각화하는 데 도움이 됩니다. 패싯에 대해 자세히 알아보려면 데이터 액세스 패턴 보기 섹션을 참조하세요.

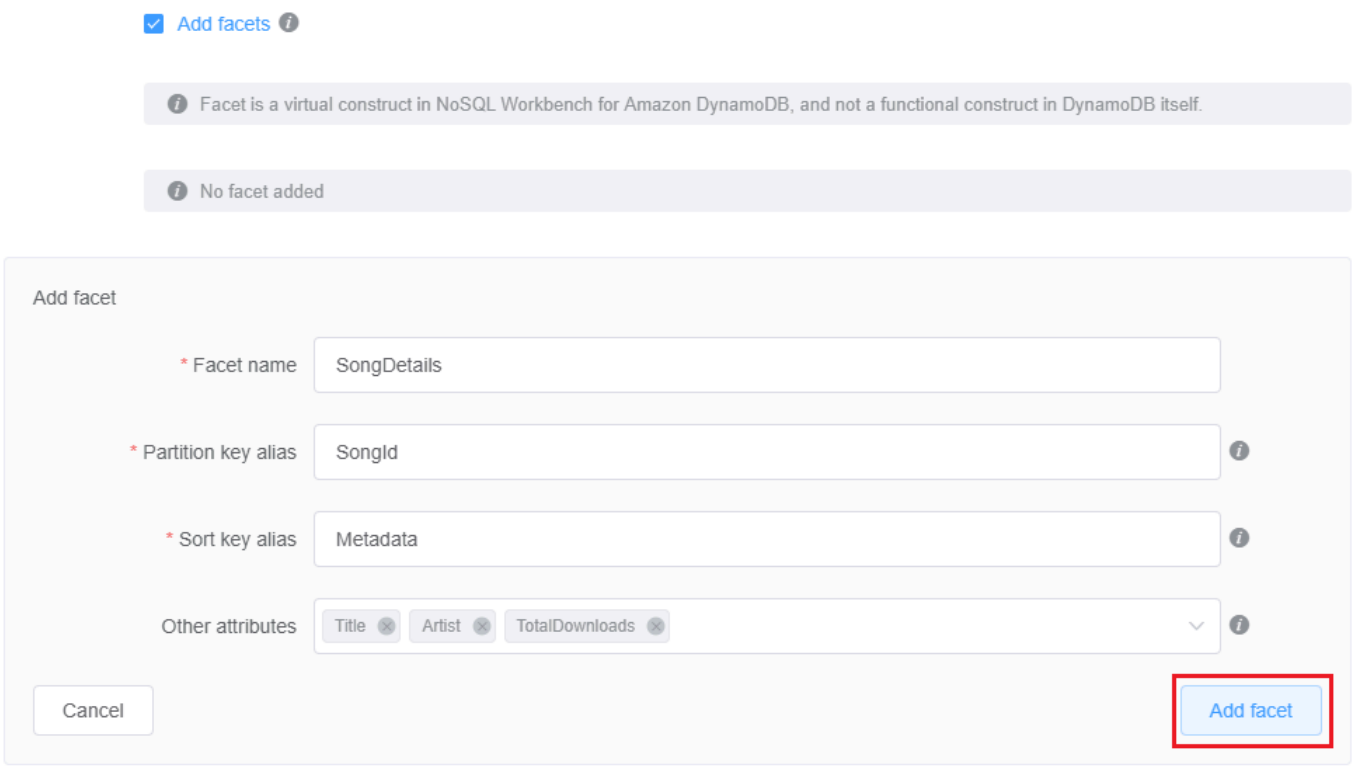
패싯을 추가하려면

- 패싯 추가를 선택합니다.
- 패싯 추가를 선택합니다.




- 다음을 지정합니다.
  - Facet name(패싯 이름)
  - 파티션 키 별칭은 이 패싯 보기를 구분하는 데 도움이 됩니다.
  - Sort key alias(정렬 키 별칭)
  - 이 패싯의 일부인 기타 속성을 선택합니다.

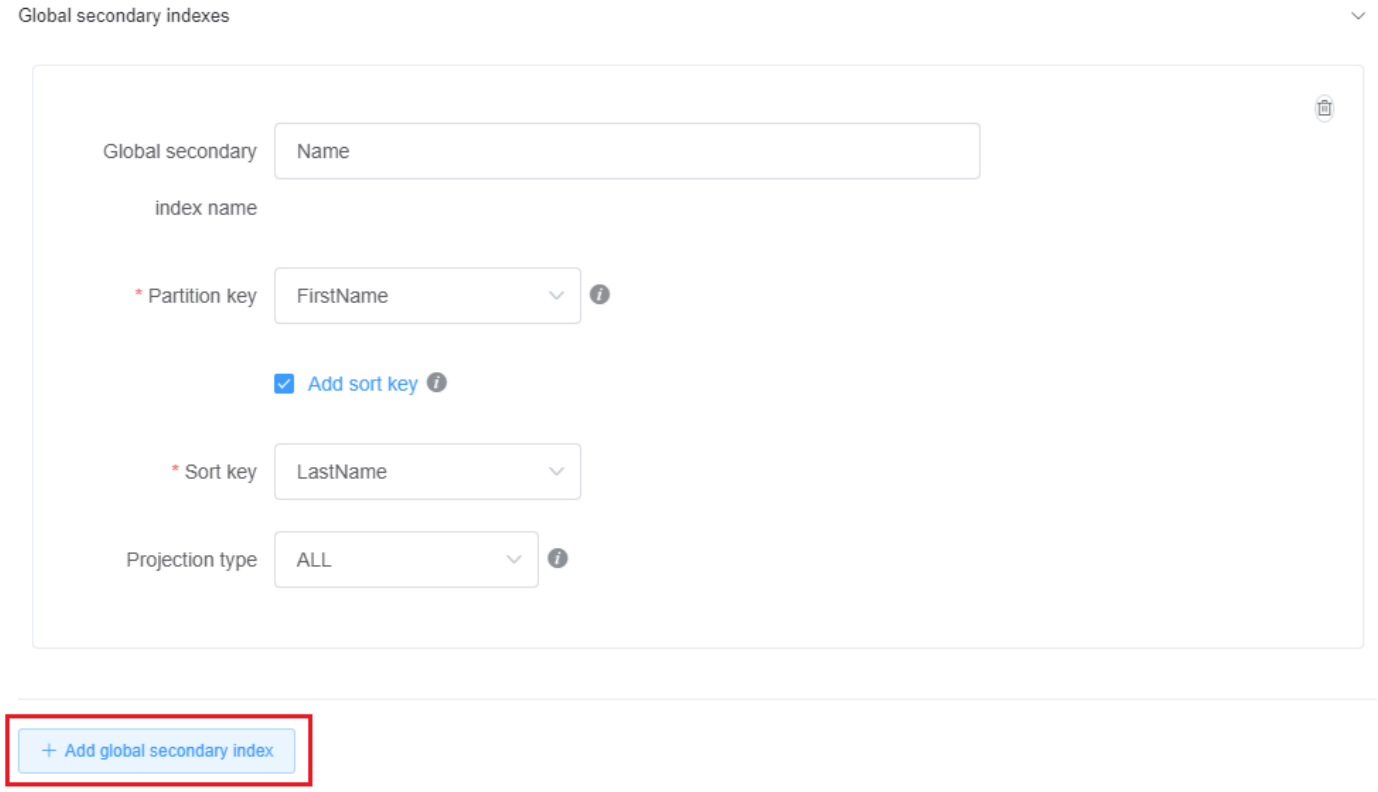
패싯 추가를 선택합니다.




패싯을 더 추가하려면 이 단계를 반복합니다.

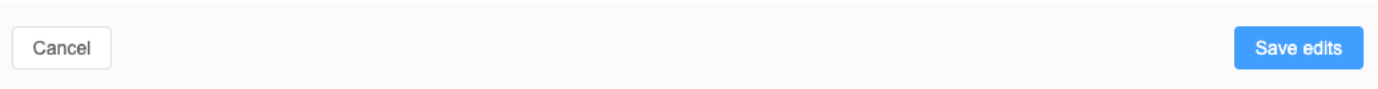
7. 전역 보조 인덱스를 추가할 경우에는 Add global secondary index(전역 보조 인덱스 추가)를 선택합니다.

   Global secondary index name(전역 보조 인덱스 이름), 파티션 키 속성 및 Projection type(프로젝션 형식)을 지정합니다.




DynamoDB의 글로벌 보조 인덱스 작업에 대한 자세한 내용은 글로벌 보조 인덱스를 참조하세요.

8. 편집 내용을 테이블 설정에 저장합니다.



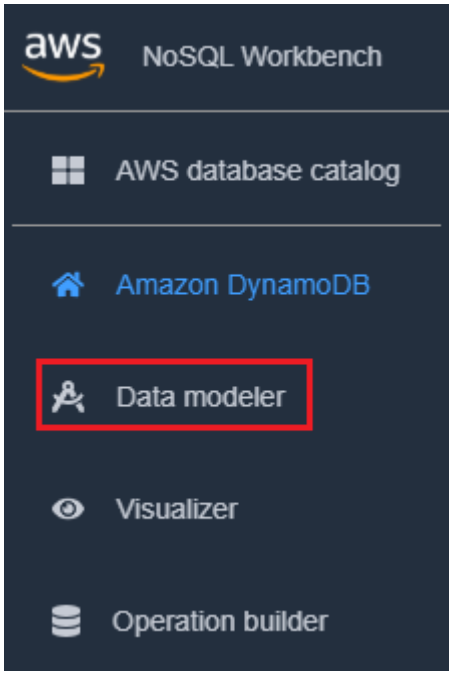

`CreateTable` API 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조의 CreateTable을 참조하세요.

## 기존 데이터 모델 가져오기

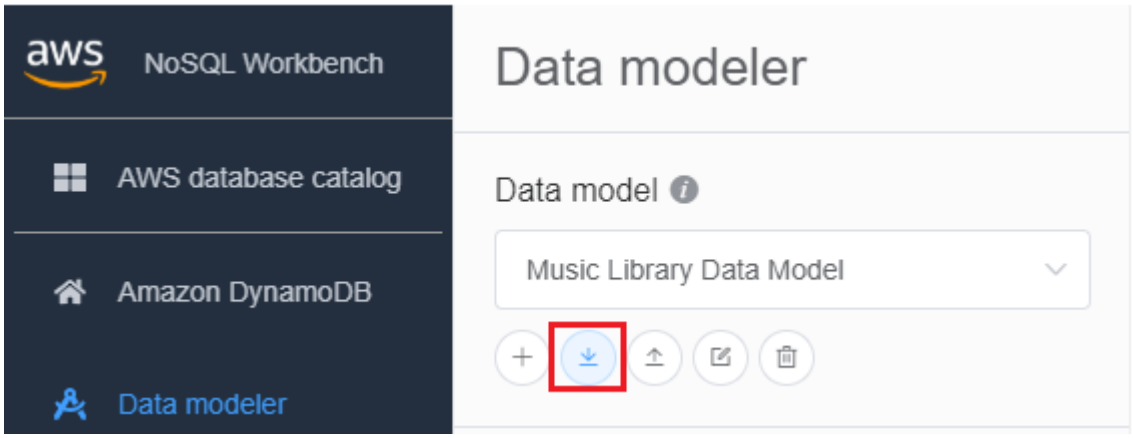
Amazon DynamoDB용 NoSQL Workbench를 사용하여 기존 모델을 가져와 수정하여 데이터 모델을 구축할 수 있습니다. 데이터 모델은 NoSQL Workbench 모델 형식이나 AWS CloudFormation JSON 템플릿 형식으로 가져올 수 있습니다.

데이터 모델 가져오기

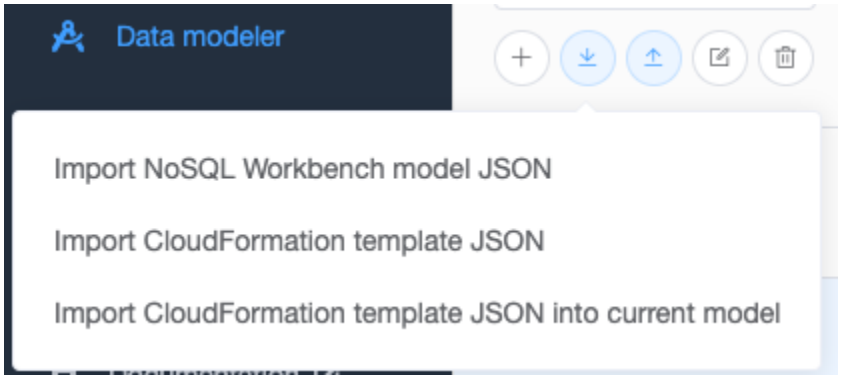
1. NoSQL Workbench의 왼쪽 탐색 창에서 데이터 모델 제작자 아이콘을 선택합니다.



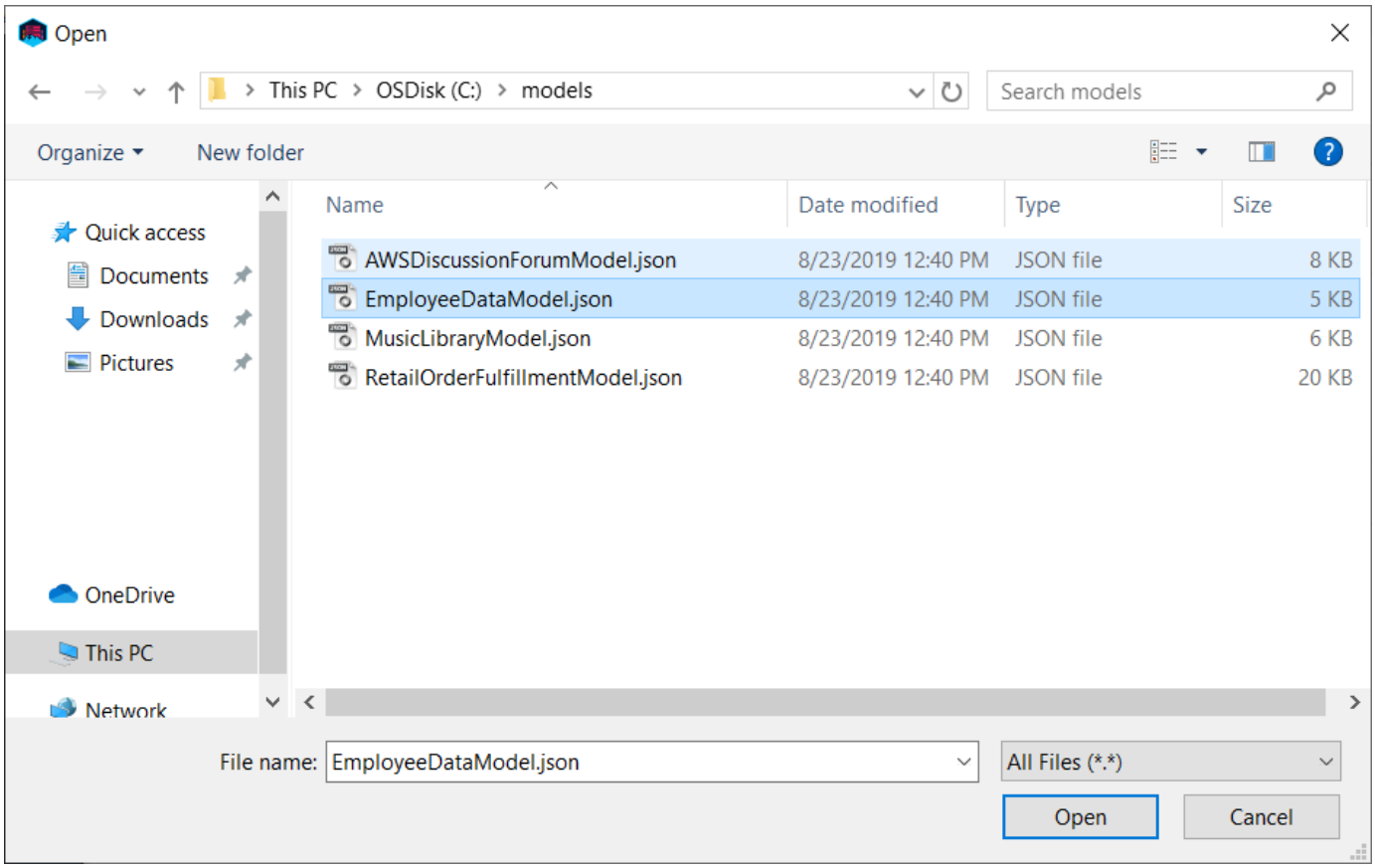

2. Import data model(데이터 모델 가져오기)을 마우스로 가리킵니다.



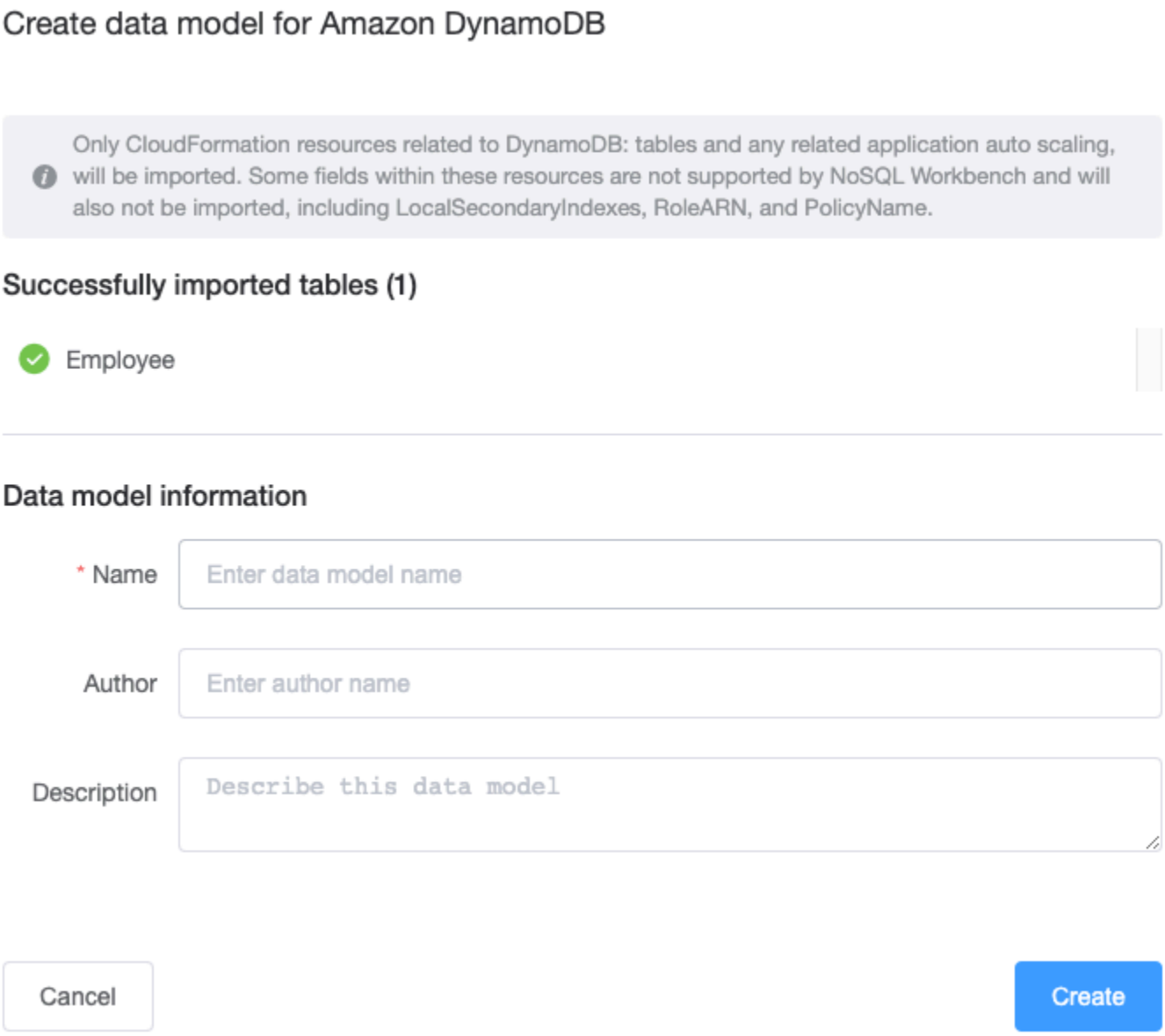

드롭다운 목록에서 가져올 모델이 NoSQL Workbench 모델 형식인지 CloudFormation JSON 템플릿 형식인지 선택합니다. NoSQL Workbench에서 기존 데이터 모델을 연 경우 CloudFormation 템플릿을 현재 모델로 가져오는 옵션이 있습니다.



3. 가져올 모델을 선택합니다.




4. 가져오는 모델이 CloudFormation 템플릿 형식인 경우 가져올 테이블 목록이 표시되며 데이터 모델 이름, 작성자, 설명을 지정할 수 있습니다.

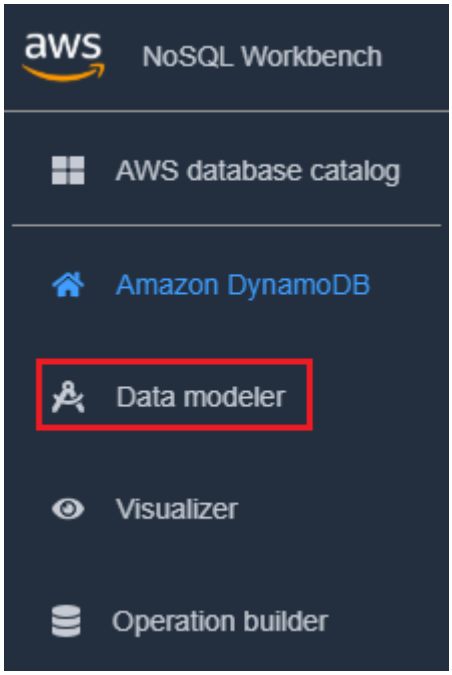



## 데이터 모델 내보내기

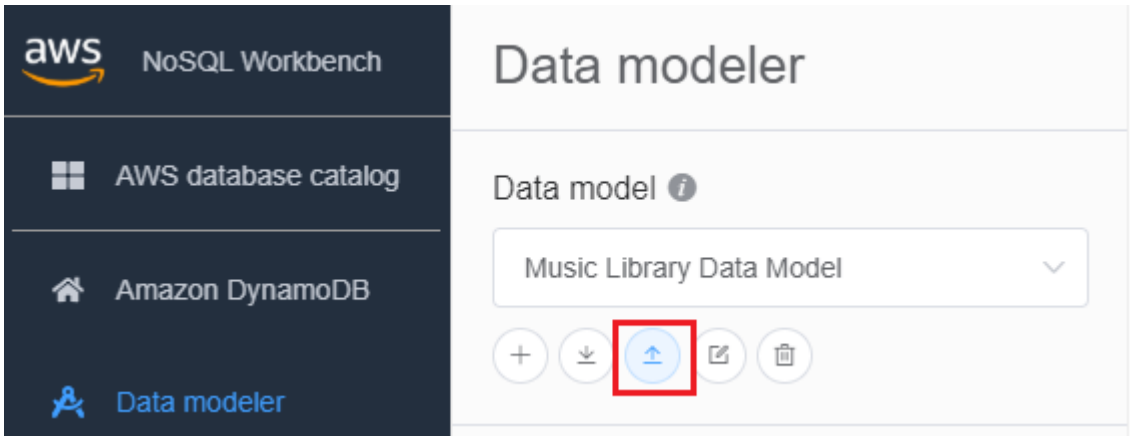
Amazon DynamoDB용 NoSQL Workbench를 사용하여 데이터 모델을 생성한 후에는 NoSQL Workbench 모델 형식 또는AWS CloudFormation JSON 템플릿 형식 중 하나로 모델을 저장하고 내보낼 수 있습니다.

데이터 모델 내보내기

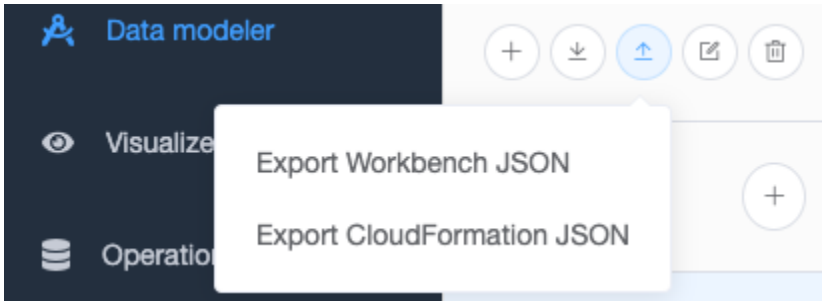
1. NoSQL Workbench의 왼쪽 탐색 창에서 데이터 모델 제작자 아이콘을 선택합니다.



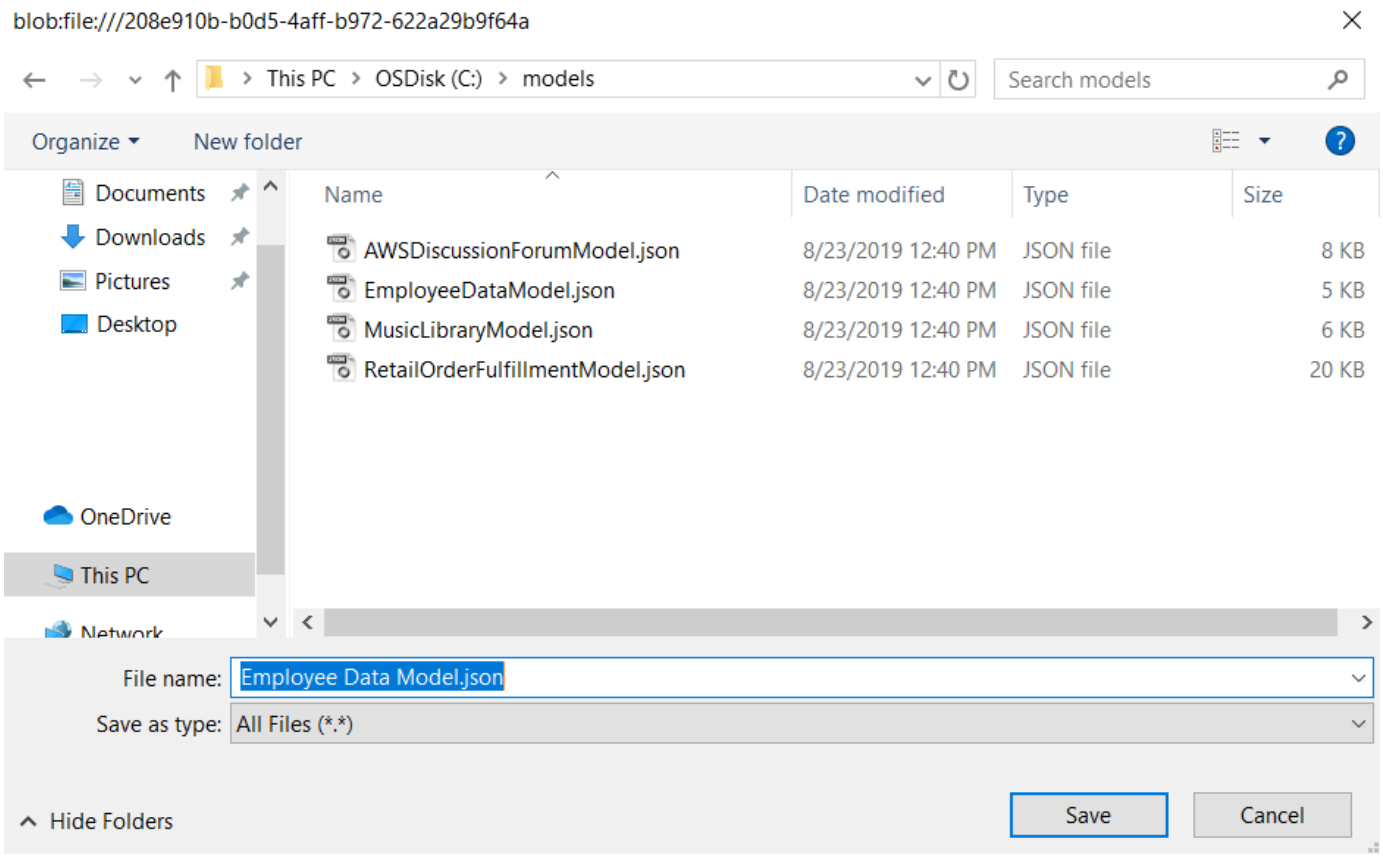

2. Export data model(데이터 모델 내보내기)을 마우스로 가리킵니다.




드롭다운 목록에서 데이터 모델을 NoSQL Workbench 모델 형식으로 내보낼지 CloudFormation JSON 템플릿 형식으로 내보낼지 선택합니다.



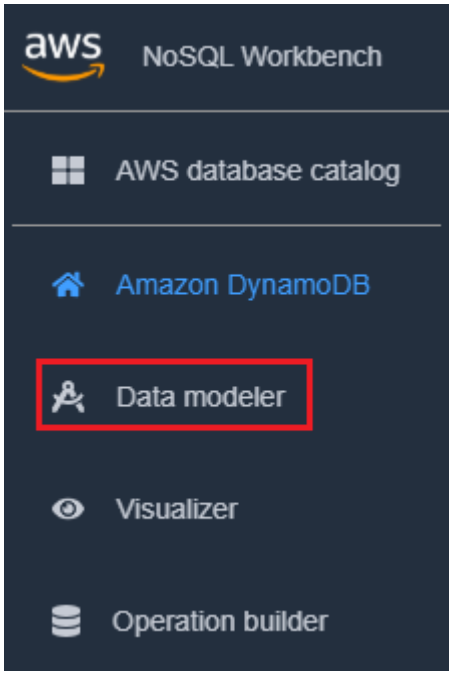

3. 모델을 저장할 위치를 선택합니다.



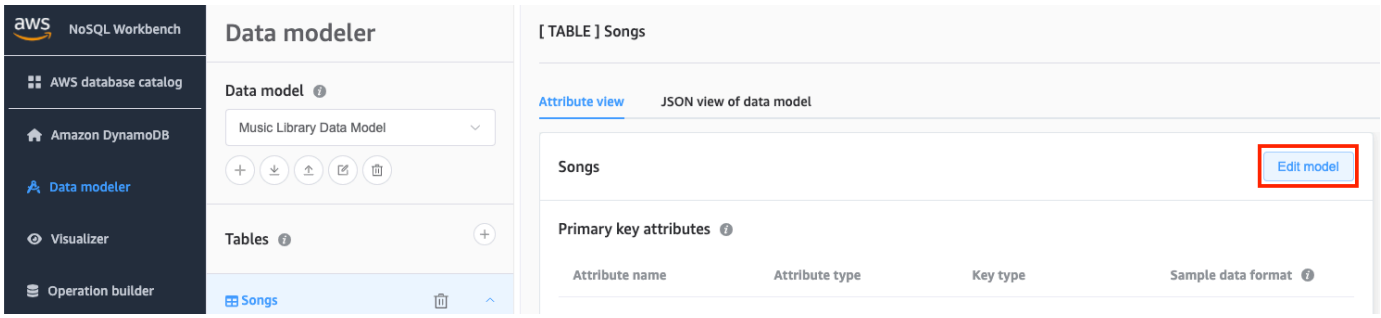

# 기존 데이터 모델 편집

기존 모델을 편집하려면

1. NoSQL Workbench의 좌측 탐색 창에서 Data modeler(데이터 모델 제작자) 버튼을 선택합니다.




2. 데이터 모델을 선택하고 편집할 테이블을 선택합니다. 모델 편집을 선택합니다.




3. 필요한 사항을 편집한 후 Save edits(편집 저장)를 선택합니다.

기존 모델을 수동으로 편집하고 패싯을 추가하려면

1. 모델을 내보냅니다. 자세한 내용은 데이터 모델 내보내기 단원을 참조하십시오.
2. 편집기에서 내보낸 파일을 엽니다.
3. 패싯을 생성할 테이블의 `DataModel` 객체를 찾습니다.

   테이블의 모든 패싯을 나타내는 `TableFacets` 어레이를 추가합니다.

   각 패싯마다 `TableFacets` 어레이에 객체를 추가합니다. 각 어레이 요소에는 다음과 같은 속성이 있습니다.

   - `FacetName` – 패싯의 이름입니다. 이 값은 모델 전체에서 고유해야 합니다.

- PartitionKeyAlias – 테이블 파티션 키의 친숙한 이름입니다. NoSQL Workbench에서 패싯을 볼 때 이 별칭이 표시됩니다.
- SortKeyAlias – 테이블 정렬 키의 친숙한 이름입니다. NoSQL Workbench에서 패싯을 볼 때 이 별칭이 표시됩니다. 테이블에 정의된 정렬 키가 없는 경우에는 이 속성이 필요 없습니다.
- NonKeyAttributes – 액세스 패턴용으로 필요한 속성 이름의 배열입니다. 이 이름은 테이블에 정의된 속성 이름으로 매핑되어야 합니다.

```
{
  "ModelName": "Music Library Data Model",
  "DataModel": [
    {
      "TableName": "Songs",
      "KeyAttributes": {
        "PartitionKey": {
          "AttributeName": "Id",
          "AttributeType": "S"
        },
        "SortKey": {
          "AttributeName": "Metadata",
          "AttributeType": "S"
        }
      },
      "NonKeyAttributes": [
        {
          "AttributeName": "DownloadMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "TotalDownloadsInMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Title",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Artist",
          "AttributeType": "S"
        },
        {
          "AttributeName": "TotalDownloads",
```

```
          "AttributeType": "S"
        },
        {
          "AttributeName": "DownloadTimestamp",
          "AttributeType": "S"
        }
      ],
      "TableFacets": [
        {
          "FacetName": "SongDetails",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "Title",
            "Artist",
            "TotalDownloads"
          ]
        },
        {
          "FacetName": "Downloads",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "DownloadTimestamp"
          ]
        }
      ]
    }
  ]
}
```

4. 이제 수정된 모델을 NoSQL Workbench로 가져올 수 있습니다. 자세한 내용은 기존 데이터 모델 가져오기 단원을 참조하십시오.

# 데이터 액세스 패턴 시각화

Amazon DynamoDB용 NoSQL Workbench의 시각화 도우미 도구를 사용하여 쿼리를 매핑하고 애플리케이션의 다양한 액세스 패턴(패싯이라고 함)을 시각화할 수 있습니다. 각 패싯은 DynamoDB의 서로

다른 각 액세스 패턴에 해당합니다. 수동으로 데이터 모델에 데이터를 추가하거나 MySQL에서 데이터를 가져올 수도 있습니다.

주제

- 데이터 모델에 샘플 데이터 추가하기
- CSV 파일에서 샘플 데이터 가져오기
- 데이터 액세스 패턴 보기
- 집계 보기를 사용하여 데이터 모델에서 모든 테이블 보기
- 데이터 모델을 DynamoDB로 커밋

## 데이터 모델에 샘플 데이터 추가하기

모델에 샘플 데이터를 추가하면 모델과 그 다양한 데이터 액세스 패턴, 즉 패싯을 시각화할 때 데이터를 표시할 수 있습니다.

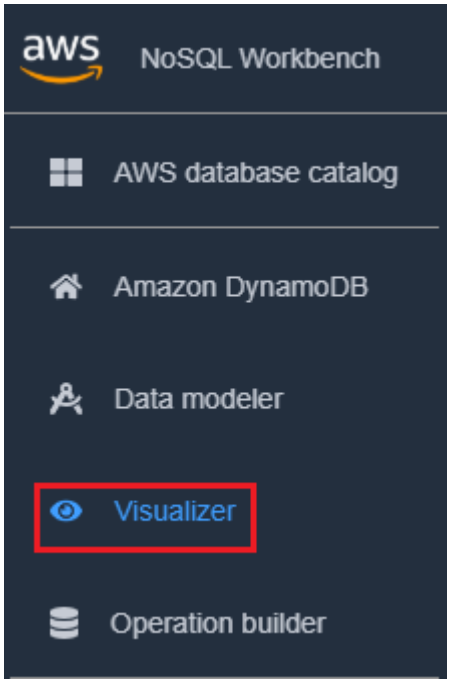
샘플 데이터를 추가하는 방법에는 두 가지가 있습니다. 한 가지 방법은 샘플 데이터 자동 생성 도구를 사용하는 것이고, 다른 방법은 데이터를 한 번에 하나씩 추가하는 것입니다.

다음 단계에 따라 Amazon DynamoDB용 NoSQL Workbench를 사용하여 데이터 모델에 샘플 데이터를 추가합니다.

샘플 데이터를 자동 생성하려면

샘플 데이터를 자동 생성하면 1~5000행의 데이터를 생성하여 즉시 사용할 수 있습니다. 세분화된 샘플 데이터 형식을 지정하여 설계 및 테스트 요구 사항에 따라 사실적인 데이터를 생성할 수 있습니다. 사실적인 데이터를 생성하는 기능을 활용하려면 데이터 모델러에서 속성에 샘플 데이터 형식을 지정해야 합니다. 샘플 데이터 형식 지정에 대한 자세한 내용은 새 데이터 모델 생성 섹션을 참조하세요.

1. 왼쪽의 탐색 창에서 시각화 도우미 아이콘을 선택합니다.



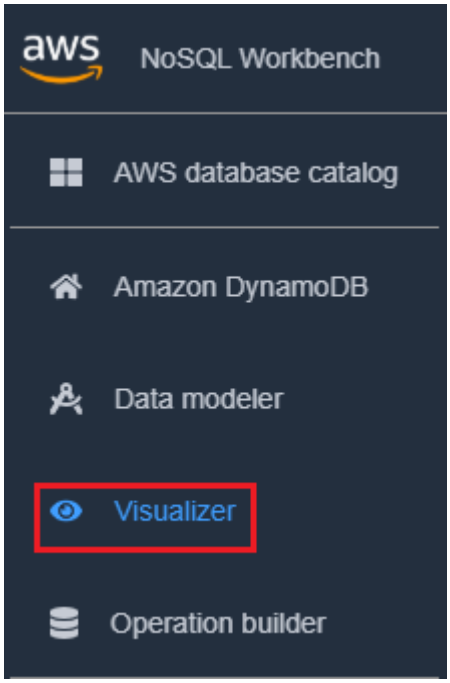

2. 시각화 도우미에서 데이터 모델을 선택한 후 테이블을 선택합니다.
3. 작업 드롭다운을 선택한 후 샘플 데이터 추가를 선택합니다.




4. 생성하려는 샘플 데이터의 수 또는 항목을 입력한 다음 확인을 선택합니다.

샘플 데이터를 한 번에 하나씩 추가하려면

1. 왼쪽의 탐색 창에서 시각화 도우미 아이콘을 선택합니다.



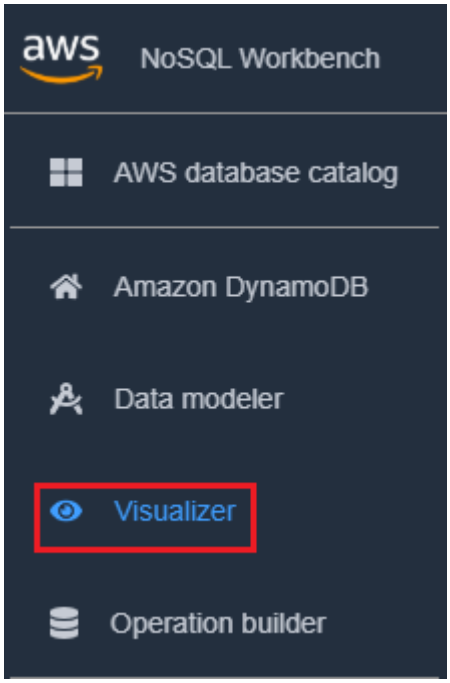

2. 시각화 도우미에서 데이터 모델을 선택한 후 테이블을 선택합니다.
3. 작업 드롭다운을 선택한 후 데이터 편집을 선택합니다.




4. 새 행 추가를 선택합니다. 샘플 데이터를 빈 텍스트 상자에 입력한 다음 새 행 추가를 클릭하여 행을 추가합니다. 작업을 마쳤으면 변경 사항 저장을 선택합니다.

### 샘플 데이터를 삭제하려면

1. 왼쪽의 탐색 창에서 시각화 도우미 아이콘을 선택합니다.




2. 시각화 도우미에서 데이터 모델을 선택한 후 테이블을 선택합니다.
3. 작업 드롭다운을 선택한 후 데이터 편집을 선택합니다.




4. 삭제하려는 각 데이터 행 옆의 삭제 아이콘을 선택합니다.

## CSV 파일에서 샘플 데이터 가져오기

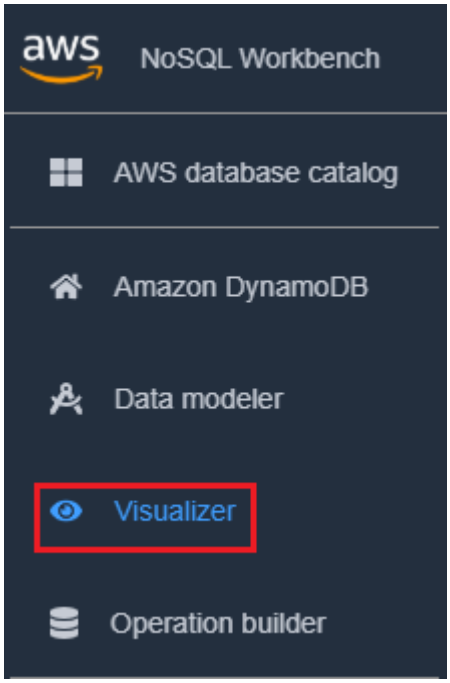
기존 샘플 데이터가 CSV 파일에 있는 경우 이를 NoSQL Workbench로 가져올 수 있습니다. 이렇게 하면 한 줄씩 입력할 필요 없이 샘플 데이터로 모델을 빠르게 채울 수 있습니다.

CSV 파일의 열 이름은 데이터 모델의 속성 이름과 일치해야 하지만 순서가 같을 필요는 없습니다. 예를 들어, 데이터 모델에 `LoginAlias`, `FirstName` 및 `LastName`이라는 속성이 있는 경우CSV 열은 `LastName`, `FirstName` 및 `LoginAlias`일 수 있습니다.

CSV 파일에서 데이터 가져오기는 한 번에 150개의 행으로 제한됩니다.

CSV 파일에서 NoSQL Workbench로 데이터 가져오기

1. 왼쪽의 탐색 창에서 시각화 도우미 아이콘을 선택합니다.




2. 시각화 도우미에서 데이터 모델을 선택한 후 테이블을 선택합니다.
3. 작업 드롭다운을 선택한 후 데이터 편집을 선택합니다.
4. 작업 드롭다운을 다시 선택하고 CSV 파일 가져오기를 선택합니다.
5. CSV 파일을 선택하고 Open(열기)을 클릭합니다. CSV 파일의 데이터가 테이블에 추가됩니다.

> **Note**
>
> CSV 파일에 테이블에 이미 있는 항목과 동일한 키가 있는 행이 하나 이상 포함된 경우, 기존 항목을 덮어쓰거나 테이블 끝에 추가할 수 있습니다. 항목을 추가하도록 선택하면 테이블에 이미 있는 항목과 중복 항목을 구별하기 위해 각 중복 항목의 키에 접미사 '-Copy'가 추가됩니다.

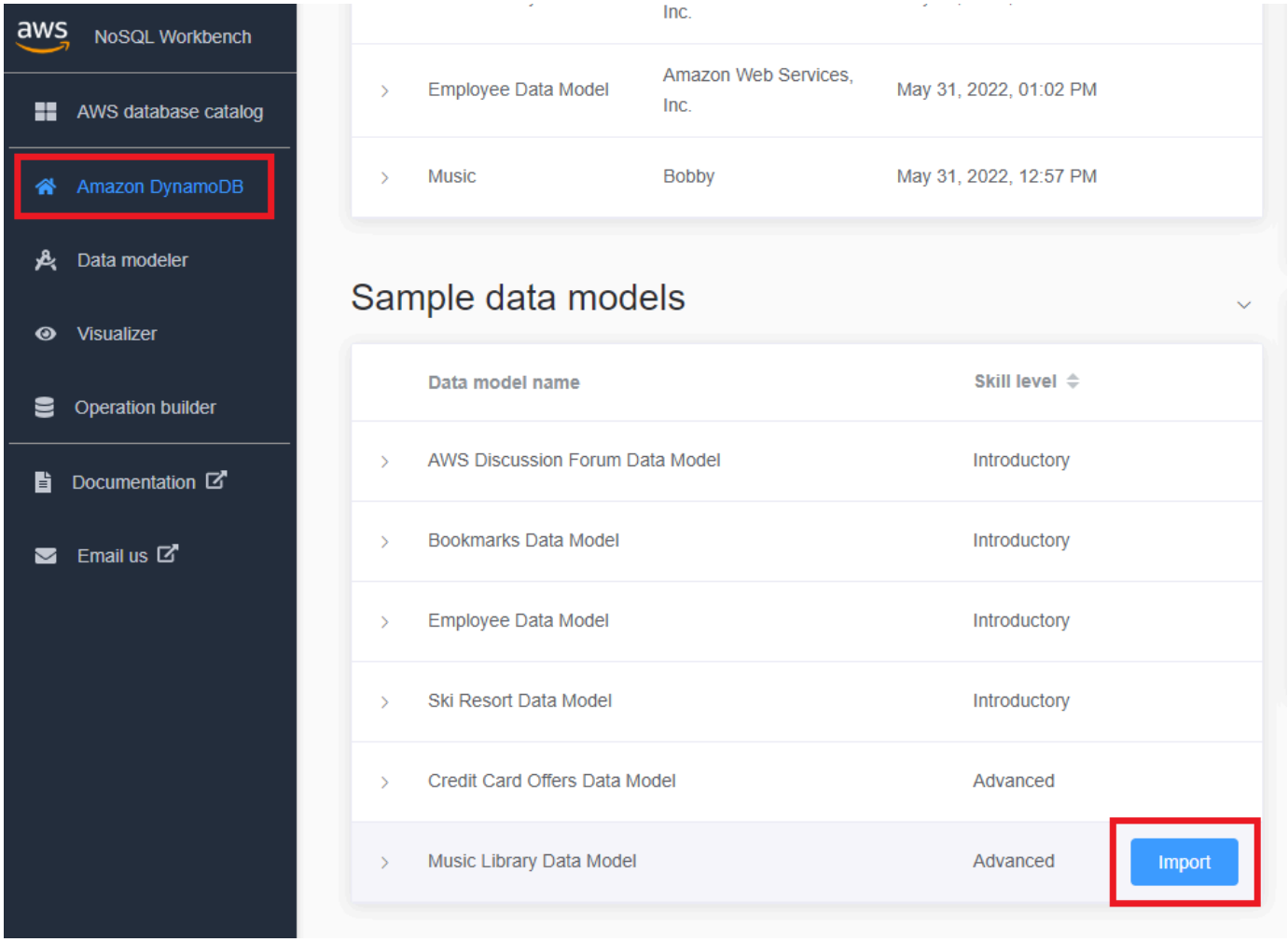
## 데이터 액세스 패턴 보기

NoSQL Workbench에서 패싯은 애플리케이션의 서로 다른 Amazon DynamoDB 데이터 액세스 패턴을 나타냅니다. 여러 데이터 유형이 정렬 키로 표시되는 경우 패싯을 사용하여 데이터 모델을 시각화할 수 있습니다. 패싯을 사용하면 패싯의 제약 조건을 충족하지 않는 레코드를 확인할 필요 없이 테이블에서 데이터의 하위 집합을 볼 수 있습니다. 패싯은 시각적 데이터 모델링 도구로 간주되며 DynamoDB에서 사용 가능한 구성체로 존재하지 않습니다. 패싯은 순전히 액세스 패턴 모델링에 도움이 되기 때문입니다.

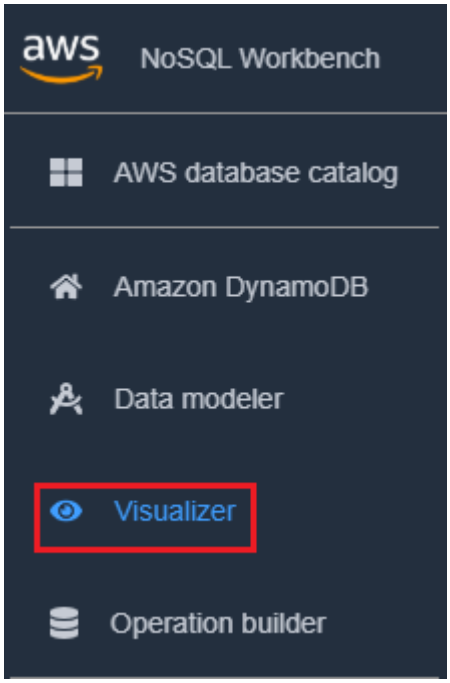
패싯의 예를 보려는 경우 패싯이 포함된 샘플 데이터 모델 중 하나를 데이터 모델 템플릿의 일부로 가져올 수 있습니다.

샘플 데이터 모델 가져오기

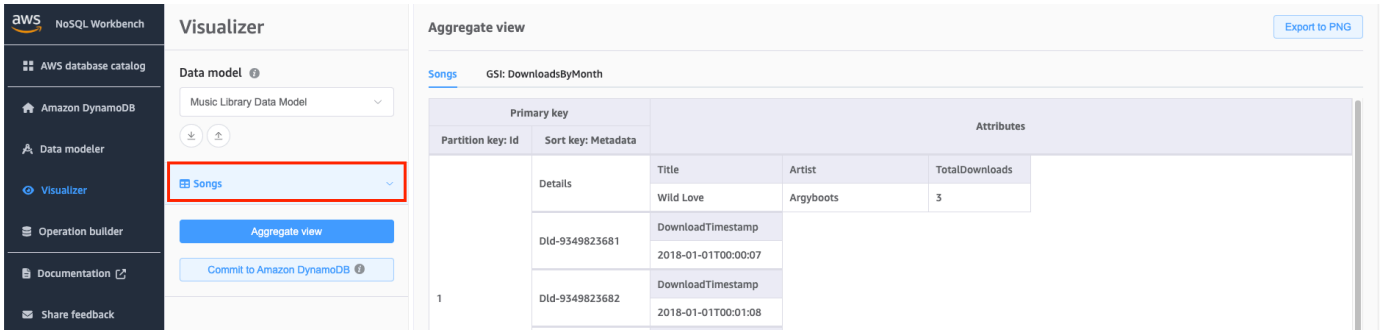
1. 왼쪽에서 Amazon DynamoDB를 선택합니다.
2. Sample data models(샘플 데이터 모델) 섹션에서 Music Library Data Model(음악 라이브러리 데이터 모델) 위에 마우스를 놓고 Import(가져오기)를 선택합니다.



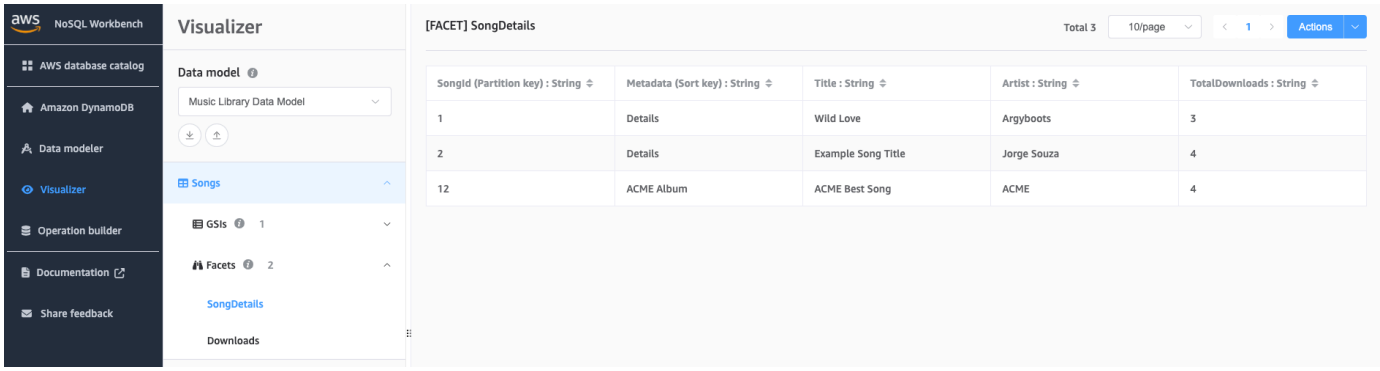

3. 왼쪽의 탐색 창에서 시각화 도우미 아이콘을 선택합니다.




4. 곡(Songs) 테이블을 선택하여 확장합니다. 데이터의 집계 보기가 표시됩니다.




5. Facets(패싯) 드롭다운 화살표를 선택하여 사용 가능한 패싯을 확장합니다.
6. SongDetails 패싯을 적용하여 데이터를 시각화하려면 SongDetails 패싯을 선택합니다.




데이터 모델 제작자를 사용하여 패싯 정의를 편집할 수도 있습니다. 자세한 내용은 기존 데이터 모델 편집 단원을 참조하십시오.

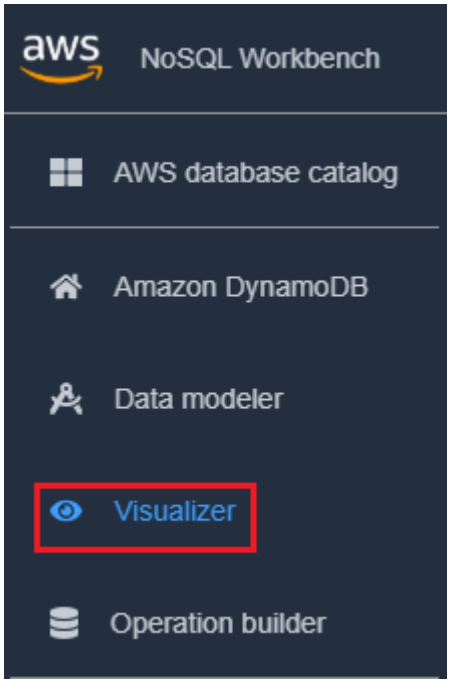
# 집계 보기를 사용하여 데이터 모델에서 모든 테이블 보기

Amazon DynamoDB용 NoSQL Workbench의 집계 보기는 데이터 모델의 모든 테이블을 나타냅니다. 각 테이블마다 다음 정보가 나타납니다.

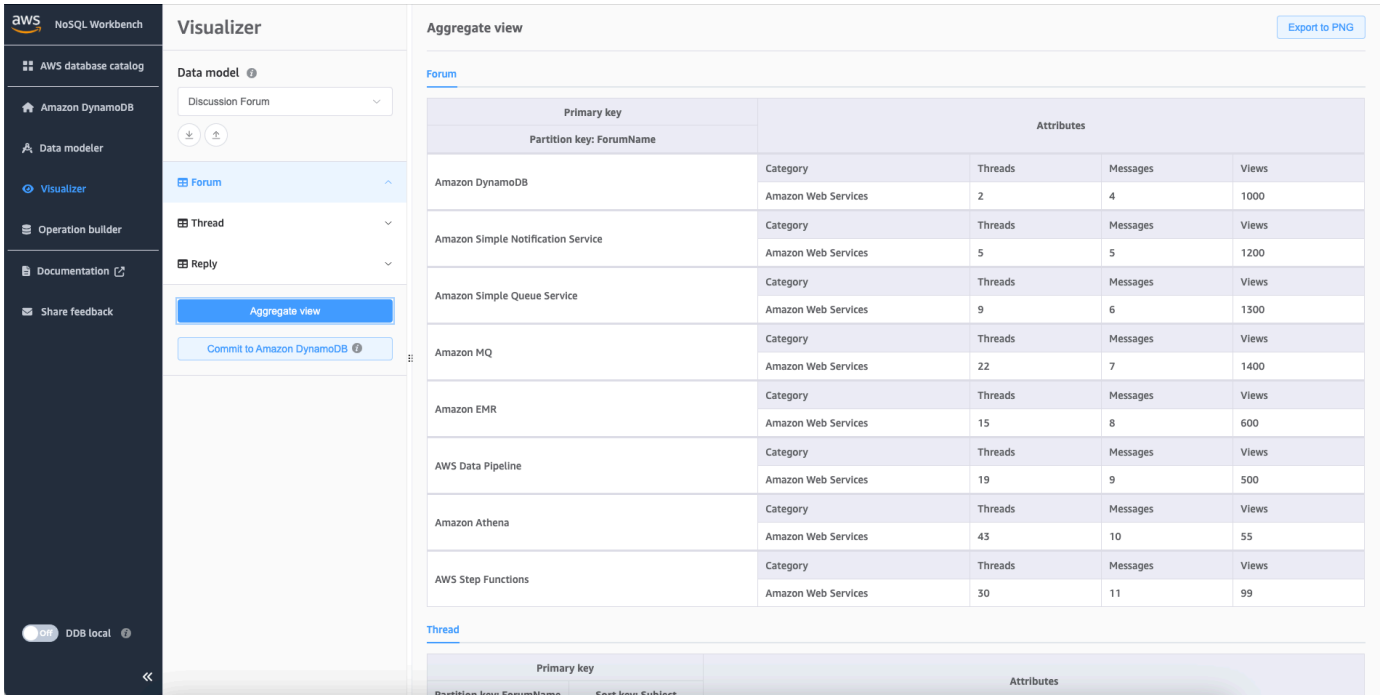
- 테이블 열 이름
- 샘플 데이터
- 테이블과 연결된 모든 전역 보조 인덱스입니다. 각 인덱스마다 다음 정보가 표시됩니다.
  - 인덱스 열 이름
  - 샘플 데이터

## 모든 테이블 정보 보기

1. 왼쪽의 탐색 창에서 시각화 도우미 아이콘을 선택합니다.




2. 시각화 도우미에서 Aggregate view(집계 보기)를 선택합니다.



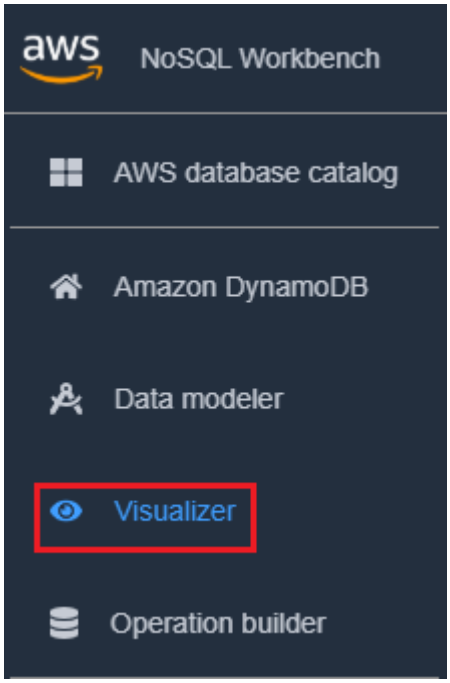

# 데이터 모델을 DynamoDB로 커밋

데이터 모델에 만족하면 모델을 Amazon DynamoDB로 커밋할 수 있습니다.

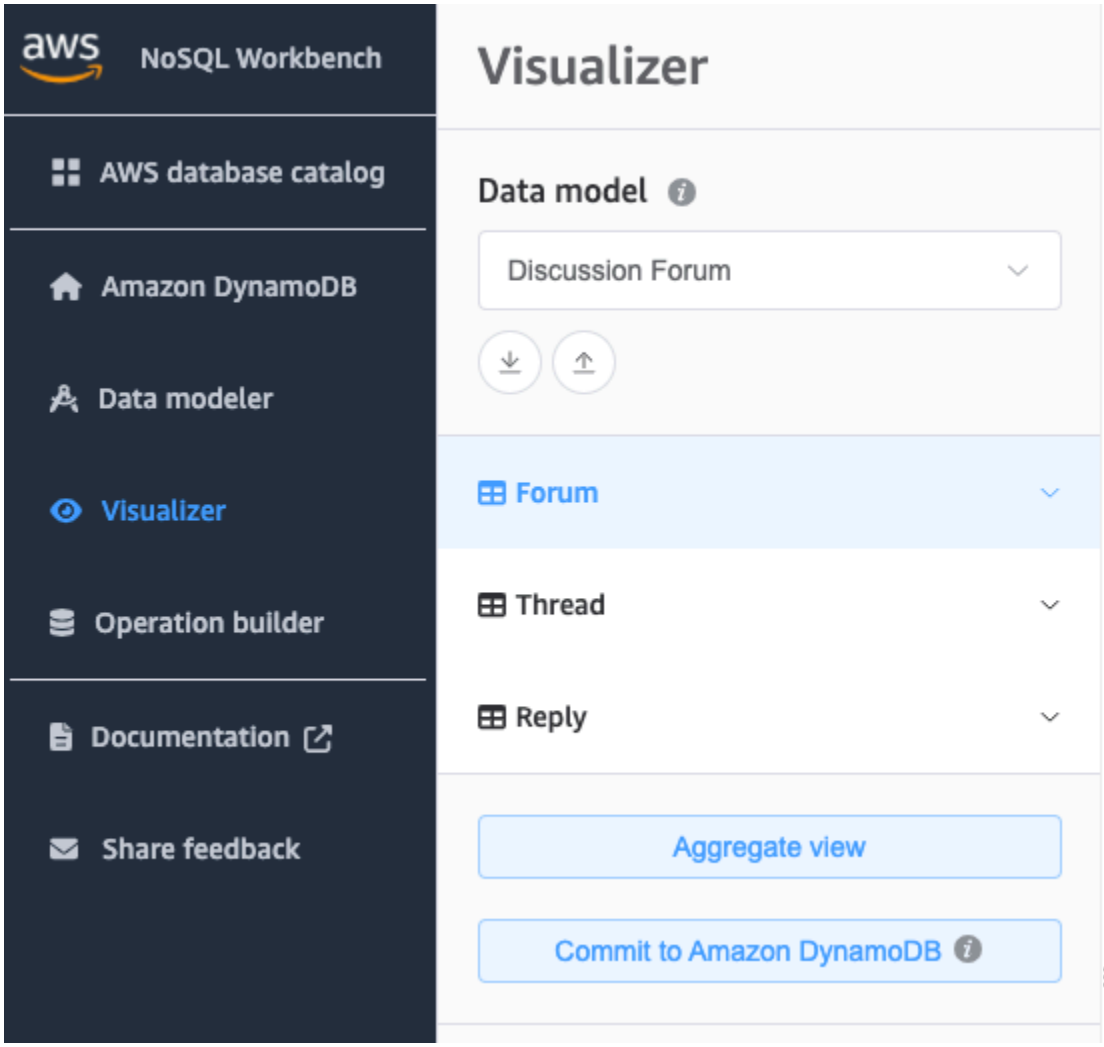
> **Note**
>
> - 이 작업을 하면 데이터 모델에 표시되는 테이블과 글로벌 보조 인덱스의 서버 측 리소스가 AWS에 생성됩니다.
> - 다음과 같은 특성을 가진 테이블이 생성됩니다.
>   - Auto scaling이 70퍼센트의 목표 사용률로 설정됩니다.
>   - 프로비저닝된 용량은 5개 읽기 용량 단위와 5개 쓰기 용량 단위로 설정됩니다.
> - 10개 읽기 용량 단위와 5개 쓰기 용량 단위의 프로비저닝된 용량으로 전역 보조 인덱스가 생성됩니다.

데이터 모델을 DynamoDB로 커밋하기

1. 왼쪽의 탐색 창에서 시각화 도우미 아이콘을 선택합니다.




2. Commit to DynamoDB(DynamoDB로 커밋)를 선택합니다.




3. 기존 연결을 선택하거나 Add new remote connection(새 원격 연결 추가) 탭을 선택하여 새 연결을 생성합니다.

   - 새 연결을 추가하려면 다음 정보를 지정합니다.
     - Account Alias(계정 별칭)
     - AWS 리전
     - 액세스 키 ID
     - 보안 액세스 키

     액세스 키를 얻는 방법에 대한 자세한 내용은 AWS 액세스 키 가져오기를 참조하세요.
   - 필요한 경우 다음을 지정할 수 있습니다.
     - 세션 토큰
     - IAM 역할 ARN

- 프리 티어 계정을 신청하지 않고 DynamoDB Local(다운로드 버전)을 사용하려는 경우:
  1. Add a new DynamoDB local connection(새 DynamoDB Local 연결 추가) 탭을 선택합니다.
  2. 연결 이름과 포트를 지정합니다.

4. 커밋을 선택합니다.

> **Note**
>
> NoSQL Workbench 설정의 일부로 DynamoDB Local을 설치한 경우 NoSQL Workbench 화면 왼쪽 하단에 있는 DynamoDB Local 서버 토글을 사용하여 DynamoDB Local을 켜야 합니다. 이 토글에 대한 자세한 내용은 DynamoDB용 NoSQL Workbench 설치 섹션을 참조하세요.

# NoSQL Workbench로 데이터 세트 탐색 및 작업 빌드

Amazon DynamoDB용 NoSQL Workbench는 쿼리 개발 및 테스트를 위한 풍부한 그래픽 사용자 인터페이스를 제공합니다. NoSQL Workbench에서 작업 빌더를 사용하여 실시간 데이터 세트를 보고, 탐색하고, 쿼리할 수 있습니다. 구조화된 작업 빌더는 프로젝션 표현식, 조건식을 지원하고 여러 언어로 샘플 코드를 생성합니다. 한 Amazon DynamoDB 계정에서 다른 리전의 다른 계정으로 테이블을 직접 복제할 수 있습니다. 또한 DynamoDB 로컬과 Amazon DynamoDB 계정 간에 테이블을 직접 복제하여 개발 환경 간에 테이블의 키 스키마(및 선택적으로 GSI 스키마와 항목)를 더 빠르게 복사할 수 있습니다. 작업 빌더에 최대 50개의 DynamoDB 데이터 작업을 저장할 수 있습니다.

주제

- 실시간 데이터 세트에 연결
- 복잡한 작업 빌드
- NoSQL Workbench로 테이블 복제
- 데이터를 CSV 파일로 내보내기

## 실시간 데이터 세트에 연결

NoSQL Workbench를 사용하여 Amazon DynamoDB 테이블에 연결하려면 먼저 AWS 계정에 연결해야 합니다.

데이터베이스에 연결 추가하기

1. NoSQL Workbench의 왼쪽 탐색 창에서 작업 빌더 아이콘을 선택합니다.
2. 연결 추가를 선택합니다.
3. 다음과 같은 정보를 지정합니다.

   - 계정 별칭
   - AWS 리전
   - 액세스 키 ID
   - 보안 액세스 키

   액세스 키를 얻는 방법에 대한 자세한 내용은 AWS 액세스 키 가져오기를 참조하세요.

   필요한 경우 다음을 지정할 수 있습니다.

   - 세션 토큰
   - IAM 역할 ARN
4. 연결을 선택합니다.

   프리 티어 계정을 신청하지 않고 DynamoDB Local(다운로드 버전)을 사용하려는 경우:

   a. 연결 화면에서 Local(로컬) 탭을 선택합니다.
   b. 다음과 같은 정보를 지정합니다.

      - 연결 이름
      - 포트

   c. 연결 버튼을 선택합니다.

   > **Note**
   >
   > DynamoDB 로컬에 연결하려면 터미널을 사용하여 DynamoDB 로컬을 수동으로 시작하거나(deploying DynamoDB local on your computer 참조) NoSQL Workbench 탐색 메뉴에서 DDB 로컬 토글을 사용하여 DynamoDB 로컬을 직접 시작하세요. 연결 포트가 DynamoDB 로컬 포트와 동일한지 확인하세요.

5. 생성된 연결에서 열기를 선택합니다.

DynamoDB 데이터베이스에 연결하면 사용 가능한 테이블 목록이 왼쪽 창에 나타납니다. 테이블 중 하나를 선택하면 테이블에 저장된 데이터의 샘플이 반환됩니다.

이제 선택한 테이블에 대해 쿼리를 실행할 수 있습니다.

테이블에서 쿼리를 실행하려면 작업 빌드에 대한 다음 단원 및 복잡한 작업 빌드 단원을 참조하세요.

## 복잡한 작업 빌드

Amazon DynamoDB용 NoSQL Workbench의 작업 빌더에는 복잡한 데이터 영역 작업을 수행할 수 있는 시각적 인터페이스가 있습니다. 프로젝션 식과 조건식 지원이 포함되어 있습니다. 작업을 빌드했으면 나중에 사용하기 위해 저장할 수 있습니다(최대 50개의 작업을 저장할 수 있음). 그러면 Saved Operations(저장된 작업) 메뉴를 사용하여 자주 사용하는 데이터 영역 작업 목록을 찾아보고 해당 작업을 사용하여 새 작업을 자동으로 채우고 빌드할 수 있습니다. 이러한 작업의 샘플 코드를 여러 언어로 생성할 수도 있습니다.

NoSQL Workbench는 DynamoDB용 PartiQL 문 빌드를 지원하므로 SQL 호환 쿼리 언어를 사용하여 DynamoDB와 상호 작용할 수 있습니다. 또한 NoSQL Workbench는 DynamoDB CRUD API 작업 빌드도 지원합니다.

NoSQL Workbench를 사용하여 작업을 빌드하려면 왼쪽 탐색 창에서 Operation builder(작업 빌더) 아이콘을 선택합니다.

주제

- PartiQL 문 빌드
- API 작업 빌드

### PartiQL 문 빌드

NoSQL Workbench를 사용하여 DynamoDB용 PartiQL 문을 빌드하려면 NoSQL Workbench UI 상단 근처에 있는 PartiQL 편집기를 선택합니다.

작업 빌더에서 다음과 같은 PartiQL 문 유형을 빌드할 수 있습니다.

주제

- Singleton 문
- 트랜잭션
- 배치

### Singleton 문

PartiQL 문의 코드를 실행하거나 생성하려면 다음을 수행합니다.

1. 창 상단 근처에서 PartiQL 편집기를 선택합니다.
2. 유효한 PartiQL 문을 입력합니다.
3. 문에서 파라미터를 사용하는 경우:
   a. Optional request parameters(선택적 요청 파라미터)를 선택합니다.
   b. Add new parameter(새 파라미터 추가)를 선택합니다.
   c. 속성 유형 및 값을 입력합니다.
   d. 파라미터를 추가하려면 b 및 c단계를 반복합니다.
4. 코드를 생성하려면 Generate code(코드 생성)를 선택합니다.

   표시된 탭에서 원하는 언어를 선택합니다. 이제 이 코드를 복사하여 애플리케이션에서 사용할 수 있습니다.
5. 작업을 즉시 실행하려면 Run(실행)을 선택합니다.
6. 나중에 사용하기 위해 이 작업을 저장하려면 Save operation(저장)을 선택합니다. 그런 다음 작업 이름을 입력하고 Save(저장)를 선택합니다.

### 트랜잭션

PartiQL 트랜잭션의 코드를 실행하거나 생성하려면 다음을 수행합니다.

1. 추가 작업 드롭다운에서 PartiQLTransaction을 선택합니다.
2. Add a new statement(새 문 추가)를 선택합니다.
3. 유효한 PartiQL 문을 입력합니다.

> **Note**
>
> 동일한 PartiQL 트랜잭션 요청에서 읽기 및 쓰기 작업이 지원되지 않습니다. 동일한 요청에서 INSERT, UPDATE 및 DELETE 문과 함께 SELECT 문을 사용할 수 없습니다. 자세한 내용은 DynamoDB용 PartiQL에서 트랜잭션 수행을 참조하세요.

4. 문에서 파라미터를 사용하는 경우
   a. Optional request parameters(선택적 요청 파라미터)를 선택합니다.

b. Add new parameter(새 파라미터 추가)를 선택합니다.

c. 속성 유형 및 값을 입력합니다.

d. 파라미터를 추가하려면 b 및 c단계를 반복합니다.

5. 문을 추가하려면 2~4단계를 반복합니다.

6. 코드를 생성하려면 Generate code(코드 생성)를 선택합니다.

   표시된 탭에서 원하는 언어를 선택합니다. 이제 이 코드를 복사하여 애플리케이션에서 사용할 수 있습니다.

7. 작업을 즉시 실행하려면 Run(실행)을 선택합니다.

8. 나중에 사용하기 위해 이 작업을 저장하려면 Save operation(저장)을 선택합니다. 그런 다음 작업 이름을 입력하고 Save(저장)를 선택합니다.

배치

PartiQL 배치의 코드를 실행하거나 생성하려면 다음을 수행합니다.

1. 추가 작업 드롭다운에서 PartiQLBatch를 선택합니다.

2. Add a new statement(새 문 추가)를 선택합니다.

3. 유효한 PartiQL 문을 입력합니다.

> **Note**
>
> 동일한 PartiQL 배치 요청에서 읽기 및 쓰기 작업이 지원되지 않습니다. 즉, 동일한 요청에서 INSERT, UPDATE 및 DELETE 문과 함께 SELECT 문을 사용할 수 없습니다. 동일한 항목에 쓰기 작업은 허용되지 않습니다. BatchGetItem 작업에서처럼 singleton 읽기 작업만 지원됩니다. 검색 및 쿼리 작업은 지원되지 않습니다. 자세한 내용은 DynamoDB용 PartiQL에서 일괄 작업 실행을 참조하세요.

4. 문에서 파라미터를 사용하는 경우:

   a. Optional request parameters(선택적 요청 파라미터)를 선택합니다.

   b. Add new parameter(새 파라미터 추가)를 선택합니다.

   c. 속성 유형 및 값을 입력합니다.

   d. 파라미터를 추가하려면 b 및 c단계를 반복합니다.

5. 문을 추가하려면 2~4단계를 반복합니다.

6. 코드를 생성하려면 Generate code(코드 생성)를 선택합니다.

   표시된 탭에서 원하는 언어를 선택합니다. 이제 이 코드를 복사하여 애플리케이션에서 사용할 수 있습니다.

7. 작업을 즉시 실행하려면 Run(실행)을 선택합니다.
8. 나중에 사용하기 위해 이 작업을 저장하려면 Save operation(저장)을 선택합니다. 그런 다음 작업 이름을 입력하고 Save(저장)를 선택합니다.

## API 작업 빌드

NoSQL Workbench를 사용하여 DynamoDB CRUD API를 빌드하려면 NoSQL Workbench 사용자 인터페이스의 왼쪽에서 작업 빌더를 선택합니다.

그런 다음, 열기를 선택하고 연결을 선택합니다.

작업 빌더에서 다음 작업을 수행할 수 있습니다.

- 테이블 삭제
- 테이블 생성
- 테이블 업데이트

- 항목 Put
- 항목 업데이트
- 항목 삭제
- Query
- Scan
- 트랜잭션 항목 가져오기
- 트랜잭션 항목 쓰기

### 테이블 삭제

`Delete Table` 작업을 실행하려면 다음을 수행합니다.

1. 테이블 섹션에서 삭제하려는 테이블을 찾습니다.
2. 가로 줄임표 메뉴에서 테이블 삭제를 선택합니다.

3. 테이블 이름을 입력하여 테이블을 삭제하려 한다는 것을 확인합니다.
4. 삭제를 선택합니다.

이 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조에서 테이블 삭제를 참조하세요.

GSI 삭제

`Delete GSI` 작업을 실행하려면 다음을 수행합니다.

1. 테이블 섹션에서 삭제하려는 테이블의 GSI를 찾습니다.
2. 가로 줄임표 메뉴에서 GSI 삭제를 선택합니다.
3. GSI 이름을 입력하여 GSI를 삭제하려 한다는 것을 확인합니다.
4. 삭제를 선택합니다.

이 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조에서 테이블 삭제를 참조하세요.

테이블 생성

`Create Table` 작업을 실행하려면 다음을 수행합니다.

1. 테이블 섹션 옆에 있는 + 아이콘을 선택합니다.
2. 원하는 테이블 이름을 입력합니다.
3. 파티션 키를 생성합니다.
4. 선택 사항: 정렬 키를 생성합니다.
5. 용량 설정을 사용자 지정하려면 기본 용량 설정 사용 옆의 확인란을 선택 취소합니다.
   - 이제 프로비저닝됨(Provisioned) 또는 온디맨드(On-demand) 용량을 선택할 수 있습니다.

     프로비저닝된 용량을 선택한 경우 최소 및 최대 읽기와 쓰기 용량 단위를 설정할 수 있습니다. 또한 오토 스케일링을 사용 설정하거나 사용 중지할 수 있습니다.
   - 테이블이 현재 온디맨드로 설정되어 있으면 프로비저닝된 처리량을 지정할 수 없습니다.
   - 온디맨드에서 프로비저닝된 처리량으로 전환하면 Auto Scaling이 최소 1, 최대 10, 목표 70%로 모든 GSI에 자동으로 적용됩니다.
6. GSI 없이 이 테이블을 생성하려면 GSI 건너뛰기 및 생성을 선택합니다. 필요에 따라 다음을 선택하여 이 새 테이블로 GSI를 생성할 수 있습니다.

이 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조에서 테이블 생성을 참조하세요.

### GSI 생성

`Create GSI` 작업을 실행하려면 다음을 수행합니다.

1. GSI를 추가하려는 테이블을 찾습니다.
2. 가로 줄임표 메뉴에서 GSI 생성을 선택합니다.
3. 인덱스 이름 아래에 GSI 이름을 지정합니다.
4. 파티션 키를 생성합니다.
5. 선택 사항: 정렬 키를 생성합니다.
6. 드롭다운에서 프로젝션 유형 옵션을 선택합니다.
7. GSI 생성을 선택합니다.

이 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조에서 테이블 생성을 참조하세요.

### 테이블 업데이트

`Update Table` 작업으로 테이블의 용량 설정을 업데이트하려면 다음을 수행합니다.

1. 용량 설정을 업데이트하려는 테이블을 찾습니다.
2. 가로 줄임표 메뉴에서 용량 설정 업데이트를 선택합니다.
3. 프로비저닝된 용량 또는 온디맨드 용량을 선택합니다.

   프로비저닝된 용량을 선택한 경우 최소 및 최대 읽기와 쓰기 용량 단위를 설정할 수 있습니다. 또한 오토 스케일링을 사용 설정하거나 사용 중지할 수 있습니다.
4. 업데이트를 선택합니다.

이 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조에서 테이블 업데이트를 참조하세요.

### GSI 업데이트

`Update Table` 작업으로 GSI의 용량 설정을 업데이트하려면 다음을 수행합니다.

> **Note**
>
> 기본적으로 글로벌 보조 인덱스는 기본 테이블의 용량 설정을 상속받습니다. 글로벌 보조 인덱스는 기본 테이블이 프로비저닝된 용량 모드인 경우에만 다른 용량 모드를 가질 수 있습니다.

> 프로비저닝된 모드 테이블에서 글로벌 보조 인덱스를 생성할 때는 해당 인덱스에 예상되는 워크로드에 대한 읽기 및 쓰기 용량 단위를 지정해야 합니다. 자세한 내용은 글로벌 보조 인덱스에 대한 프로비저닝된 처리량 고려 사항 단원을 참조하십시오.

1. 용량 설정을 업데이트하려는 GSI를 찾습니다.
2. 가로 줄임표 메뉴에서 용량 설정 업데이트를 선택합니다.
3. 이제 프로비저닝됨(Provisioned) 또는 온디맨드(On-demand) 용량을 선택할 수 있습니다.

   프로비저닝된 용량을 선택한 경우 최소 및 최대 읽기와 쓰기 용량 단위를 설정할 수 있습니다. 또한 오토 스케일링을 사용 설정하거나 사용 중지할 수 있습니다.
4. 업데이트를 선택합니다.

이 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조에서 테이블 업데이트를 참조하세요.

항목 추가

`Put Item` 작업을 사용하여 항목을 생성합니다. `Put Item` 작업의 코드를 실행하거나 생성하려면 다음을 수행합니다.

1. 항목을 생성하려는 테이블을 찾습니다.
2. 작업 드롭다운에서 항목 생성을 선택합니다.
3. 파티션 키 값을 입력합니다.
4. 있는 경우 정렬 키 값을 입력합니다.
5. 비-키 속성을 추가하려면 다음을 수행합니다.

   a. + 다른 속성 추가를 선택합니다.

   b. 속성 이름, 형식 및 값을 지정합니다.
6. `Put Item` 작업이 성공할 수 있는 조건식이 충족되어야 할 경우에는 다음을 수행합니다.

   a. 조건을 선택합니다.

   b. 속성 이름, 비교 연산자, 속성 형식 및 속성 값을 지정합니다.

   c. 다른 조건이 필요할 경우에는 조건을 다시 선택합니다.

   자세한 내용은 DynamoDB 조건 표현식 CLI 예제 단원을 참조하십시오.

7. 코드를 생성하려면 Generate code(코드 생성)를 선택합니다.

   표시된 탭에서 원하는 언어를 선택합니다. 이제 이 코드를 복사하여 애플리케이션에서 사용할 수 있습니다.
8. 작업을 즉시 실행하려면 Run(실행)을 선택합니다.
9. 나중에 사용하기 위해 이 작업을 저장하려면 Save operation(작업 저장)을 선택한 다음 작업 이름을 입력하고 Save(저장)를 선택합니다.

이 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조에서 PutItem을 참조하세요.

항목 업데이트

`Update Item` 작업의 코드를 실행하거나 생성하려면 다음을 수행합니다.

1. 항목을 업데이트하려는 테이블을 찾습니다.
2. 항목을 선택합니다.
3. 선택한 표현식의 속성 이름과 속성 값을 입력합니다.
4. 표현식을 더 추가하려면 표현식 업데이트 드롭다운 목록에서 다른 표현식을 선택한 다음, + 아이콘을 선택합니다.
5. `Update Item` 작업이 성공할 수 있는 조건식이 충족되어야 할 경우에는 다음을 수행합니다.

   a. 조건을 선택합니다.

   b. 속성 이름, 비교 연산자, 속성 형식 및 속성 값을 지정합니다.

   c. 다른 조건이 필요할 경우에는 조건을 다시 선택합니다.

   자세한 내용은 DynamoDB 조건 표현식 CLI 예제 단원을 참조하십시오.
6. 코드를 생성하려면 Generate code(코드 생성)를 선택합니다.

   사용하려는 언어의 탭을 선택합니다. 이제 이 코드를 복사하여 애플리케이션에서 사용할 수 있습니다.
7. 작업을 즉시 실행하려면 Run(실행)을 선택합니다.
8. 나중에 사용하기 위해 이 작업을 저장하려면 Save operation(작업 저장)을 선택한 다음 작업 이름을 입력하고 Save(저장)를 선택합니다.

이 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조에서 UpdateItem을 참조하세요.

### 항목 삭제

`Delete Item` 작업을 실행하려면 다음을 수행합니다.

1. 항목을 삭제하려는 테이블을 찾습니다.
2. 항목을 선택합니다.
3. 작업 드롭다운에서 항목 삭제를 선택합니다.
4. 삭제를 선택하여 항목을 삭제하려 한다는 것을 확인합니다.

이 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조에서 DeleteItem을 참조하세요.

### 항목 복제

속성이 동일한 새 항목을 생성하여 항목을 복제할 수 있습니다. 항목을 복제하려면 다음을 수행합니다.

1. 항목을 복제하려는 테이블을 찾습니다.
2. 항목을 선택합니다.
3. 작업 드롭다운에서 항목 복제를 선택합니다.
4. 새 파티션 키를 지정합니다.
5. 새 정렬 키를 지정합니다(필요한 경우).
6. 실행을 선택합니다.

이 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조에서 DeleteItem을 참조하세요.

### Query

`Query` 작업의 코드를 실행하거나 생성하려면 다음을 수행합니다.

1. NoSQL Workbench UI 상단에서 쿼리를 선택합니다.
2. 파티션 키 값을 지정합니다.
3. `Query` 작업에 정렬 키가 필요한 경우
   a. 정렬 키를 선택합니다.
   b. 비교 연산자 및 속성 값을 지정합니다.
4. 쿼리를 선택하여 이 쿼리 작업을 실행합니다. 옵션이 더 필요한 경우 추가 옵션 확인란을 선택하고 다음 단계를 계속 진행합니다.

5. 작업 결과로 일부 속성만 반환할 경우에는 Projection expression(프로젝션 식)을 선택합니다.
6. + 아이콘을 선택합니다.
7. 커리 결과로 반환될 속성을 입력합니다.
8. 속성이 더 필요한 경우에는 +를 선택합니다.
9. `Query` 작업이 성공할 수 있는 조건식이 충족되어야 할 경우에는 다음을 수행합니다.

    a. 조건을 선택합니다.

    b. 속성 이름, 비교 연산자, 속성 형식 및 속성 값을 지정합니다.

    c. 다른 조건이 필요할 경우에는 조건을 다시 선택합니다.

    자세한 내용은 DynamoDB 조건 표현식 CLI 예제 단원을 참조하십시오.
10. 코드를 생성하려면 Generate code(코드 생성)를 선택합니다.

    사용하려는 언어의 탭을 선택합니다. 이제 이 코드를 복사하여 애플리케이션에서 사용할 수 있습니다.
11. 작업을 즉시 실행하려면 Run(실행)을 선택합니다.
12. 나중에 사용하기 위해 이 작업을 저장하려면 Save operation(작업 저장)을 선택한 다음 작업 이름을 입력하고 Save(저장)를 선택합니다.

이 작업에 대한 자세한 내용은 Amazon DynamoDB API 참조에서 Query를 참조하세요.

스캔

`Scan` 작업의 코드를 실행하거나 생성하려면 다음을 수행합니다.

1. NoSQL Workbench UI 상단에서 스캔을 선택합니다.
2. 스캔 버튼을 선택하여 이 기본 스캔 작업을 수행합니다. 옵션이 더 필요한 경우 추가 옵션 확인란을 선택하고 다음 단계를 계속 진행합니다.
3. 스캔 결과를 필터링하려면 속성 이름을 지정합니다.
4. 작업 결과로 일부 속성만 반환할 경우에는 Projection expression(프로젝션 식)을 선택합니다.
5. 스캔 작업이 성공할 수 있는 조건식이 충족되어야 할 경우에는 다음을 수행합니다.

    a. 조건을 선택합니다.

    b. 속성 이름, 비교 연산자, 속성 형식 및 속성 값을 지정합니다.

c. 다른 조건이 필요할 경우에는 조건을 다시 선택합니다.

자세한 내용은 DynamoDB 조건 표현식 CLI 예제 단원을 참조하십시오.

6. 코드를 생성하려면 Generate code(코드 생성)를 선택합니다.

사용하려는 언어의 탭을 선택합니다. 이제 이 코드를 복사하여 애플리케이션에서 사용할 수 있습니다.

7. 작업을 즉시 실행하려면 Run(실행)을 선택합니다.
8. 나중에 사용하기 위해 이 작업을 저장하려면 Save operation(작업 저장)을 선택한 다음 작업 이름을 입력하고 Save(저장)를 선택합니다.

TransactGetItems

`TransactGetItems` 작업의 코드를 실행하거나 생성하려면 다음을 수행합니다.

1. NoSQL Workbench UI 상단의 추가 작업 드롭다운에서 TransactGetItems를 선택합니다.
2. TransactGetItem 근처의 + 아이콘을 선택합니다.
3. 파티션 키를 지정합니다.
4. 정렬 키를 지정합니다(필요한 경우).
5. 작업을 수행하려면 실행을 선택하고, 저장하려면 작업 저장을 선택하며, 해당 코드를 생성하려면 코드 생성을 선택합니다.

트랜잭션에 대한 자세한 내용은 Amazon DynamoDB Transactions를 참조하세요.

TransactWriteItems

`TransactWriteItems` 작업의 코드를 실행하거나 생성하려면 다음을 수행합니다.

1. NoSQL Workbench UI 상단의 추가 작업 드롭다운에서 TransactWriteItems를 선택합니다.
2. 작업 드롭다운에서 작업을 선택합니다.
3. TransactWriteItem 근처의 + 아이콘을 선택합니다.
4. 작업 드롭다운에서 수행하려는 작업을 선택합니다.

- `DeleteItem`은 항목 삭제 작업 지침을 따릅니다.
- `PutItem`은 항목 추가 작업 지침을 따릅니다.

- `UpdateItem`은 항목 업데이트 작업 지침을 따릅니다.

작업 순서를 변경하려면 좌측 목록에서 작업을 선택한 다음 위쪽 또는 아래쪽 화살표를 선택하여 목록에서 위로 또는 아래로 옮깁니다.

작업을 삭제하려면 목록에서 작업을 선택한 다음 삭제(휴지통) 아이콘을 선택합니다.

5. 작업을 수행하려면 실행을 선택하고, 저장하려면 작업 저장을 선택하며, 해당 코드를 생성하려면 코드 생성을 선택합니다.

트랜잭션에 대한 자세한 내용은 Amazon DynamoDB Transactions를 참조하세요.

## NoSQL Workbench로 테이블 복제

테이블을 복제하면 개발 환경 간에 테이블의 키 스키마(및 선택적으로 GSI 스키마와 항목)가 복사됩니다. DynamoDB 로컬에서 Amazon DynamoDB 계정으로 테이블을 복제할 수 있으며, 더 빠른 실험을 위해 한 계정에서 다른 리전의 계정으로 테이블을 복제할 수도 있습니다.

테이블 복제하는 방법

1. 작업 빌더에서 연결 및 리전을 선택합니다(DynamoDB 로컬의 경우 리전 선택이 가능하지 않음).
2. DynamoDB에 연결되면 테이블을 검색하고 복제하려는 테이블을 선택합니다.
3. 가로 줄임표 메뉴에서 복제 옵션을 선택합니다.
4. 복제 대상 세부 정보를 입력합니다.

    a. 연결을 선택합니다.

    b. 리전을 선택합니다(DynamoDB 로컬에서는 리전을 사용할 수 없음).

    c. 새 테이블 이름을 입력합니다.

    d. 다음과 같이 복제 옵션을 선택합니다.

        i. 키 스키마는 기본적으로 선택되며 선택을 취소할 수 없습니다. 기본적으로 테이블을 복제하면 프라이머리 키와 정렬 키가 복사됩니다(사용 가능한 경우).

        ii. 복제할 테이블에 GSI가 있는 경우 GSI 스키마가 기본적으로 선택됩니다. 테이블을 복제하면 GSI 프라이머리 키와 정렬 키가 복사됩니다(사용 가능한 경우). GSI 스키마의 선택을 취소하여 GSI 스키마 복제를 건너뛸 수 있습니다. 테이블을 복제하면 기본 테이블의 용량 설정이 GSI의 용량 설정으로 복사됩니다. 복제가 완료된 후 작업 빌더의 `UpdateTable` 작업을 사용하여 테이블의 GSI 용량 설정을 업데이트할 수 있습니다.

5. 복제할 항목 수를 입력합니다. 키 스키마와 선택적으로 GSI 스키마만 복제하려면 복제할 항목 값을 0으로 두면 됩니다. 복제할 수 있는 최대 항목 수는 5,000개입니다.
6. 용량 모드를 선택합니다.
   a. 온디맨드 모드가 기본적으로 선택됩니다. DynamoDB on-demand는 읽기 및 쓰기 요청에 대해 요청당 지불 가격을 제공하므로 사용하는 만큼에 대해서만 비용을 지불하면 됩니다. 자세히 알아보려면 DynamoDB On-demand mode 모드를 참조하세요.
   b. 프로비저닝된 모드를 사용하면 애플리케이션에 필요한 초당 읽기 및 쓰기 횟수를 지정할 수 있습니다. Auto Scaling을 사용하여 트래픽 변경에 따라 테이블의 프로비저닝된 용량을 자동으로 조정할 수 있습니다. 자세히 알아보려면 DynamoDB Provisioned mode를 참조하세요.
7. 복제를 선택하여 복제를 시작합니다.
8. 복제 프로세스는 백그라운드에서 실행됩니다. 복제 테이블 상태가 변경될 경우 작업 빌더 탭에 알림이 표시됩니다. 작업 빌더 탭을 선택한 다음 화살표 버튼을 선택하여 이 상태에 액세스할 수 있습니다. 화살표 버튼은 메뉴 사이드바 하단 근처의 테이블 복제 상태 위젯에 있습니다.

## 데이터를 CSV 파일로 내보내기

쿼리 결과를 작업 빌더에서 CSV 파일로 내보낼 수 있습니다. 이렇게 하면 스프레드시트에 데이터를 로드하거나 원하는 프로그래밍 언어를 사용하여 처리할 수 있습니다.

CSV로 내보내기

1. 작업 빌더에서 스캔 또는 쿼리 등의 원하는 작업을 실행합니다.

   > **Note**
   >
   > - 읽기 API 작업 및 PartiQL 문의 결과만 CSV 파일로 내보낼 수 있습니다. 트랜잭션 읽기 문의 결과는 내보낼 수 없습니다.
   > - 현재는 결과를 한 번에 한 페이지씩 CSV 파일로 내보낼 수 있습니다. 결과 페이지가 여러 개인 경우 각 페이지를 개별적으로 내보내야 합니다.

2. 결과에서 내보내려는 항목을 선택합니다.
3. 작업 드롭다운에서 CSV로 내보내기를 선택합니다.
4. CSV 파일의 파일 이름과 위치를 선택한 다음 저장(Save)을 선택합니다.

# NoSQL Workbench용 샘플 데이터 모델

모델 제작자 및 시각화 도우미 홈 페이지에는 NoSQL Workbench와 함께 제공되는 여러 샘플 모델이 나와 있습니다. 이 섹션에서는 이러한 모델 및 사용 가능한 용도에 대해 설명합니다.

주제


- 직원 데이터 모델
- 토론 포럼 데이터 모델
- 음악 라이브러리 데이터 모델
- 스키장 데이터 모델
- 신용카드 제안 데이터 모델
- 북마크 데이터 모델


## 직원 데이터 모델

이 데이터 모델은 소개 모델입니다. 고유한 별칭, 이름, 성, 직책, 상사 및 역량 같은 직원의 기본 세부 정보를 나타냅니다.

이 데이터 모델에서는 두 가지 이상의 역량 보유와 같은 복잡한 속성 처리 등의 몇 가지 기법을 보여줍니다. 또한 이 모델은 보조 인덱스인 DirectReports를 사용하여 얻은 상사 및 부하 직원들을 통한 일대다 관계의 예입니다.

이 데이터 모델에서 제공되는 액세스 패턴은 다음과 같습니다.

- `Employee` 테이블에서 제공되는 직원의 로그인 별칭을 사용하여 직원 레코드 검색
- Employee 테이블의 글로벌 보조 인덱스인 `Name`에서 제공되는 이름별로 직원 검색
- Employee 테이블의 글로벌 보조 인덱스인 `DirectReports`에서 제공되는 상사의 로그인 별칭을 사용하여 상사의 모든 부하 직원 검색

## 토론 포럼 데이터 모델

이 데이터 모델은 토론 포럼을 나타냅니다. 이 모델을 사용하여 고객은 개발자 커뮤니티에 참여하고, 질문하고, 다른 고객의 게시물에 댓글을 달 수 있습니다. 각 AWS 서비스에는 전용 포럼이 있습니다. 누구라도 포럼에서 메시지를 게시하여 새 토론 스레드를 시작할 수 있으며 각 스레드는 여러 개의 회신을 받습니다.

이 데이터 모델에서 제공되는 액세스 패턴은 다음과 같습니다.

- Forum 테이블에서 제공되는 포럼의 이름을 사용하여 포럼 레코드 검색
- Thread 테이블에서 제공되는 포럼에 대한 특정 스레드 또는 모든 스레드 검색
- Reply 테이블의 글로벌 보조 인덱스인 PostedBy-Message-Index에서 제공되는 게시 사용자의 이메일 주소를 사용하여 회신 검색

## 음악 라이브러리 데이터 모델

이 데이터 모델은 광범위한 노래 모음이 있는 음악 라이브러리를 나타내며 가장 많이 다운로드된 노래를 거의 실시간으로 보여줍니다.

이 데이터 모델에서 제공되는 액세스 패턴은 다음과 같습니다.

- Songs 테이블에서 제공되는 노래 레코드 검색
- Songs 테이블에서 제공되는 노래에 대한 특정 다운로드 레코드 또는 모든 다운로드 레코드 검색
- Song 테이블에서 제공되는 노래에 대한 특정 월별 다운로드 횟수 레코드 또는 모든 월별 다운로드 횟수 레코드 검색
- Songs 테이블에서 제공되는 노래에 대한 모든 레코드(노래 레코드, 다운로드 레코드 및 월별 다운로드 횟수 레코드 포함) 검색
- Songs 테이블의 글로벌 보조 인덱스인 DownloadsByMonth에서 제공되는 가장 많이 다운로드된 노래 검색

## 스키장 데이터 모델

이 데이터 모델은 각 스키 리프트에서 매일 수집된 광범위한 데이터 모음이 있는 스키장을 나타냅니다.

이 데이터 모델에서 제공되는 액세스 패턴은 다음과 같습니다.

- SkiLifts 테이블에서 제공되는 주어진 스키 리프트 또는 스키장 전체에 대한 모든 데이터(동적 및 정적 데이터) 검색
- SkiLifts 테이블에서 제공되는 특정 날짜에서의 스키 리프트 또는 스키장 전체에 대한 모든 동적 데이터(고유 리프트 탑승자, 적설량, 눈사태 위험 및 리프트 상태 포함) 검색
- SkiLifts 테이블에서 제공되는 특정 스키 리프트에 대한 모든 정적 데이터(리프트가 숙련된 탑승자 전용인 경우 수직 피트, 리프트 상승 및 리프트 탑승 시간 포함) 검색

- SkiLifts 테이블의 글로벌 보조 인덱스인 SkiLiftsByRiders에서 제공되는 총 고유 탑승자별로 정렬된 특정 스키 리프트 또는 스키장 전체에 대해 기록된 데이터의 날짜 검색

## 신용카드 제안 데이터 모델

이 데이터 모델은 신용 카드 제안 애플리케이션에서 사용됩니다.

신용 카드 공급자는 시간이 지남에 따라 제안을 생성합니다. 이러한 제안에는 수수료 없이 잔액 이체, 신용 한도 증가, 낮은 금리, 캐시백 및 항공사 마일리지가 포함됩니다. 고객이 이러한 제안을 수락하거나 거부한 후에 각각의 제안 상태가 그에 따라 업데이트됩니다.

이 데이터 모델에서 제공되는 액세스 패턴은 다음과 같습니다.

- 기본 테이블에서 제공되는 AccountId를 사용하여 계정 레코드 검색
- 보조 인덱스인 AccountIndex에서 제공되는 몇 가지 예상된 항목이 있는 모든 계정 검색
- 기본 테이블에서 제공되는 AccountId를 사용하여 계정 및 해당 계정과 연관된 모든 제안 레코드 검색
- 기본 테이블에서 제공되는 AccountId 및 OfferId를 사용하여 계정 및 해당 계정과 연관된 특정 제안 레코드 검색
- 보조 인덱스인 GSI1에서 제공되는 AccountId, OfferType 및 Status를 사용하여 계정과 연관된 특정 OfferType의 모든 ACCEPTED/DECLINED 제안 레코드 검색
- 기본 테이블에서 제공되는 OfferId를 사용하여 제안 및 연관된 제안 항목 레코드 검색

## 북마크 데이터 모델

이 데이터 모델은 고객의 북마크를 저장하는 데 사용됩니다.

한 고객이 여러 북마크를 보유할 수 있으며 하나의 북마크에 여러 고객이 소속될 수 있습니다. 이 데이터 모델은 다대다 관계를 나타냅니다.

이 데이터 모델에서 제공되는 액세스 패턴은 다음과 같습니다.

- 이제 customerId별 단일 쿼리를 통해 북마크뿐만 아니라 고객 데이터도 반환할 수 있습니다.
- 쿼리 ByEmail 인덱스는 이메일 주소별로 고객 데이터를 반환합니다. 북마크는 이 인덱스에 의해 검색되지 않습니다.
- 쿼리 ByUrl 인덱스는 URL별로 북마크 데이터를 가져옵니다. 동일한 URL을 여러 고객이 북마크할 수 있기 때문에 인덱스용 정렬 키로서 customerId를 제공하고 있습니다.

- 쿼리 `ByCustomerFolder` 인덱스는 각 고객에 대한 폴더별로 북마크를 가져옵니다.

# NoSQL Workbench 릴리스 기록

다음 표는 NoSQL Workbench 클라이언트 도구의 각 릴리스별 주요 변경사항을 설명합니다.

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| 3.13.5 | 이제 기본 테이블 설정의 용량 모드가 온디맨드 | 기본 설정으로 테이블을 생성하면 DynamoDB가 프로비저닝된 용량 모드 대신 온디맨드 용량 모드를 사용하는 테이블을 생성합니다. | 2025년 2월 24일 |
| 3.13.0 | NoSQL Workbench 작업 빌더 개선 | NoSQL Workbench에는 이제 다크 모드에 대한 기본 지원이 포함됩니다. 작업 빌더의 테이블 및 항목 작업이 개선되었습니다. 항목 결과 및 작업 빌더 요청 정보가 JSON 형식으로 제공됩니다. | 2024년 4월 24일 |
| 3.12.0 | NoSQL Workbench를 사용한 테이블 복제 및 사용된 용량 반환 | 이제 DynamoDB 로컬과 DynamoDB 웹 서비스 계정 간에 또는 DynamoDB 웹 서비스 계정들 간에 테이블을 복제하여 개발 반복 속도를 높일 수 있습니다. 작업 빌더를 사용하여 작업을 실행한 후 소비된 RCU 또는 WCU를 확인합니다. | 2024년 2월 26일 |

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| | | CSV 파일에서 가져올 때 발생하는 데이터 덮어쓰기 문제를 수정했습니다. | |
| 3.11.0 | DynamoDB 로컬 개선 | 이제 기본 제공되는 DynamoDB 로컬 인스턴스를 시작할 때 포트를 지정할 수 있습니다. 이제 관리자 권한 없이 Windows에 NoSQL Workbench를 설치할 수 있습니다. 데이터 모델 템플릿을 업데이트했습니다. | 2024년 1월 17일 |
| 3.10.0 | Apple 실리콘에 대한 기본 지원 | NoSQL Workbench는 이제 Apple 실리콘을 사용하는 Mac에 대한 기본 지원을 포함합니다. 이제 `Number` 유형의 속성에 대한 샘플 데이터 생성 형식을 구성할 수 있습니다. | 2023년 12월 5일 |
| 3.9.0 | 데이터 모델 제작자 개선 | 이제 Visualizer는 기존 테이블을 덮어쓰는 옵션을 사용하여 데이터 모델을 DynamoDB Local로 커밋할 수 있도록 지원합니다. | 2023년 11월 3일 |

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| 3.8.0 | 샘플 데이터 생성 | NoSQL Workbench는 이제 DynamoDB 데이터 모델을 위한 샘플 데이터 생성을 지원합니다. | 2023년 9월 25일 |
| 3.6.0 | 작업 빌더 개선 사항 | 작업 빌더에서 연결 관리 개선. 이제 데이터를 삭제하지 않고 데이터 모델 제작자의 속성 이름을 변경할 수 있습니다. 기타 버그 수정 사항. | 2023년 4월 11일 |
| 3.5.0 | 새 AWS 리전 지원 | 이제 NoSQL Workbench가 ap-south-2, ap-southeast-3, ap-southeast-4, eu-central-2, eu-south-2, me-central-1 및 me-west-1 리전을 지원합니다. | 2023년 2월 23일 |
| 3.4.0 | LynamoDB Local 지원 | 이제 NoSQL Workbench에서 설치 프로세스의 일부로 DynamoDB Local을 설치할 수 있습니다. | 2022년 12월 6일 |
| 3.3.0 | 컨트롤 플레인 작업 지원 | 이제 작업 빌더에서 컨트롤 플레인 작업을 지원합니다. | 2022년 6월 1일 |

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| 3.2.0 | CSV 가져오기 및 내보내기 | 이제 Visualizer 도구의 CSV 파일에서 샘플 데이터를 가져올 수 있으며, 작업 빌더 쿼리의 결과를 CSV 파일로 내보낼 수도 있습니다. | 2021년 10월 11일 |
| 3.1.0 | 작업 저장 | 이제 NoSQL Workbench의 작업 빌더에서는 나중에 사용하기 위해 작업을 저장할 수 있습니다. | 2021년 7월 12일 |
| 3.0.0 | 용량 설정 및 CloudFormation 가져오기/내보내기 | Amazon DynamoDB용 NoSQL Workbench에서는 이제 테이블의 읽기/쓰기 용량 모드를 지정할 수 있고 데이터 모델을 AWS CloudFormation 형식으로 가져오고 내보낼 수 있습니다. | 2021년 4월 21일 |
| 2.2.0 | PartiQL 지원 | Amazon DynamoDB용 NoSQL Workbench에 DynamoDB용 PartiQL 문 빌드에 대한 지원이 추가되었습니다. | 2020년 12월 4일 |
| 1.1.0 | Linux에 대한 지원. | Amazon DynamoDB용 NoSQL Workbench가 Linux(Ubuntu, Fedora 및 Debian)에서 지원됩니다. | 2020년 5월 4일 |

| 버전 | 변경 사항 | 설명 | 날짜 |
| --- | --- | --- | --- |
| 1.0.0 | Amazon DynamoDB 용 NoSQL Workbench – 정식 버전 | Amazon DynamoDB 용 NoSQL Workbench 를 상용 버전으로 사용할 수 있습니다. | 2020년 3월 2일 |
| 0.4.1 | IAM 역할 및 임시 보안 자격 증명 지원 | Amazon DynamoDB 용 NoSQL Workbench 에 AWS Identity and Access Managemen t(IAM) 역할 및 임시 보안 자격 증명에 대한 지원이 추가되었습니다. | 2019년 12월 19일 |
| 0.3.1 | DynamoDB Local(다운로드 가능 버전) 지원 | NoSQL Workbench 는 이제 DynamoDB Local(다운로드 가능 버전)에 대한 연결을 지원하므로 DynamoDB 테이블을 설계, 생성, 쿼리 및 관리할 수 있습니다. | 2019년 11월 8일 |
| 0.2.1 | 릴리스된 NoSQL Workbench 평가판 | 이것은 DynamoDB용 NoSQL Workbench 의 최초 릴리스입니다. NoSQL Workbench를 사용하여 DynamoDB 테이블을 설계하고, 생성하고, 쿼리하고, 관리합니다. | 2019년 9월 16일 |

# DynamoDB에 대한 백업 및 복원

DynamoDB는 온디맨드 백업과 시점 복구(PITR) 백업을 통해 DynamoDB 데이터를 재해 이벤트로부터 보호하며 장기 보존을 위한 데이터 아카이빙 기능을 제공합니다. 프로덕션 애플리케이션의 성능과 가용성에 아무런 영향을 주지 않고도 몇 메가바이트에서 수백 테라바이트에 이르는 데이터 테이블을 백업할 방법이 있습니다. 모든 백업은 자동으로 암호화되고 카탈로그화되며 쉽게 검색할 수 있습니다.

온디맨드 백업을 사용하면 DynamoDB가 저장하고 관리하는 테이블의 스냅샷 백업을 생성할 수 있습니다. 백업의 크기와 기간을 기준으로 요금이 부과됩니다. 온디맨드 백업을 사용하면 전체 DynamoDB 테이블을 백업이 생성되었을 때의 정확한 상태로 복원할 수 있습니다.

DynamoDB 온디맨드 백업을 생성하고 관리할 때 다음과 같은 2가지 옵션이 있습니다.

- DynamoDB
- AWS Backup

DynamoDB 온디맨드 백업 기능을 사용하면 테이블의 전체 백업을 생성하여 규정 준수 요건에 맞도록 장기간 유지하고 보관할 수 있습니다. AWS Management Console에서 또는 단일 API 직접 호출을 사용하여 언제든지 테이블 데이터를 백업하고 복원할 수 있습니다.

시점 복구(PITR) 백업은 DynamoDB에서 완전히 관리되며 초당 최대 35일 동안 시점 복구(PITR)를 제공합니다. 시점 복구(연속 백업)를 사용하려면 DynamoDB 테이블에서 시점 복구(PITR)를 활성화합니다. 테이블에 PITR을 활성화한 기간 동안 DynamoDB 테이블 크기를 기준으로 요금이 부과됩니다. DynamoDB에서 시점 복구(PITR)를 활성화하면 데이터가 지속적으로 백업됩니다. 이렇게 하면 해당 시점의 원래 테이블 상태와 동일한 새 DynamoDB 테이블을 생성하여 PITR 복구 기간 내의 특정 시점으로 테이블을 복원할 수 있습니다.

특정 시점으로 복구를 사용하면 우발적인 쓰기 또는 삭제 작업으로부터 DynamoDB 테이블을 보호할 수 있습니다. 특정 시점으로 복구를 설정해 두면 온디맨드 백업의 생성, 유지 관리, 예약을 걱정할 필요가 없습니다. 테스트 스크립트가 프로덕션 DynamoDB 테이블에 우발적으로 데이터를 쓴 경우를 예로 들어 보겠습니다.

시점 복구가 설정되어 있으면 최근 35일 중 원하는 시점으로 테이블을 복원할 수 있습니다. 복구 기간을 1~35일 사이의 값으로 설정할 수 있습니다. 시점 복구를 활성화한 뒤에는 현재 시간을 기준으로 5분 전부터 구성된 복구 기간까지 원하는 시점으로 복원할 수 있습니다. DynamoDB는 테이블의 증분식 백업을 관리합니다.

또한 특정 시점 작업은 성능이나 API 지연 시간에 영향을 주지 않습니다.

AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 DynamoDB API를 사용하여 DynamoDB 테이블을 특정 시점으로 복원할 수 있습니다. 특정 시점으로 복구 프로세스는 새 테이블로 복원합니다.

AWS 리전의 가용성과 요금에 대한 자세한 내용은 Amazon DynamoDB 요금을 참조하세요.

> **Note**
>
> - DynamoDB 백업에는 태그 지정 및 속성 기반 액세스 제어(ABAC)가 지원되지 않습니다. 백업과 함께 ABAC를 사용하려면 AWS Backup을 사용하는 것이 좋습니다.
> - 태그는 복원된 테이블에 보존되지 않습니다. 정책에서 태그 기반 조건을 사용하려면 복원된 테이블에 태그를 추가해야 합니다.

다음 비디오에서는 백업 및 복원 개념을 소개하고, 특정 시점으로 복구를 자세히 설명합니다.

백업 및 복원

주제

- DynamoDB의 시점 백업
- 온디맨드 DynamoDB 백업 및 복원 사용
- 백업에 대한 Amazon DynamoDB 요금 청구 이해
- DynamoDB에서 테이블 복원
- DynamoDB에서 AWS Backup 사용

# DynamoDB의 시점 백업

Amazon DynamoDB의 시점 복구(PITR)를 통해 DynamoDB 테이블 데이터를 지속적으로 자동 백업할 수 있습니다. 시점 복구(PITR) 백업은 DynamoDB에서 완전히 관리되며 초당 최대 35일 동안 시점 복구(PITR)를 제공합니다. 특정 시점으로 복구를 설정해 두면 온디맨드 백업의 생성, 유지 관리, 예약을 걱정할 필요가 없습니다. 이 단원에서는 DynamoDB에서 이 프로세스가 어떻게 작동하는지 간략히 살펴봅니다.

## 시작하기 전 준비 사항

Amazon DynamoDB 테이블에서 PITR(특정 시점으로 복구)을 활성화하기 전에 다음을 고려하세요.

- `RecoveryPeriodinDays`를 설정하여 연속 백업을 수행하는 기간을 단축할 수 있습니다. 기본적으로 `RecoveryPeriodinDays`는 35입니다. 그런데 1~35 사이의 값으로 설정할 수 있습니다. 요금은 테이블 크기와 로컬 보조 인덱스를 기반으로 하므로 `RecoveryPeriodinDays`를 단축해도 PITR 요금에는 영향을 미치지 않습니다.
- 특정 시점으로 복구를 비활성화했다가 나중에 테이블에서 다시 활성화하면 해당 테이블을 복구할 수 있는 시작 시간이 재설정됩니다. 따라서 `LatestRestorableDateTime`을 사용해서 해당 테이블을 즉시 복원하는 것만 가능합니다.
- 전역 테이블의 로컬 복사본 각각에 대해 특정 시점으로 복구를 활성화할 수 있습니다. 테이블을 복원하면 해당 전역 테이블에 속하지 않는 별도의 테이블로 백업이 복원됩니다. 글로벌 테이블의 글로벌 테이블 버전 2019.11.21(현재)을 사용하고 있는 경우 복원된 테이블에서 새로운 글로벌 테이블을 생성할 수 있습니다. 자세한 내용은 DynamoDB 글로벌 테이블 작동 방식 단원을 참조하십시오.
- 또한 복원된 테이블이 원본 테이블이 속하는 리전과 다른 리전에 생성되도록 AWS 리전 간에 DynamoDB 테이블 데이터를 복원할 수 있습니다. AWS 커머셜 리전, AWS 중국 리전 및 AWS GovCloud(미국) 리전 간에 교차 리전 복원을 수행할 수 있습니다. 소스 리전에서 전송한 데이터와 대상 리전의 새 테이블로 복원하는 데 사용한 리소스에 대해서만 요금을 지불하면 됩니다.
- AWS CloudTrail은 특정 시점으로 복구를 위해 모든 콘솔 작업과 API 작업을 로깅하여 로깅, 연속 모니터링 및 감사를 활성화합니다. 자세한 내용은 AWS CloudTrail을 사용하여 DynamoDB 작업 로깅 단원을 참조하십시오.

주제


- DynamoDB에서 특정 시점 복구 사용


## DynamoDB에서 특정 시점 복구 사용

Amazon DynamoDB의 PITR(특정 시점으로 복구)을 통해 DynamoDB 테이블 데이터를 자동으로 백업할 수 있습니다. 이 단원에서는 DynamoDB에서 이 프로세스가 어떻게 작동하는지 간략히 살펴봅니다.

> **Note**
>
> DynamoDB는 테이블 데이터 및 로컬 보조 인덱스를 포함하여 각 DynamoDB 테이블의 크기를 기준으로 PITR 요금을 부과합니다. 복구 기간을 변경해도(예: 35일에서 1일로) 요금이 절감되지 않습니다. 선택한 복구 기간에 관계없이 비용은 동일하게 유지됩니다. 구성된 최대 복구 기간은 PITR을 켜는 데 부과되는 요금에 영향을 주지 않습니다. 백업 요금을 결정하기 위해 DynamoDB는 PITR이 켜진 테이블의 크기를 지속적으로 모니터링합니다. 각 테이블의 PITR을 끄기 전까지는 PITR 사용량에 대한 요금이 청구됩니다.

주제


- 특정 시점으로 복구 활성화
- PITR 활성화(콘솔)
- PITR 활성화(AWS CLI)
- PITR 활성화(AWS CloudFormation)
- PITR 활성화(API)
- 복구 기간
- PITR 편집
- PITR이 활성화된 테이블 삭제


## 특정 시점으로 복구 활성화

AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 DynamoDB API를 사용하여 특정 시점으로 복구를 활성화할 수 있습니다. 특정 시점으로 복구가 활성화되면 사용자가 명시적으로 이 기능을 끌 때까지 지속적으로 백업됩니다.

특정 시점으로 복구를 활성화한 뒤에는 `EarliestRestorableDateTime`과 `LatestRestorableDateTime` 사이의 원하는 시점으로 복원할 수 있습니다. `LatestRestorableDateTime`은 일반적으로 현재 시간으로부터 5분 전입니다. 자세한 내용은 DynamoDB 테이블을 특정 시점으로 복원 섹션을 참조하세요.

> **Note**
>
> 특정 시점으로 복구 프로세스는 항상 새 테이블로 복원됩니다.

## PITR 활성화(콘솔)

DynamoDB 콘솔을 사용하여 PITR을 활성화하려면

1. DynamoDB 콘솔로 이동합니다.
2. 왼쪽 탐색 메뉴에서 테이블을 선택하고 DynamoDB 테이블을 선택합니다.
3. 백업 탭에서 시점 복구 옵션에 대해 편집을 선택합니다.
4. 시점 복구 켜기를 선택합니다.
5. 백업 복구 기간으로 1~35 사이의 값을 선택합니다. 이는 연속 백업을 복구할 수 있는 최대 기간을 나타냅니다.

## PITR 활성화(AWS CLI)

> **Note**
>
> AWS CLI 명령을 실행할 때 오류가 발생하는 경우 AWS CLI 오류 문제 해결을 참조하세요. 최신 AWS CLI 버전을 사용해야 합니다.

point-in-time-recovery-specification 설정을 켠 상태에서 update-continuous-backups 명령을 실행합니다.

```
aws dynamodb update-continuous-backups \
--table-name Music \
--point-in-time-recovery-specification
 PointInTimeRecoveryEnabled=true,RecoveryPeriodInDays=35
```

## PITR 활성화(AWS CloudFormation)

`PointInTimeRecoverySpecification` 속성이 켜진 상태에서 AWS::DynamoDB::Table 리소스를 사용합니다.

```
Resources:
  iotCatalog:
    Type: AWS::DynamoDB::Table
      Properties:
      ...
      PointInTimeRecoverySpecification:
        PointInTimeRecoveryEnabled: true
        RecoveryPeriodInDays: 35
```

요청 구문 예시:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays: number
   },
   "TableName": "string"
}
```

## PITR 활성화(API)

`PointInTimeRecoverySpecification` 파라미터를 켠 상태에서 UpdateContinuousBackups API 작업을 실행합니다.

요청 구문 예시:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays" : number
   },
   "TableName": "string"
}
```

응답 구문 예시:

```
{
   "ContinuousBackupsDescription": {
      "ContinuousBackupsStatus": "string",
      "PointInTimeRecoveryDescription": {
         "PointInTimeRecoveryStatus": "string",
         "EarliestRestorableDateTime": number,
         "RecoveryPeriodInDays": number,
         "LatestRestorableDateTime": number
      }
   }
}
```

Python

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.update_continuous_backups(
    TableName=<table_name>,
    PointInTimeRecoverySpecification={
        'PointInTimeRecoveryEnabled': True,
        'RecoveryPeriodInDays': 35
    }
)
```

### 복구 기간

연속 백업의 복구 기간을 1~35일로 설정할 수 있습니다. 이 `RecoveryPeriodInDays`는 연속 백업이 유지되는 기간을 결정합니다. 예를 들어 이 값을 30일로 설정하면 지난 30일 동안의 특정 시점으로만 테이블을 복원할 수 있습니다.

> **Note**
>
> DynamoDB는 테이블 데이터 및 로컬 보조 인덱스를 포함하여 각 DynamoDB 테이블의 크기를 기준으로 PITR 요금을 부과합니다. 구성된 최대 복구 기간은 PITR을 켜는 데 부과되는 요금에 영향을 주지 않습니다. 요금 세부 정보는 DynamoDB 요금을 참조하세요.

### PITR 편집

테이블에서 PITR 설정을 편집하고 복구 기간을 변경할 수 있습니다. 복구 기간을 변경하고 이전에 설정한 값보다 높은 값으로 늘리는 경우 `EarliestRestorePoint`가 즉시 변경되지 않습니다. 복구 기간은 롤링 기간이므로 DynamoDB는 새로 늘어난 기간에 도달할 때까지 자동 백업을 계속 수행합니다. 복구 기간을 변경하고 이전에 설정한 값보다 낮은 값으로 줄이는 경우 `EarliestRestorePoint`가 복구 기간과 일치하도록 즉시 감소하며 새 설정 값을 벗어나는 연속 백업은 복구할 수 없게 됩니다.

### PITR이 활성화된 테이블 삭제

특정 시점으로 복구가 활성화된 테이블을 삭제하면 DynamoDB는 시스템 백업이라는 백업 스냅샷을 자동으로 생성하고 35일 동안 유지합니다(추가 비용 없음). 시스템 백업을 사용하면 삭제된 테이블을 삭제 시점 전의 상태로 복원할 수 있습니다. 모든 시스템 백업은 표준 이름 지정 규칙인 *`table-name`*`$DeletedTableBackup`을 따릅니다.

> **Note**
>
> 시점 복구가 활성화된 테이블이 삭제되면 시스템 백업을 사용하여 해당 테이블을 단일 시점으로 복원할 수 있습니다. 시스템 백업은 테이블 삭제 시 생성되며 테이블이 삭제되기 직전의 테이블 스냅샷입니다.

## 온디맨드 DynamoDB 백업 및 복원 사용

Amazon DynamoDB는 독립형 온디맨드 백업 및 복원 기능을 지원합니다. 이러한 기능은 AWS 백업 사용 여부와 관계없이 사용할 수 있습니다.

DynamoDB 온디맨드 백업 기능을 사용하면 테이블의 전체 백업을 생성하여 규정 준수 요건에 맞도록 장기간 유지하고 보관할 수 있습니다. AWS 관리 콘솔에서 클릭 한 번으로 또는 단일 API 호출을 사용하여 언제든지 테이블 데이터를 백업하고 복원할 수 있습니다. 백업 및 복원 작업을 실행해도 테이블 성능이나 가용성에는 아무런 영향을 주지 않습니다.

콘솔, AWS 명령줄 인터페이스(AWS CLI) 또는 DynamoDB API를 사용하여 테이블 백업을 생성할 수 있습니다. 자세한 내용은 DynamoDB 테이블 백업 섹션을 참조하세요.

백업에서 테이블을 복원하는 방법은 백업에서 DynamoDB 테이블 복원 단원을 참조하십시오.

## DynamoDB를 사용하여 DynamoDB 테이블 백업 및 복원: 작동 방식

DynamoDB 온디맨드 백업 기능을 사용하여 Amazon DynamoDB 테이블의 전체 백업을 생성할 수 있습니다. 이 기능은 AWS 백업에서 독립적으로 사용할 수 있습니다. 이 섹션에서는 DynamoDB 백업 및 복원 프로세스를 수행하는 중에 발생하는 상황에 대한 개요를 제공합니다.

### 백업

DynamoDB에서 온디맨드 백업을 생성하는 경우, 해당 요청의 타임 마커가 카탈로그화됩니다. 백업은 마지막 전체 테이블 스냅샷에 대한 요청 시간까지 모든 변경 내용을 적용하여 비동기식으로 생성됩니다. DynamoDB 백업 요청은 즉시 처리되며 몇 분 내에 복원할 수 있는 상태가 됩니다.

> **Note**
>
> 온디맨드 백업을 생성할 때마다 전체 테이블 데이터가 백업됩니다. 수행할 수 있는 온디맨드 백업 수에는 제한이 없습니다.

DynamoDB의 모든 백업은 테이블의 프로비저닝된 처리량을 사용하지 않고 작동합니다.

DynamoDB 백업은 항목에 걸쳐 인과 관계의 일관성을 보장하지는 않지만, 백업의 업데이트 간 간격은 보통 1초 미만입니다.

백업이 진행 중인 동안에는 다음 작업을 수행할 수 없습니다.

- 백업 작업을 일시 중지하거나 취소합니다.
- 백업의 원본 테이블을 삭제합니다.
- 테이블에 대한 백업이 진행 중인 경우 해당 테이블에서 백업을 비활성화합니다.

예약 스크립트 및 정리 작업을 생성하지 않으려면 AWS Backup을 사용하여 DynamoDB 테이블에 대한 일정 및 보존 정책을 사용하여 백업 계획을 생성할 수 있습니다. AWS Backup은 백업을 실행하고 만료되면 삭제합니다. 자세한 내용은 개발자 안내서AWS Backup를 참조하세요.

AWS Backup 외에도 AWS Lambda 기능을 사용하여 정기적인 백업이나 향후 백업을 예약할 수 있습니다. 자세한 내용은 블로그 게시물 Amazon DynamoDB 온디맨드 백업 예약을 위한 서버리스 솔루션을 참조하세요.

콘솔을 사용하는 경우 AWS Backup을 사용하여 생성된 모든 백업은 Backup type(백업 유형)이 AWS로 설정되어 Backups(백업) 탭에 나열됩니다.

> **Note**
>
> Backup type(백업 유형)이 AWS로 표시된 백업은 DynamoDB 콘솔을 사용하여 삭제할 수 없습니다. 이러한 백업을 관리하려면 AWS Backup 콘솔을 사용합니다.

백업을 수행하는 방법은 DynamoDB 테이블 백업 단원을 참조하십시오.

### 복원

테이블의 프로비저닝된 처리량을 사용하지 않고 해당 테이블을 복원합니다. DynamoDB 백업에서 전체 테이블 복원을 수행하거나 대상 테이블 설정을 구성할 수 있습니다. 복원을 수행할 때 다음 테이블 설정을 변경할 수 있습니다.

- 글로벌 보조 인덱스(GSI)
- 로컬 보조 인덱스(LSI)
- 결제 모드
- 프로비저닝된 읽기 및 쓰기 용량
- 암호화 설정

> **Important**
>
> 전체 테이블 복원을 수행할 때 대상 테이블은 백업이 요청된 시점에 기록된 대로, 원본 테이블과 동일하게 프로비저닝된 읽기 용량 단위 및 쓰기 용량 단위로 설정됩니다. 복원 프로세스는 로컬 보조 인덱스와 글로벌 보조 인덱스도 복원합니다.

또한 복원된 테이블이 백업이 속하는 리전과 다른 리전에 생성되도록 AWS 리전 간에 DynamoDB 테이블 데이터를 복원할 수 있습니다. AWS 커머셜 리전, AWS 중국 리전 및 AWS GovCloud(미국) 리전 간에 교차 리전 복원을 수행할 수 있습니다. 소스 리전에서 전송한 데이터와 대상 리전의 새 테이블로 복원하는 데 사용된 리소스에 대해서만 요금을 지불하면 됩니다.

새로 복원된 테이블에 일부 또는 전체 보조 인덱스가 생성되지 않도록 제외한 경우 복원이 보다 빠르고 비용 효율적일 수 있습니다.

복원된 테이블에서 다음을 수동으로 설정해야 합니다.

- Auto Scaling 정책
- AWS Identity and Access Management (IAM) 정책
- Amazon CloudWatch 지표 및 경보
- Tags
- 스트림 설정
- Time To Live(TTL) 설정
- 삭제 방지 설정
- 시점 복구(PITR) 설정

전체 테이블 데이터만 백업에서 새 테이블로 복원할 수 있습니다. 복원된 테이블은 활성 상태가 된 이후에만 쓸 수 있습니다.

> **Note**
>
> 복원 작업 중에는 기존 테이블을 덮어쓸 수 없습니다.

서비스 지표에는 고객 테이블 복원의 95%가 1시간 이내에 완료된다고 표시됩니다. 그러나 복원 시간은 테이블 구성(예: 테이블 크기 및 기본 파티션 수) 및 기타 관련 변수와 직접 관련이 있습니다. 재해 복구를 계획할 때 가장 좋은 방법은 평균 복원 완료 시간을 정기적으로 기록하고 이러한 시간이 전체 복구 시간 목표에 어떤 영향을 미치는지 설정하는 것입니다.

복원을 수행하는 방법은 백업에서 DynamoDB 테이블 복원 단원을 참조하십시오.

액세스 제어에 IAM 정책을 사용할 수 있습니다. 자세한 내용은 DynamoDB 백업 및 복원에 IAM 사용 섹션을 참조하세요.

모든 백업 및 복원 콘솔 작업과 API 작업이 로깅, 연속 모니터링 및 감사를 위해 AWS CloudTrail에 캡처되고 기록됩니다.

## DynamoDB 테이블 백업

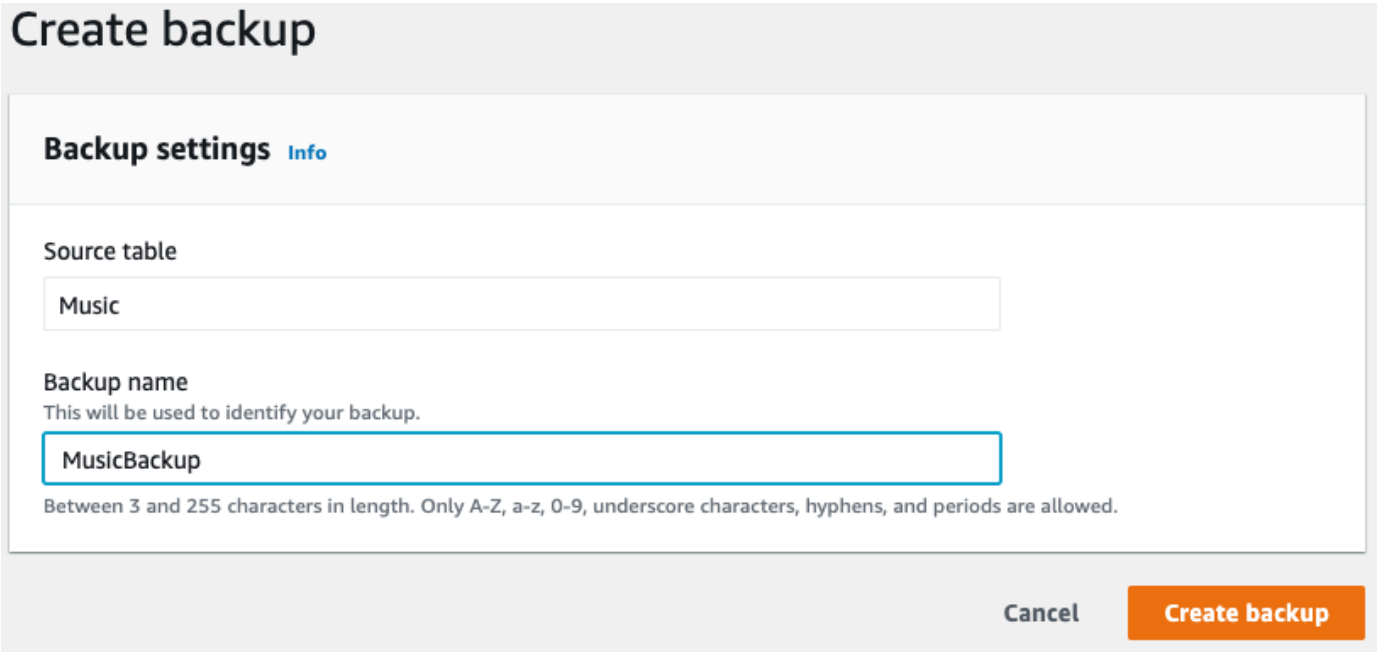
이 단원에서는 Amazon DynamoDB 콘솔 또는 AWS Command Line Interface를 사용하여 테이블을 백업하는 방법을 설명합니다.

테이블 백업 생성(콘솔)

다음 단계에 따라 AWS Management Console을 사용하여 기존 `Music` 테이블에 대해 이름이 `MusicBackup`인 백업을 생성합니다.

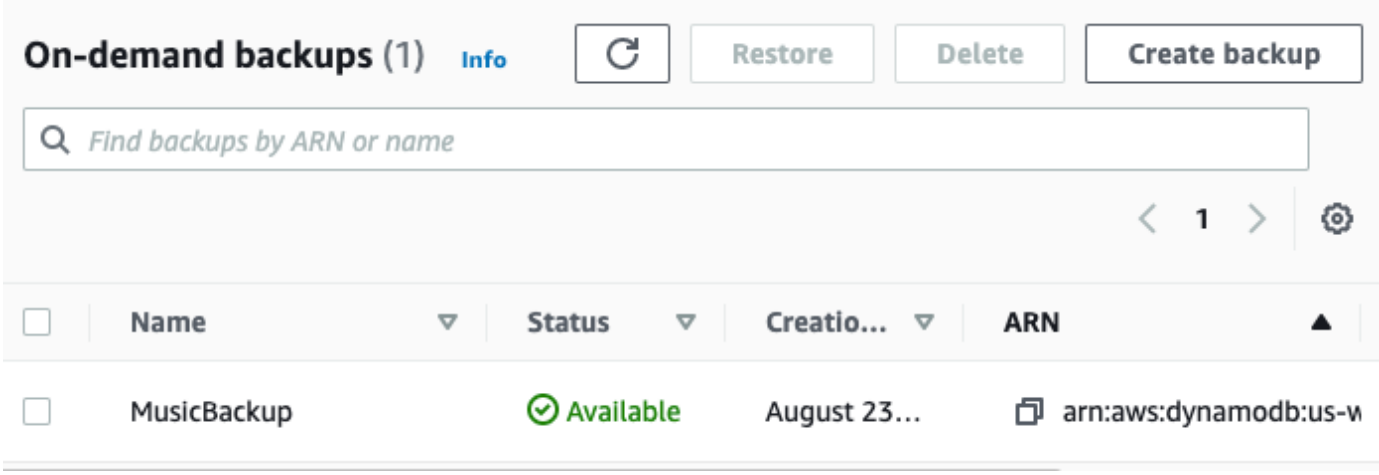
테이블 백업을 생성하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 다음 중 하나를 수행하여 백업을 생성할 수 있습니다.
   - `Music` 테이블의 Backups(백업) 탭에서 Create backup(백업 생성)을 선택합니다.
   - 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다. 그런 다음 [Create backup]을 선택합니다.
3. 테이블 이름이 `Music`인지 확인하고 백업 이름으로 **`MusicBackup`**을 입력합니다. 그런 다음 백업 생성을 선택하여 백업을 생성합니다.



> **Note**
>
> 탐색 창의 [Backups] 섹션을 사용하여 백업을 생성하는 경우, 테이블이 자동으로 미리 선택되어 있지 않습니다. 백업에 대한 원본 테이블 이름을 수동으로 선택해야 합니다.

백업 상태는 백업이 생성되는 동안 [Creating]으로 설정되며, 백업이 완료되면 백업 상태가 Available로 바뀝니다.




테이블 백업 생성(AWS CLI)

다음 단계에 따라 AWS CLI를 사용하여 기존 `Music` 테이블에 대한 백업을 생성합니다.

테이블 백업을 생성하려면

- `Music` 테이블에 대해 이름이 `MusicBackup`인 백업을 생성합니다.

```
aws dynamodb create-backup --table-name Music \
 --backup-name MusicBackup
```

백업 상태는 백업이 생성되는 동안 `CREATING`으로 설정됩니다.

```
{
    "BackupDetails": {
        "BackupName": "MusicBackup",
        "BackupArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc",
```

```
        "BackupStatus": "CREATING",
        "BackupCreationDateTime": 1489602797.149
    }
}
```

백업이 완료되면 BackupStatus가 AVAILABLE로 바뀌어야 합니다. 제대로 바뀌었는지 확인하려면 describe-backup 명령을 사용합니다. 이전 단계의 출력에서 또는 list-backups 명령을 사용하여 backup-arn의 입력 값을 얻을 수 있습니다.

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/01489173575360-b308cd7d
```

백업을 추적하기 위해 list-backups 명령을 사용할 수 있습니다. 이 명령은 CREATING 또는 AVAILABLE 상태에 있는 모든 백업을 나열합니다.

```
aws dynamodb list-backups
```

list-backups 명령과 describe-backup 명령은 백업의 원본 테이블에 대한 정보를 확인하는 데 유용합니다.

## 백업에서 DynamoDB 테이블 복원

이 단원에서는 Amazon DynamoDB 콘솔 또는 AWS Command Line Interface(AWS CLI)를 사용하여 백업에서 테이블을 복원하는 방법을 설명합니다.

> **Note**
>
> AWS CLI를 사용하려면 먼저 구성을 해야 합니다. 자세한 내용은 DynamoDB 액세스 섹션을 참조하세요.

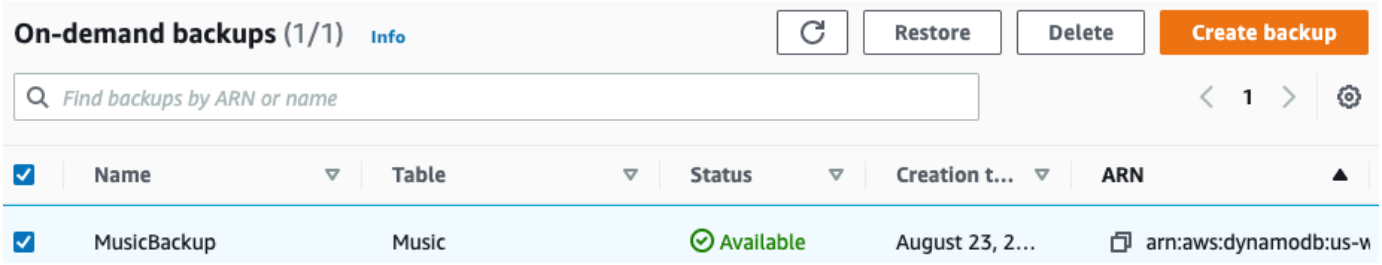
백업에서 테이블 복원(콘솔)

다음 절차는 Music 자습서에서 만든 MusicBackup 파일을 사용하여 DynamoDB 테이블 백업 테이블을 복원하는 방법을 보여 줍니다.

> **Note**
>
> 이 절차는 `MusicBackup` 파일을 사용하여 이를 복원하기 전에 `Music` 테이블이 더 이상 존재하지 않음을 가정합니다.

백업에서 테이블을 복원하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.
3. 백업 목록에서 `MusicBackup`을 선택합니다.




4. 복원을 선택합니다.
5. **`Music`**을 새 테이블 이름으로 입력합니다. 백업 이름과 기타 백업 세부 정보를 확인합니다. [Restore table]을 선택하여 복원 프로세스를 시작합니다.

> **Note**
>
> 테이블을 동일한 AWS 리전 또는 백업이 속하는 리전과 다른 리전에 복원할 수 있습니다. 새로 복원된 테이블에 보조 인덱스가 생성되지 않도록 제외할 수도 있습니다. 또한 다른 암호화 모드를 지정할 수 있습니다.
>
> 백업에서 복원된 테이블은 항상 DynamoDB Standard 테이블 클래스를 사용하여 생성됩니다.

# Restore table from backup Info

Restoring a table from a backup will restore it as a new table.

## Restore settings

Name of restored table

This name will identify your restored table.

Music

Between 3 and 255 characters in length. Only A–Z, a–z, 0–9, underscore characters, hyphens, and periods allowed.

Secondary indexes

- (●) Restore the entire table
  Your restored table will include all local and global secondary indexes.
- ( ) Restore the table without secondary indexes
  Your restored table will exclude all local and global secondary indexes. Restoring this way can be faster and more cost efficient.

## Destination AWS Region

- (●) Same Region (Oregon)
  Restore the table to the same Region as the original table.
- ( ) Cross-Region
  Restore the table to a different Region for greater redundancy but with higher data transfer costs.

## ▼ Encryption at rest - *optional*

All user data stored in Amazon DynamoDB is fully encrypted at rest. By default, Amazon DynamoDB manages the encryption key, and you are not charged any fee for using it.

Encryption key management Info

- (●) Owned by Amazon DynamoDB
  The key is owned and managed by DynamoDB. You are not charged an additional fee for using this customer master key (CMK).
- ( ) AWS managed CMK
  The key is stored in your account and is managed by AWS Key Management Service (AWS KMS). AWS KMS charges apply.
- ( ) Stored in your account, and owned and managed by you
  Choose a key that is owned and managed by you, and stored in AWS KMS.

ⓘ The time it takes to restore a table from a backup can vary and is based on multiple variables. After your table is restored from the backup, you might need to reapply configuration settings. Learn more

Cancel Restore

복원하는 테이블은 상태가 [Creating]으로 표시됩니다. 복원 프로세스가 완료되면 Music 테이블의 상태가 Active로 변경됩니다.

### 백업에서 테이블 복원(AWS CLI)

다음 단계에 따라 AWS CLI를 사용하여 DynamoDB 테이블 백업 자습서에서 만든 MusicBackup을 이용해 Music 테이블을 복원합니다.

백업에서 테이블을 복원하려면

1. list-backups 명령을 사용하여 복원하려는 백업을 확인합니다. 이 예제에서는 MusicBackup를 사용합니다.

   ```
   aws dynamodb list-backups
   ```

   백업에 대한 자세한 정보를 보려면 describe-backup 명령을 사용합니다. 이전 단계의 backup-arn에서 입력을 볼 수 있습니다.

   ```
   aws dynamodb describe-backup \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

2. 백업에서 테이블을 복원합니다. 이 경우 MusicBackup은 Music 테이블을 동일한 AWS 리전으로 복원합니다.

   ```
   aws dynamodb restore-table-from-backup \
   --target-table-name Music \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

3. 사용자 지정 테이블 설정을 통해 백업에서 테이블을 복원합니다. 이 경우 MusicBackup은 Music 테이블을 복원하고 복원된 테이블의 암호화 모드를 지정합니다.

   > **Note**
   >
   > sse-specification-override 파라미터는 CreateTable 명령에 사용된 sse-specification-override 파라미터와 동일한 값을 사용합니다. 자세한 내용은 DynamoDB의 암호화된 테이블 관리을 참조하십시오.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

테이블을 백업이 속하는 다른 AWS 리전으로 복원할 수 있습니다.

> **Note**
>
> - 이 `sse-specification-override` 파라미터는 교차 리전 복원에는 필수지만 원본 테이블과 동일한 리전으로 복원하는 경우에는 선택 사항입니다.
> - 명령줄에서 교차 리전 복원을 수행할 때는 기본 AWS 리전을 원하는 대상 리전으로 설정해야 합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 명령줄 옵션을 참조하세요.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS
```

복원된 테이블에 대한 결제 모드와 프로비저닝된 처리량을 재정의할 수 있습니다.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d \
--billing-mode-override PAY_PER_REQUEST
```

복원된 테이블에 일부 또는 전체 보조 인덱스가 생성되지 않도록 제외할 수 있습니다.

> **Note**
>
> 복원된 테이블에 일부 또는 전체 보조 인덱스가 생성되지 않도록 제외하는 경우 복원이 보다 빠르고 비용 효율적일 수 있습니다.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581081403719-db9c1f91 \
--global-secondary-index-override '[]' \
--sse-specification-override Enabled=true,SSEType=KMS
```

> **Note**
>
> 제공된 보조 인덱스는 기존 인덱스와 일치해야 합니다. 복원 시 새 인덱스를 생성할 수 없습니다.

다른 재정의 조합을 사용할 수 있습니다. 예를 들어, 다음과 같이 단일 글로벌 보조 인덱스를 사용하는 동시에 프로비저닝된 처리량을 변경할 수 있습니다.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:eu-west-1:123456789012:table/Music/
backup/01581082594992-303b6239 \
--billing-mode-override PROVISIONED \
--provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100 \
--global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEYS_
 \
--sse-specification-override Enabled=true,SSEType=KMS
```

복원을 확인하려면 `describe-table` 명령을 사용하여 `Music` 테이블을 명시합니다.

```
aws dynamodb describe-table --table-name Music
```

백업에서 복원하는 테이블은 상태가 [Creating]으로 표시됩니다. 복원 프로세스가 완료되면 `Music` 테이블의 상태가 Active로 변경됩니다.

> **Important**
>
> 복원 진행 중에는 IAM 역할 정책을 수정하거나 삭제하지 마세요. 그럴 경우 예기치 않은 동작이 발생할 수 있습니다. 예를 들어 테이블 복원 중에 쓰기 권한을 삭제할 경우, 기본 `RestoreTableFromBackup` 작업에서, 복원된 데이터를 테이블에 쓸 수 없게 됩니다.
> 복원 작업이 완료된 후에는 IAM 역할 정책을 수정하거나 삭제해도 됩니다.
> 대상 복원 테이블에 액세스하기 위한 소스 IP 제한 관련 IAM 정책에서는 `aws:ViaAWSService` 키를 `false`로 설정하여 해당 제한이 보안 주체가 직접 수행한 요청에만 적용되도록 해야 합니다. 그러지 않으면 복원이 취소됩니다.
> AWS 관리형 키 또는 고객 관리형 키로 백업이 암호화된 경우, 복원이 진행되는 동안 키를 비활성화하거나 삭제하지 마세요. 그렇지 않으면 복원에 실패합니다.
> 복원 작업이 완료된 후에는 복원된 테이블에 암호화된 키를 변경하거나 이전 키를 비활성화 또는 삭제할 수 있습니다.

## DynamoDB 테이블 백업 삭제

이 단원에서는 AWS Management Console 또는 AWS Command Line Interface(AWS CLI)를 사용하여 Amazon DynamoDB 테이블 백업을 삭제하는 방법에 대해 설명합니다.

> **Note**
>
> AWS CLI를 사용하고 싶다면 먼저 구성해야 합니다. 자세한 내용은 AWS CLI 사용 섹션을 참조하세요.

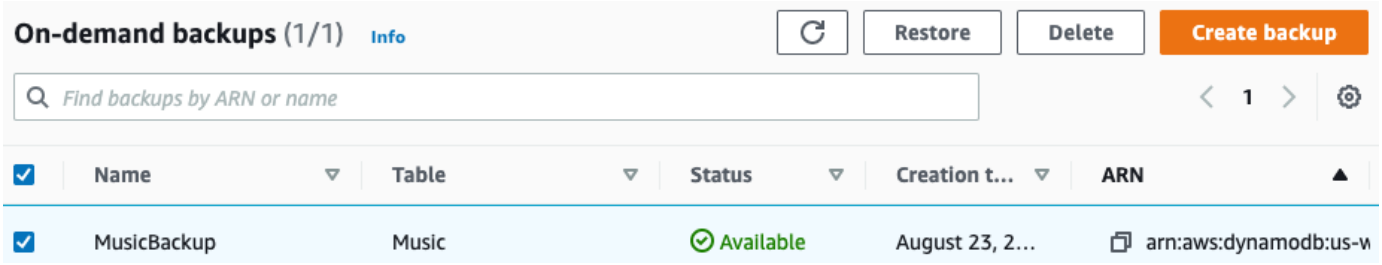
### 테이블 백업 삭제(콘솔)

다음 절차는 콘솔을 사용하여 DynamoDB 테이블 백업 자습서에서 생성한 `MusicBackup`을 삭제하는 방법을 보여줍니다.

#### 백업 삭제

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.

3. 백업 목록에서 MusicBackup을 선택합니다.




4. Delete(삭제)를 선택합니다. **delete**를 입력하고 삭제(Delete)를 클릭하여 백업을 삭제할 것인지 확인합니다.

테이블 백업 삭제(AWS CLI)

다음은 AWS CLI를 사용하여 기존 Music 테이블의 백업을 삭제하는 예제입니다.

```
aws dynamodb delete-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc
```

## DynamoDB 백업 및 복원에 IAM 사용

AWS Identity and Access Management(IAM)를 사용하여 일부 리소스에 대한 Amazon DynamoDB 백업 및 복원 작업을 제한할 수 있습니다. CreateBackup 및 RestoreTableFromBackup API는 테이블 단위로 작동합니다.

DynamoDB에서 IAM 정책을 사용하는 방법에 대한 자세한 내용은 DynamoDB에 대한 자격 증명 기반 정책 단원을 참조하세요.

다음은 DynamoDB에서 특정 백업 및 복원 기능을 구성하는 데 사용할 수 있는 IAM 정책의 예입니다.

### 예 1: CreateBackup 및 RestoreTableFromBackup 작업 허용

다음 IAM 정책은 모든 테이블에서 CreateBackup 및 RestoreTableFromBackup DynamoDB 작업을 수행할 수 있는 권한을 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        }
    ]
}
```

**Important**

소스 백업에는 DynamoDB RestoreTableFromBackup 권한이 필요하며, 복원 기능을 위해서는 대상 테이블에 대한 DynamoDB 읽기 및 쓰기 권한이 필요합니다.
소스 테이블에는 DynamoDB RestoreTableToPointInTime 권한이 필요하며, 복원 기능을 위해서는 대상 테이블에 대한 DynamoDB 읽기 및 쓰기 권한이 필요합니다.

### 예 2: CreateBackup 허용 및 RestoreTableFromBackup 거부

다음 IAM 정책은 `CreateBackup` 작업에 대한 권한을 부여하고, `RestoreTableFromBackup` 작업을 거부합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": "*"
```

```
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:RestoreTableFromBackup"],
            "Resource": "*"
        }

    ]
}
```

### 예 3: ListBackups 허용, CreateBackup과 RestoreTableFromBackup 거부

다음 IAM 정책은 `ListBackups` 작업에 대한 권한을 부여하고, `CreateBackup` 및 `RestoreTableFromBackup` 작업을 거부합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "*"
        }

    ]
}
```

### 예 4: ListBackups 허용 및 DeleteBackup 거부

다음 IAM 정책은 `ListBackups` 작업에 대한 권한을 부여하고, `DeleteBackup` 작업을 거부합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:DeleteBackup"],
            "Resource": "*"
        }

    ]
}
```

### 예 5: 모든 리소스에 대해 RestoreTableFromBackup 및 DescribeBackup 허용, 특정 백업에 대해 DeleteBackup 거부

다음 IAM 정책은 특정 백업 리소스에 대해 `RestoreTableFromBackup` 및 `DescribeBackup` 작업에 대한 권한을 부여하고, `DeleteBackup` 작업을 거부합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DeleteBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        }
    ]
}
```

> **Important**
>
> 소스 백업에는 DynamoDB RestoreTableFromBackup 권한이 필요하며, 복원 기능을 위해서는 대상 테이블에 대한 DynamoDB 읽기 및 쓰기 권한이 필요합니다.
> 소스 테이블에는 DynamoDB RestoreTableToPointInTime 권한이 필요하며, 복원 기능을 위해서는 대상 테이블에 대한 DynamoDB 읽기 및 쓰기 권한이 필요합니다.

## 예 6: 특정 테이블에 대해 CreateBackup 허용

다음 IAM 정책은 `Movies` 테이블에서만 `CreateBackup` 작업에 대한 권한을 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/Movies"
            ]
        }
    ]
}
```

### 예 7: ListBackups 허용

다음 IAM 정책은 `ListBackups` 작업에 대한 권한을 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        }
    ]
}
```

> **Important**
>
> 특정 테이블에서 `ListBackups` 작업에 대한 권한을 부여할 수 없습니다.

### 예제 8: AWS Backup 기능에 액세스 허용

고급 기능이 포함된 성공적인 백업을 위해서는 `StartAwsBackupJob` 작업에 대한 API 권한이 필요하고 해당 백업을 성공적으로 복원하려면 `dynamodb:RestoreTableFromAwsBackup` 작업에 대한 API 권한이 필요합니다.

다음 IAM 정책은 AWS Backup에 고급 기능 및 복원을 사용하여 백업을 트리거할 수 있는 권한을 부여합니다. 또한 테이블이 암호화된 경우 정책에서 AWS KMS 키에 액세스할 수 있어야 합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:StartAwsBackupJob",
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        },
        {
            "Sid": "AllowRestoreFromAwsBackup",
            "Effect": "Allow",
            "Action": [
                "dynamodb:RestoreTableFromAwsBackup"
            ],
            "Resource": "*"
        }
    ]
}
```

## 예제 9: 특정 소스 테이블에 대한 RestoreTableToPointInTime 거부

다음 IAM 정책은 특정 소스 테이블에 대한 `RestoreTableToPointInTime` 작업 권한을 거부합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableToPointInTime"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music"
        }
    ]
}
```

### 예제 10: 특정 소스 테이블의 모든 백업에 대한 RestoreTableFromBackup 거부

다음 IAM 정책은 특정 소스 테이블의 모든 백업에 대한 `RestoreTableToPointInTime` 작업 권한을 거부합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/*"
        }
    ]
}
```

## 백업에 대한 Amazon DynamoDB 요금 청구 이해

이 안내서에서는 백업에 대한 DynamoDB 요금 청구가 어떻게 작동하는지 자세한 정보를 제공합니다. 전체 요금에 영향을 미치는 다양한 구성 요소를 세분화하여 명확한 설명과 실제 사례를 제공합니다.

DynamoDB는 온디맨드 백업과 시점 복구(PITR) 백업을 통해 DynamoDB 데이터를 재해 이벤트로부터 보호하며 장기 보존을 위한 데이터 아카이빙 기능을 제공합니다.

# 작동 방법

DynamoDB 온디맨드 백업은 월별로 요금이 청구됩니다. 해당 월의 특정 날짜에 백업을 수행하는 경우 백업 요금이 해당 월의 남은 일수를 기준으로 계산되어 한 번만 청구됩니다(예: 27일에 백업을 생성하면 해당 월의 남은 일수에 대해서만 27일에 한 번 요금이 적용됨).

이전에 생성한 백업을 다음 몇 달간 유지하는 경우 항상 해당 백업에 대한 한 달치 요금이 1일에 적용됩니다. 월말 이전에 백업이 제거되면 실제 사용량에 따라 요금이 조정됩니다.

예를 들어 7월 27일에 백업을 생성하고 8월까지 유지하는 경우 해당 백업에 대해 다음과 같이 요금이 청구됩니다.

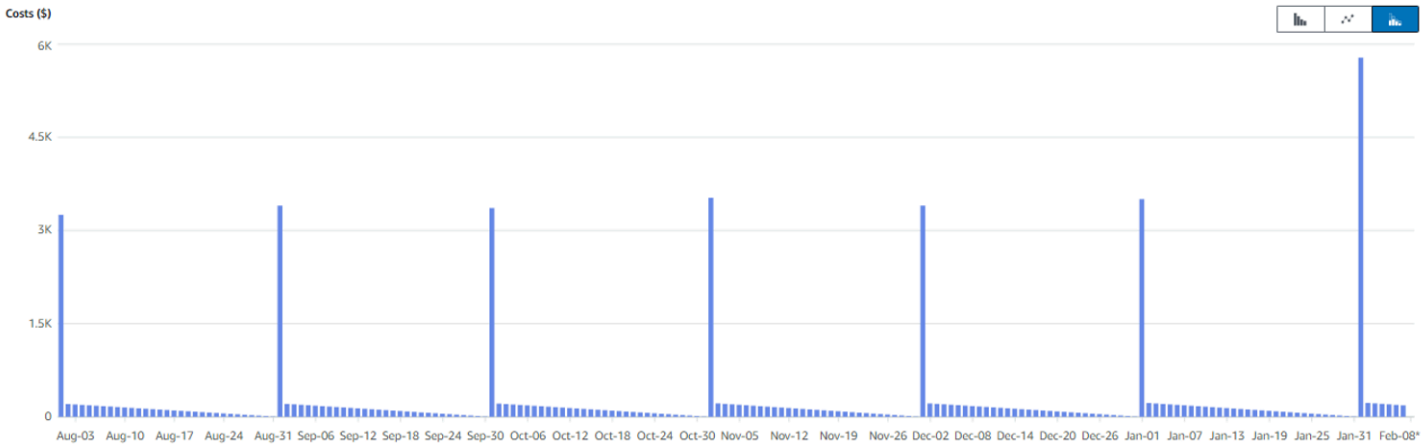
- 7월 27일에 7월의 남은 일수에 대한 요금 청구
- 8월 1일에 8월 한 달에 대한 요금 청구
- 이후 몇 달간 백업이 존재하는 경우 매월 1일에 요금 청구

DynamoDB 테이블에 대한 백업이 유지되는 경우 `DynamoDB (Region)-TimedBackupStorage-ByteHrs` 사용량 지표에 대한 요금이 매월 1일에 비정상적으로 높아 보일 수 있습니다. 또한 새로운 달이 시작될 때 이 지표를 확인하고 이전 청구 주기와 비교하면 사용량이 크게 급증하는 것처럼 나타날 수 있습니다. 이것은 설계에 따른 것입니다. 기존 DynamoDB 백업에는 매월 1일에 한 달 전체의 사용 요금이 적용됩니다. 해당 월 동안 제거된 DynamoDB 백업의 경우 실제 사용량을 반영하여 사용 요금이 비례 할당으로 계산됩니다. 따라서 해당 월 동안 요금(1일에 적용됨)이 감소하는 것을 볼 수 있습니다. 이는 보존 정책이 이월된 백업에 만료 또는 수동 삭제를 적용하기 때문입니다. 이 내용은 아래 시나리오에서 살펴볼 수 있습니다.

마찬가지로 새 백업이 생성되면 한 달 내내 더 작은 급증이 발생하며, 요금은 해당 월의 나머지 기간 동안 생성된 날에 적용됩니다.

## DynamoDB 백업 요금 청구 예시

다음은 월 초에 Cost Explorer에서 볼 수 있는 내용의 예시입니다.




2월 1일에는 전월에 비해 훨씬 더 큰 급증세를 보이는 것에 주목하세요. 왜 이런 일이 발생하는지 분석해 보겠습니다.

DynamoDB 요금 페이지에는 다음과 같은 내용이 나옵니다.

“매월 청구되는 총 백업 스토리지 크기는 DynamoDB 테이블의 모든 백업의 합계입니다. DynamoDB는 한 달 내내 온디맨드 백업 크기를 지속적으로 모니터링하여 백업 요금을 결정합니다.”

청구서에서 늘 매월 1일에 사용량이 크게 증가하는 이유가 이 때문입니다. 새로운 달에 들어오는 기존 백업에는 1일에 전체 달의 요금이 적용됩니다. 다시 말해, 새 달에 300개의 DynamoDB 백업이 들어올 경우 300개의 백업 모두에 대해 해당 월 1일에 한 달치 사용 요금이 적용됩니다.

한 달 동안 생성된 새 백업에는 생성일부터 월말까지 사용 요금이 부과됩니다.

백업이 월중에 삭제되면 어떻게 되나요?

고려해야 할 몇 가지 시나리오는 다음과 같습니다.

1. 이전 달의 백업이 당월 15일에 삭제되는 경우, 해당 백업에 대한 사용 요금은 여전히 1일에 적용되며 이전에 적용된 가정된 전체 한 달 사용 요금이 아닌 실제 사용 요금으로 조정됩니다. 아래 예제에서는 이에 대해 자세히 설명합니다.
2. 해당 월에 백업을 생성하는 경우 해당 월의 나머지 기간에 대한 사용 요금이 백업 생성일에 적용됩니다. 하지만 월말 전에 이 백업을 삭제하면 사용 요금이 백업이 활성화된 날짜만 포함하도록 조정되며 여전히 생성일에 적용됩니다.

당월 사용량이 전월에 비해 1일에 훨씬 더 높은 것처럼 보이는 이유는 무엇이며, 백업을 제거하면 어떻게 되나요?

두 부분으로 구성된 이 중요한 질문에 답하기 위해 다음 정보를 사용하여 예시 시나리오를 만들어 보겠습니다.

- 한 달 일수: 30일
- DynamoDB 백업 빈도: 일별 10회, 월별 300회
- DynamoDB 백업 보존 정책: 30일
- DynamoDB 백업당 비용: 일별 2 USD, 월별 60 USD
- 전월 1일 합계(`TimedBackupStorage-ByteHrs`, 당월 1일에 확인): 9,300 USD
- 전월 합계(`TimedBackupStorage-ByteHrs`): 18,600 USD
- 당월 1일 합계(`TimedBackupStorage-ByteHrs`, 1일에 확인): 18,000 USD
- DynamoDB 월간 사용량 변동: 없음

위의 정보에 따르면 전월에 300개의 백업이 생성되었으며 30일 동안 백업을 유지하는 정책이 있음을 알 수 있습니다. 새 달 1일에 이러한 모든 백업은 아직 복구 기간이 끝나지 않았기 때문에 계속 유지됩니다. 그러나 다음과 같이 날이 갈수록 가장 오래된 백업 세트부터 차례로 삭제됩니다.

DynamoDB 백업 삭제 표

| 새 달 | 1일차 | 2일차 | 3일차 | 4일차 | 5일차 |
|---|---|---|---|---|---|
| 이월된 전월 총 백업 수 | 300 | 290 | 280 | 270 | 260 |

- 1일에는 백업당 월별 60 USD의 요금으로 300개의 백업이 있어 총 18,000 USD의 `TimedBackupStorage-ByteHrs`가 적용됩니다. 이는 한 달 합계가 18,600 USD였던 전월과 대비됩니다.
- 2일에는 이러한 백업 중 10개가 만료되어 삭제됩니다. 이때 해당 백업에 적용되는 요금이 추정 사용량 대신 실제 사용량으로 조정됩니다. 즉, 이 10개의 백업은 이전에 1일에 600 USD(백업 10개 x 30일)의 요금이 적용되었으나 이제 20 USD(백업 10개 x 1일)로 조정됩니다.
- 다음 날에는 다음 10개 백업 블록이 만료되고 삭제되어 사용량이 30일에서 2일로 줄어들며 해당 백업 요금이 40 USD(백업 10개 x 2일)로 감소합니다.

날이 갈수록 전월보다 더 큰 폭의 급증세가 줄어들기 시작하는 것을 볼 수 있습니다. 이를 한 달 전체로 확장하면 다음과 같은 결과가 나타납니다.

DynamoDB 백업 요금(월 1일) 변화

| 10개 블록으로 300개 백업 | 1일 | 10일 | 20일 | 30일 |
|---|---|---|---|---|
| 블록 1 | 600 USD | 20 USD | 20 USD | 20 USD |
| 블록 2 | 600 USD | 40 USD | 40 USD | 40 USD |
| 블록 3 | 600 USD | 60 USD | 60 USD | 60 USD |
| 블록 4 | 600 USD | 80 USD | 80 USD | 80 USD |
| 블록 5 | 600 USD | 100 USD | 100 USD | 100 USD |
| 블록 6 | 600 USD | 120 USD | 120 USD | 120 USD |
| 블록 7 | 600 USD | 140 USD | 140 USD | 140 USD |
| 블록 8 | 600 USD | 160 USD | 160 USD | 160 USD |
| 블록 9 | 600 USD | 180 USD | 180 USD | 180 USD |
| 블록 10 | 600 USD | 600 USD | $200 | $200 |
| 블록 11 | 600 USD | 600 USD | 220 USD | 220 USD |
| 블록 12 | 600 USD | 600 USD | 240 USD | 240 USD |
| 블록 13 | 600 USD | 600 USD | 260 USD | 260 USD |
| 블록 14 | 600 USD | 600 USD | 280 USD | 280 USD |
| 블록 15 | 600 USD | 600 USD | $300 | $300 |
| 블록 16 | 600 USD | 600 USD | 320 USD | 320 USD |
| 블록 17 | 600 USD | 600 USD | 340 USD | 340 USD |
| 블록 18 | 600 USD | 600 USD | 360 USD | 360 USD |
| 블록 19 | 600 USD | 600 USD | 380 USD | 380 USD |
| 블록 20 | 600 USD | 600 USD | 600 USD | 400 USD |

| 10개 블록으로 300개 백업 | 1일 | 10일 | 20일 | 30일 |
|---|---|---|---|---|
| 블록 21 | 600 USD | 600 USD | 600 USD | 420 USD |
| 블록 22 | 600 USD | 600 USD | 600 USD | 440 USD |
| 블록 23 | 600 USD | 600 USD | 600 USD | 460 USD |
| 블록 24 | 600 USD | 600 USD | 600 USD | 480 USD |
| 블록 25 | 600 USD | 600 USD | 600 USD | 500 USD |
| 블록 26 | 600 USD | 600 USD | 600 USD | 520 USD |
| 블록 27 | 600 USD | 600 USD | 600 USD | 540 USD |
| 블록 28 | 600 USD | 600 USD | 600 USD | 560 USD |
| 블록 29 | 600 USD | 600 USD | 600 USD | 580 USD |
| 블록 30 | 600 USD | 600 USD | 600 USD | 600 USD |
| 월 1일 합계(USD) | 18,000 USD | 13,500 USD | 10,400 USD | 9,300 USD |

날마다 새로운 블록이 삭제되면 해당 블록의 사용량이 한 달 사용량이 아닌 블록이 존재한 일수에 맞춰 조정됩니다. 그 결과, 월말에는 1일에 관찰된 요금이 초기 18,000 USD에서 예상 9,300 USD로 감소합니다. 이 수치가 해당 월에 새로 생성된 백업(위와 비슷하지만 청구 테이블이 반전됨)과 합쳐져 전월의 18,600 USD와 비슷한 월별 요금이 청구됩니다.

# DynamoDB에서 테이블 복원

AWS Management Console, AWS 명령줄 인터페이스(AWS CLI) 또는 DynamoDB API를 사용하여 PITR 백업 또는 온디맨드 백업에서 DynamoDB 테이블을 복원할 수 있습니다. 복구 프로세스는 새 DynamoDB 테이블로 복원합니다.

## 특정 시점으로 복구를 사용하여 테이블 복원

`EarliestRestoreableDateTime`까지 원하는 시점으로 테이블을 복원할 수 있습니다.

> **Important**
>
> 시점 복구를 비활성화했다가 나중에 테이블에서 활성화하면 해당 테이블을 복구할 수 있는 시작 시간이 재설정됩니다. 따라서 `LatestRestorableDateTime`을 사용해서 해당 테이블을 즉시 복원하는 것만 가능합니다.

시점 복구를 사용해서 복원하는 경우, DynamoDB는 선택한 날짜와 시간(day:hour:minute:second)을 기준으로 테이블 데이터를 해당 상태로 복원합니다. 테이블의 프로비저닝된 처리량을 사용하지 않고 해당 테이블을 복원합니다. 특정 시점으로 복구를 사용한 전체 테이블 복원을 수행하거나 대상 테이블 설정을 구성할 수 있습니다. 복원된 테이블에서 다음 테이블 설정을 변경할 수 있습니다.

- 글로벌 보조 인덱스(GSI)
- 로컬 보조 인덱스(LSI)
- 결제 모드
- 프로비저닝된 읽기 및 쓰기 용량
- 암호화 설정

> **Important**
>
> 전체 테이블 복원을 수행할 때 대상 테이블은 백업이 요청되었을 때 원본 테이블과 동일하게 프로비저닝된 읽기 용량 단위 및 쓰기 용량 단위로 설정됩니다. 예를 들어 테이블의 프로비저닝된 처리량이 최근에 읽기 용량 단위 50 및 쓰기 용량 단위 50으로 낮춰졌다고 가정합니다. 그러면 테이블의 상태를 3주 전으로 복원합니다. 당시에 프로비저닝된 처리량은 읽기 용량 단위 100 및 쓰기 용량 단위 100으로 설정되었습니다. 이 경우 DynamoDB는 해당 시점에서 프로비저닝된 처리량(throughput)(100 읽기 용량 단위 및 100 쓰기 용량 단위)으로 테이블 데이터를 해당 시점으로 복원합니다.

또한 복원된 테이블이 원본 테이블이 속하는 리전과 다른 리전에 생성되도록 AWS 리전 간에 DynamoDB 테이블 데이터를 복원할 수 있습니다. AWS 커머셜 리전, AWS 중국 리전 및 AWS GovCloud (US) 간에 교차 리전 복원을 수행할 수 있습니다. 소스 리전에서 전송한 데이터와 대상 리전의 새 테이블로 복원하는 데 사용한 리소스에 대해서만 요금을 지불하면 됩니다.

> **Note**
>
> 소스 또는 대상 리전이 아시아 태평양(홍콩) 또는 중동(바레인)인 경우 교차 리전 복원이 지원되지 않습니다.

복원된 테이블에 일부 또는 전체 인덱스가 생성되지 않도록 제외하는 경우 복원이 보다 빠르고 비용 효율적일 수 있습니다. 복원된 테이블에서 다음을 수동으로 설정해야 합니다.

- Auto Scaling 정책
- AWS Identity and Access Management 정책
- Amazon CloudWatch Events 지표 및 경보
- Tags
- 스트림 설정
- Time To Live(TTL) 설정
- 특정 시점으로 복구 설정

테이블을 복원하는 데 걸리는 시간은 여러 요소에 따라 달라지며 테이블의 크기와 항상 상관 관계가 있는 것은 아닙니다.

## DynamoDB 테이블을 특정 시점으로 복원

Amazon DynamoDB의 PITR(특정 시점으로 복구)을 통해 DynamoDB 테이블 데이터를 지속적으로 백업할 수 있습니다. DynamoDB 콘솔이나 AWS Command Line Interface(AWS CLI)를 사용하여 테이블을 특정 시점으로 복원할 수 있습니다. 특정 시점으로 복구 프로세스는 새 테이블로 복원합니다.

AWS CLI를 사용하려면 먼저 구성을 해야 합니다. 자세한 내용은 DynamoDB 액세스 섹션을 참조하세요.

주제

- DynamoDB 테이블을 특정 시점으로 복원(콘솔)
- 테이블을 특정 시점으로 복원(AWS CLI)

# DynamoDB 테이블을 특정 시점으로 복원(콘솔)

다음은 DynamoDB 콘솔을 사용하여 `Music`이라는 기존 테이블을 특정 시점으로 복원하는 방법을 보여 주는 예입니다.

> **Note**
>
> 이 절차에서는 특정 시점으로 복구를 활성화했다고 가정합니다. `Music` 테이블에 대해 활성화하려면 백업(Backups) 탭의 특정 시점으로 복구(PITR)(Point-in-time recovery (PITR)) 섹션에서 편집(Edit)을 선택한 다음 특정 시점으로 복구 활성화(Enable point-in-time-recovery) 옆의 확인란을 선택합니다.

테이블을 특정 시점으로 복원하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 테이블을 선택합니다.
3. 테이블 목록에서 `Music` 테이블을 선택합니다.
4. `Music` 테이블에 있는 백업(Backups) 탭의 특정 시점으로 복구(PITR)(Point-in-time recovery (PITR)) 섹션에서 복원(Restore)을 선택합니다.
5. 새 테이블 이름에 **`MusicMinutesAgo`**를 입력합니다.

   > **Note**
   >
   > 테이블을 동일한 AWS 리전 또는 소스 테이블이 속하는 리전과 다른 리전에 복원할 수 있습니다. 복원된 테이블에 보조 인덱스가 생성되지 않도록 제외할 수도 있습니다. 또한 다른 암호화 모드를 지정할 수 있습니다.

6. 복원 가능한 시간을 확인하려면 복원 날짜 및 시간을 가장 빠른 시간(Earliest)로 설정합니다. 그런 다음 복원(Restore)을 선택하여 복원 프로세스를 시작합니다.

   복원 중인 테이블은 상태가 복원 중으로 표시됩니다. 복원 프로세스가 완료되면 `MusicMinutesAgo` 테이블의 상태가 Active로 변경됩니다.

## 테이블을 특정 시점으로 복원(AWS CLI)

다음은 AWS CLI를 사용하여 Music이라는 기존 테이블을 특정 시점으로 복원하는 방법을 보여 주는 절차입니다.

> **Note**
>
> 이 절차에서는 특정 시점으로 복구를 활성화했다고 가정합니다. 다음 명령을 실행하여 Music 테이블에 대해 이 기능을 활성화하세요.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name Music \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

테이블을 특정 시점으로 복원하려면

1. describe-continuous-backups 명령으로 Music 테이블에 대해 특정 시점으로 복구가 활성화되었는지 확인합니다.

   ```
   aws dynamodb describe-continuous-backups \
       --table-name Music
   ```

   연속 백업(테이블 생성 시 자동으로 활성화) 및 특정 시점으로 복구가 활성화되었습니다.

   ```
   {
       "ContinuousBackupsDescription": {
           "PointInTimeRecoveryDescription": {
               "PointInTimeRecoveryStatus": "ENABLED",
               "EarliestRestorableDateTime": 1519257118.0,
               "LatestRestorableDateTime": 1520018653.01
           },
           "ContinuousBackupsStatus": "ENABLED"
       }
   }
   ```

2. 테이블을 특정 시점으로 복원합니다. 이 경우 Music 테이블은 동일한 AWS 리전의 LatestRestorableDateTime(~5분 전)으로 복원됩니다.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time
```

> **Note**
>
> 특정 시점으로 복원할 수도 있습니다. 그러려면 `--restore-date-time` 인수를 사용하여 명령을 실행하고 타임스탬프를 지정합니다. 구성된 복구 기간 내의 원하는 시점을 지정할 수 있으며, 1~35일 사이의 값으로 설정할 수 있습니다. 예를 들어, 다음 명령은 테이블을 `EarliestRestorableDateTime`으로 복원합니다.
>
> ```
> aws dynamodb restore-table-to-point-in-time \
>     --source-table-name Music \
>     --target-table-name MusicEarliestRestorableDateTime \
>     --no-use-latest-restorable-time \
>     --restore-date-time 1519257118.0
> ```
>
> 원한다면 특정 시점으로 복원할 때 `--no-use-latest-restorable-time` 인수를 지정할 수 있습니다.

3. 사용자 지정 테이블 설정을 통해 특정 시점으로 테이블을 복원합니다. 이 경우 `Music` 테이블은 `LatestRestorableDateTime`(~5분 전)으로 복원됩니다.

   다음과 같이 복원된 테이블에 다른 암호화 모드를 지정할 수 있습니다.

   > **Note**
   >
   > `sse-specification-override` 파라미터는 `CreateTable` 명령에 사용된 `sse-specification-override` 파라미터와 동일한 값을 사용합니다. 자세한 내용은 DynamoDB의 암호화된 테이블 관리을 참조하십시오.

   ```
   aws dynamodb restore-table-to-point-in-time \
       --source-table-name Music \
       --target-table-name MusicMinutesAgo \
       --use-latest-restorable-time \
   ```

```
    --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

테이블을 소스 테이블이 속하는 리전과 다른 AWS 리전에 복원할 수 있습니다.

> **Note**
>
> - `sse-specification-override` 파라미터는 교차 리전 복원에는 필수지만 소스 테이블과 동일한 리전으로 복원하는 경우에는 선택 사항입니다.
> - 교차 리전 복원에는 `source-table-arn` 파라미터를 제공해야 합니다.
> - 명령줄에서 교차 리전 복원을 수행할 때는 기본 AWS 리전을 원하는 대상 리전으로 설정해야 합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 명령줄 옵션을 참조하세요.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

복원된 테이블에 대한 결제 모드와 프로비저닝된 처리량을 재정의할 수 있습니다.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --billing-mode-override PAY_PER_REQUEST
```

복원된 테이블에 일부 또는 전체 보조 인덱스가 생성되지 않도록 제외할 수 있습니다.

> **Note**
>
> 복원된 새 테이블에 일부 또는 전체 보조 인덱스가 생성되지 않도록 제외하는 경우 복원이 보다 빠르고 비용 효율적일 수 있습니다.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --global-secondary-index-override '[]'
```

다른 재정의 조합을 사용할 수 있습니다. 예를 들어, 다음과 같이 단일 글로벌 보조 인덱스를 사용하는 동시에 프로비저닝된 처리량을 변경할 수 있습니다.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --billing-mode-override PROVISIONED \
    --provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100
 \
    --global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEY
 \
    --sse-specification-override Enabled=true,SSEType=KMS \
    --use-latest-restorable-time
```

복원을 확인하려면 `describe-table` 명령을 사용하여 `MusicEarliestRestorableDateTime` 테이블을 명시합니다.

```
aws dynamodb describe-table --table-name MusicEarliestRestorableDateTime
```

복원 중인 테이블은 상태가 생성 중이 되고, 복원 진행 중에 true가 표시됩니다. 복원 프로세스가 완료되면 `MusicEarliestRestorableDateTime` 테이블의 상태가 Active로 변경됩니다.

> **Important**
>
> 복원을 진행하는 동안 IAM 엔터티(예: 사용자, 그룹 또는 역할)에 복원 수행 권한을 부여하는 AWS Identity and Access Management(IAM) 정책을 수정하거나 삭제하지 마세요. 그러면 예기치 않은 동작이 발생할 수 있습니다. 예를 들어 테이블 복원 중에 테이블에 대한 쓰기 권한을 삭제한다고 가정해 보겠습니다. 기본 `RestoreTableToPointInTime` 작업에서 복원된 데이터를 해당 테이블에 쓸 수 없게 됩니다. 대상 복원 테이블에 액세스하기 위한 소스 IP 제한 관련 IAM 정책은 유사한 문제의 이유가 될 수 있습니다.

복원 작업이 완료되어야 권한을 수정하거나 삭제할 수 있습니다.

# DynamoDB에서 AWS Backup 사용

Amazon DynamoDB는 AWS Backup의 향상된 백업 기능을 통해 규정 준수 및 비즈니스 연속성 요구 사항을 충족할 수 있도록 지원합니다. AWS Backup은 클라우드 및 온프레미스에서 AWS 서비스 전반에 걸쳐 백업을 쉽게 중앙 집중화하고 자동화할 수 있는 완전관리형 데이터 보호 서비스입니다. 이 서비스를 사용하여 백업 정책을 구성하고 AWS 리소스에 대한 활동을 한 곳에서 모니터링할 수 있습니다. AWS Backup을 사용하려면 옵트인해야 합니다. 옵트인 선택 사항은 특정 계정 및 AWS 리전에 적용되므로 동일한 계정을 사용하여 여러 리전에 옵트인해야 할 수도 있습니다. 자세한 내용은 AWS 백업 개발자 안내서를 참조하세요.

Amazon DynamoDB는 기본적으로 AWS Backup과 통합됩니다. AWS Backup을 사용하여 DynamoDB 온디맨드 백업을 자동으로 예약, 복사, 태그 지정하고 수명 주기를 관리할 수 있습니다. DynamoDB 콘솔에서 이러한 백업을 계속 확인하고 복원할 수 있습니다. DynamoDB 콘솔, API 및 AWS 명령줄 인터페이스(AWSCLI)를 사용하여 DynamoDB 테이블에 대한 자동 백업을 활성화할 수 있습니다.

> **Note**
>
> DynamoDB를 통해 수행된 모든 백업은 변경되지 않습니다. 현재 DynamoDB 워크플로를 통해 계속해서 백업을 생성할 수 있습니다.

AWS Backup을 통해 제공되는 향상된 백업 기능은 다음과 같습니다.

예약된 백업 - 백업 계획을 사용하여 정기적으로 예약된 DynamoDB 테이블 백업을 설정할 수 있습니다.

교차 계정 및 교차 리전 복사 - 백업을 다른 AWS 리전 또는 계정의 다른 백업 볼트에 자동으로 복사하여 데이터 보호 요구 사항을 지원할 수 있습니다.

콜드 스토리지 계층화 - 백업을 삭제하거나 콜드 스토리지로 전환하는 수명 주기 규칙을 구현하도록 백업을 구성할 수 있습니다. 이렇게 하면 백업 비용을 최적화할 수 있습니다.

태그 - 결제 및 비용 할당을 위해 백업에 자동으로 태그를 지정할 수 있습니다.

암호화 - AWS Backup을 통해 관리되는 DynamoDB 온디맨드 백업은 이제 AWS Backup 볼트에 저장됩니다. 이렇게 하면 DynamoDB 테이블 암호화 키와 독립적인 AWS KMS key를 사용하여 백업을 암호화하고 보호할 수 있습니다.

백업 감사 - AWS Backup Audit Manager를 사용하여 AWS Backup 정책의 준수를 감사하고 정의한 제어를 아직 준수하지 않은 백업 작업 및 리소스를 찾을 수 있습니다. 또한 백업 거버넌스를 위해 일일 및 온디맨드 보고서의 감사 추적을 자동으로 생성하는 데 사용할 수 있습니다.

WORM 모델을 사용하여 백업 보안 - AWS Backup 볼트 잠금을 사용하여 백업에 대해 write-once-read-many(WORM) 설정을 활성화할 수 있습니다. AWS Backup 볼트 잠금을 사용하면 우발적이거나 악의적인 삭제 작업, 백업 복구 기간 변경 및 수명 주기 설정 업데이트로부터 백업을 보호하는 추가 방어 계층을 추가할 수 있습니다. 자세한 내용은 AWS Backup 볼트 잠금 섹션을 참조하세요.

이러한 향상된 백업 기능은 모든 AWS 리전에서 사용할 수 있습니다. 이러한 기능에 대한 자세한 내용은 AWS Backup 개발자 안내서를 참조하세요.

주제


- AWS Backup을 사용한 DynamoDB 테이블 백업 및 복원: 작동 방식
- AWS Backup을 사용하여 DynamoDB 테이블의 백업 생성
- AWS Backup을 사용하여 DynamoDB 테이블의 백업 복사
- AWS Backup에서 DynamoDB 테이블의 백업 복원
- AWS Backup을 사용하여 DynamoDB 테이블의 백업 삭제
- AWS Backup와 DynamoDB에서 관리하는 온디맨드 백업 간의 사용 노트 비교


## AWS Backup을 사용한 DynamoDB 테이블 백업 및 복원: 작동 방식

온디맨드 백업 기능을 사용하여 Amazon DynamoDB 테이블의 전체 백업을 생성할 수 있습니다. 이 섹션에서는 백업 및 복원 프로세스를 수행하는 중에 발생하는 상황에 대한 개요를 제공합니다.

### 백업

AWS Backup을 통해 온디맨드 백업을 생성하는 경우, 해당 요청의 타임 마커가 카탈로그화됩니다. 백업은 마지막 전체 테이블 스냅샷에 대한 요청 시간까지 모든 변경 내용을 적용하여 비동기식으로 생성됩니다.

온디맨드 백업을 생성할 때마다 전체 테이블 데이터가 백업됩니다. 수행할 수 있는 온디맨드 백업 수에는 제한이 없습니다.

> **Note**
>
> DynamoDB 백업과 달리 AWS Backup를 통한 백업은 즉각적으로 이루어지지 않습니다.

백업이 진행 중인 동안에는 다음 작업을 수행할 수 없습니다.

- 백업 작업을 일시 중지하거나 취소합니다.
- 백업의 원본 테이블을 삭제합니다.
- 테이블에 대한 백업이 진행 중인 경우 해당 테이블에서 백업을 비활성화합니다.

AWS Backup에서는 자동화된 백업 일정, 보존 관리 및 수명 주기 관리 기능을 제공합니다. 따라서 사용자 지정 스크립트와 수동 프로세스가 필요하지 않습니다. AWS Backup은 백업을 실행하고 백업이 만료되면 백업을 삭제합니다. 자세한 내용은 개발자 안내서AWS Backup를 참조하세요.

콘솔을 사용하는 경우 AWS Backup을 사용하여 생성된 모든 백업은 Backup type(백업 유형)이 `AWS_BACKUP`로 설정되어 Backups(백업) 탭에 나열됩니다.

> **Note**
>
> Backup type(백업 유형)이 `AWS_BACKUP`로 표시된 백업은 DynamoDB 콘솔을 사용하여 삭제할 수 없습니다. 이러한 백업을 관리하려면 AWS Backup 콘솔을 사용합니다.

백업을 수행하는 방법은 DynamoDB 테이블 백업 단원을 참조하십시오.

### 복원

테이블의 프로비저닝된 처리량을 사용하지 않고 해당 테이블을 복원합니다. DynamoDB 백업에서 전체 테이블 복원을 수행하거나 대상 테이블 설정을 구성할 수 있습니다. 복원을 수행할 때 다음 테이블 설정을 변경할 수 있습니다.

- 글로벌 보조 인덱스(GSI)
- 로컬 보조 인덱스(LSI)
- 결제 모드
- 프로비저닝된 읽기 및 쓰기 용량
- 암호화 설정

> **Important**
>
> 전체 테이블 복원을 수행할 때 대상 테이블은 백업이 요청되었을 때 원본 테이블과 동일하게 프로비저닝된 읽기 용량 단위 및 쓰기 용량 단위로 설정됩니다. 복원 프로세스는 로컬 보조 인덱스와 글로벌 보조 인덱스도 복원합니다.

DynamoDB 테이블 데이터의 백업을 다른 AWS 리전에 복사한 다음 새 리전에 복원할 수 있습니다. AWS 상용 리전, AWS 중국 리전 및 AWS GovCloud(미국) 리전 간에 백업을 복사한 다음 복원할 수 있습니다. 소스 리전에서 복사한 데이터와 대상 리전의 새 테이블에 복원한 데이터에 대해서만 요금을 지불합니다.

AWS Backup은 모든 원본 인덱스가 있는 테이블을 복원합니다.

복원된 테이블에서 다음을 수동으로 설정해야 합니다.

- Auto Scaling 정책
- AWS Identity and Access Management (IAM) 정책
- Amazon CloudWatch 지표 및 경보
- Tags
- 스트림 설정
- Time To Live(TTL) 설정
- 삭제 방지 설정
- 시점 복구(PITR) 설정

전체 테이블 데이터만 백업에서 새 테이블로 복원할 수 있습니다. 복원된 테이블은 활성 상태가 된 이후에만 쓸 수 있습니다.

> **Note**
>
> AWS Backup 복원은 비파괴적입니다. 복원 작업 중에는 기존 테이블을 덮어쓸 수 없습니다.

서비스 지표에는 고객 테이블 복원의 95%가 1시간 이내에 완료된다고 표시됩니다. 그러나 복원 시간은 테이블 구성(예: 테이블 크기 및 기본 파티션 수) 및 기타 관련 변수와 직접 관련이 있습니다. 재해 복

구를 계획할 때 가장 좋은 방법은 평균 복원 완료 시간을 정기적으로 기록하고 이러한 시간이 전체 복구 시간 목표에 어떤 영향을 미치는지 설정하는 것입니다.

복원을 수행하는 방법은 백업에서 DynamoDB 테이블 복원 단원을 참조하십시오.

액세스 제어에 IAM 정책을 사용할 수 있습니다. 자세한 내용은 DynamoDB 백업 및 복원에 IAM 사용 섹션을 참조하세요.

모든 백업 및 복원 콘솔 작업과 API 작업이 로깅, 연속 모니터링 및 감사를 위해 AWS CloudTrail에 캡처되고 기록됩니다.

## AWS Backup을 사용하여 DynamoDB 테이블의 백업 생성

이 섹션에서는 AWS Backup을 켜서 온디맨드 및 예약한 백업을 DynamoDB 테이블에서 생성하는 방법을 설명합니다.

주제

- AWS Backup 기능 켜기
- 온디맨드 백업
- 예약한 백업

### AWS Backup 기능 켜기

AWS Backup을 켜서 DynamoDB와 함께 사용해야 합니다.

AWS Backup을 켜려면 다음 단계를 수행하세요.

1. AWS 관리 콘솔에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.
3. 백업 설정 창에서 켜기를 선택합니다.
4. 확인 화면이 표시됩니다. 기능 켜기를 선택합니다.

이제 DynamoDB 테이블에 AWS Backup 기능을 사용할 수 있습니다.

AWS Backup 기능을 켠 후에 끄도록 선택하는 경우 다음 단계를 수행하세요.

1. AWS 관리 콘솔에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.

2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.
3. 백업 설정 창에서 끄기를 선택합니다.
4. 확인 화면이 표시됩니다. 기능 끄기를 선택합니다.

AWS Backup 기능을 켜거나 끌 수 없는 경우 AWS 관리자가 해당 작업을 수행해야 할 수 있습니다.

### 온디맨드 백업

DynamoDB 테이블의 온디맨드 백업을 생성하려면 다음 단계를 수행하세요.

1. AWS 관리 콘솔에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.
3. [백업 생성]을 선택합니다.
4. 표시된 드롭다운 메뉴에서 온디맨드 백업 생성(Create an on-demand backup)을 선택합니다.
5. 웜 스토리지 및 기타 기본 기능을 사용하여 AWS Backup에서 관리하는 백업을 생성하려면 기본 설정(Default Settings)을 선택합니다. 콜드 스토리지로 전환할 수 있는 백업을 생성하거나 AWS Backup 대신에 DynamoDB 기능을 사용하여 백업을 생성하려면 설정 사용자 지정(Customize settings)을 선택합니다.

   대신 이전 DynamoDB 기능을 사용하여 이 백업을 생성하려면 설정 사용자 지정(Customize settings)을 선택한 다음 DynamoDB를 사용하여 백업(Backup with DynamoDB)을 선택합니다.
6. 설정을 완료한 후 백업 생성(Create backup)을 선택합니다.

### 예약한 백업

백업을 예약하려면 다음 단계를 수행하세요.

1. AWS 관리 콘솔에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.
3. 표시되는 드롭다운 메뉴에서 AWS Backup로 백업 예약(Schedule backups with)을 선택합니다.
4. 그러면 AWS Backup으로 이동하여 백업 계획을 생성할 수 있습니다.

# AWS Backup을 사용하여 DynamoDB 테이블의 백업 복사

현재 백업의 복사본을 만들 수 있습니다. 요청 시 백업을 여러 AWS 계정 또는 AWS 리전으로 복사하거나 예약된 백업 계획의 일부로 자동 복사할 수 있습니다. Amazon DynamoDB Encryption Client에 대한 교차 계정 및 리전 간 복사본의 시퀀스를 자동화할 수도 있습니다.

교차 리전 복제는 프로덕션 데이터로부터 최소 거리에 백업을 저장해야 하는 비즈니스 연속성 또는 규정 준수 요구 사항이 있는 경우 특히 유용합니다.

교차 계정 백업은 백업을 운영 또는 보안상의 이유로 조직의 하나 이상의 AWS 계정에 안전하게 복사하는 데 유용합니다. 원본 백업이 실수로 삭제된 경우 대상 계정에서 소스 계정으로 백업을 복사한 다음 복원을 시작할 수 있습니다. 이렇게 하려면 먼저 Organizations 서비스에서 동일한 조직에 속하는 2개의 계정이 있어야 합니다.

별도로 지정하지 않는 한 복사본은 소스 백업의 구성을 상속합니다. 단, 새 복사본이 '절대' 만료되지 않음으로 지정하는 경우 예외입니다. 이 설정을 사용하면 새 복사본은 여전히 원본 만료 날짜를 상속합니다. 새 백업 복사본을 영구적으로 사용하려면 소스 백업이 만료되지 않도록 설정하거나 새 복사본을 만든 후 100년 후에 만료되도록 지정합니다.

> **Note**
>
> 다른 계정으로 복사하는 경우 먼저 해당 계정의 권한이 있어야 합니다.

백업을 복사하려면 다음을 수행합니다.

1. AWS 관리 콘솔에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.
3. 복사할 백업 옆에 있는 확인란을 선택합니다.
   - 복사할 백업이 회색으로 표시되면 AWS Backup의 고급 기능을 사용 설정해야 합니다. 그런 다음 새 백업을 생성합니다. 이제 이 새 백업을 다른 리전 및 계정에 복사할 수 있으며 앞으로 다른 새 백업을 복사할 수 있습니다.
4. 복사를 선택합니다.
5. 백업을 다른 계정 또는 리전에 복사하려면 복구 지점을 다른 대상에 복사(Copy the recovery point to another destination) 옆의 확인란을 선택합니다. 그런 다음 계정의 다른 리전으로 복사할지 아니면 다른 리전의 다른 계정으로 복사할지 선택합니다.

> **Note**
>
> 백업을 다른 리전 또는 계정으로 복원하려면 먼저 해당 리전 또는 계정에 백업을 복사해야 합니다.

6. 파일을 복사하려는 볼트를 선택합니다. 또한 원하는 경우 새 백업 볼트를 생성할 수도 있습니다.
7. 백업 복사(Copy backup)를 선택합니다.

## AWS Backup에서 DynamoDB 테이블의 백업 복원

이 섹션에서는 AWS Backup에서 DynamoDB 테이블의 백업을 복원하는 방법을 설명합니다.

주제

- AWS Backup에서 DynamoDB 테이블 복원
- DynamoDB 테이블을 다른 리전 또는 계정으로 복원

### AWS Backup에서 DynamoDB 테이블 복원

AWS Backup에서 DynamoDB 테이블을 복원하려면 다음 단계를 따르세요.

1. AWS 관리 콘솔에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 테이블을 선택합니다.
3. 백업 탭을 선택합니다.
4. 복원할 이전 백업 옆의 확인란을 선택합니다.
5. 복원을 선택합니다. 백업에서 테이블 복원(Restore table from backup) 화면으로 이동합니다.
6. 새로 복원된 테이블의 이름, 이 새 테이블에 포함할 암호화, 복원을 암호화할 키 및 기타 옵션을 입력합니다.
7. 작업을 마쳤으면 복원(Restore)을 선택합니다.

### DynamoDB 테이블을 다른 리전 또는 계정으로 복원

DynamoDB 테이블을 다른 리전 또는 계정으로 복원하려면 먼저 백업을 새 리전 또는 계정에 복사해야 합니다. 다른 계정으로 복사하려면 먼저 해당 계정에서 권한을 부여해야 합니다. DynamoDB 백업을 새 리전 또는 계정에 복사한 후에는 이전 섹션의 프로세스를 통해 복원할 수 있습니다.

# AWS Backup을 사용하여 DynamoDB 테이블의 백업 삭제

이 섹션에서는 AWS Backup을 사용하여 DynamoDB 테이블의 백업을 삭제하는 방법을 설명합니다.

AWS 백업 기능 통해 생성된 DynamoDB 백업은 AWS 백업 볼트에 저장됩니다.

이러한 백업을 삭제하려면 다음을 수행하세요.

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.
3. 다음 화면에서 AWS 백업 계속(Continue to Backup)을 선택합니다.

   이제 AWS Backup 콘솔으로 이동하게 됩니다. AWS Backup 콘솔에서 백업을 삭제하는 방법에 대한 자세한 내용은 백업 삭제를 참조하세요.

   AWS Backup에 대한 자세한 내용은 AWS 권장 가이드의 AWS Backup을 사용하여 백업 및 복구를 참조하세요.

# AWS Backup와 DynamoDB에서 관리하는 온디맨드 백업 간의 사용 노트 비교

이 섹션에서는 AWS Backup으로 관리되는 온디맨드 백업과 DynamoDB에서 관리되는 온디맨드 백업의 기술적 차이점을 설명합니다.

AWS Backup에는 DynamoDB와 다른 워크플로와 동작이 있습니다. 다음이 포함됩니다.

암호화 - AWS Backup 계획을 통해 생성된 백업은 AWS Backup 서비스에서 관리하는 키로 암호화된 볼트에 저장됩니다. 볼트에는 추가 보안을 위한 액세스 제어 정책이 있습니다.

백업 ARN - AWS Backup에서 생성된 백업 파일에는 이제 AWS Backup ARN이 부여되며, 이는 사용자 권한 모델에 영향을 줄 수 있습니다. 백업 리소스 이름(ARN)이 `arn:aws:dynamodb`에서 `arn:aws:backup`으로 변경됩니다.

백업 삭제 - AWS Backup을 통해 생성된 백업은 AWS Backup 볼트에서만 삭제할 수 있습니다. DynamoDB 콘솔에서 가져온 AWS Backup 파일은 삭제할 수 없습니다.

백업 프로세스 - DynamoDB 백업과 달리 AWS Backup을 통해 생성된 백업은 즉각적으로 이루어지지 않습니다.

결제 - AWS Backup 기능을 사용하는 DynamoDB 테이블 백업은 AWS Backup에서 청구됩니다.

IAM 역할 - IAM 역할을 통해 액세스를 관리하는 경우 다음과 같은 새로운 권한으로 새 IAM 역할을 구성해야 합니다.

```
"dynamodb:StartAwsBackupJob",
"dynamodb:RestoreTableFromAwsBackup"
```

`dynamodb:StartAwsBackupJob`은 AWS Backup 기능을 사용하는 백업에 필요하고 `dynamodb:RestoreTableFromAwsBackup`은 AWS Backup 기능으로 만든 백업에서 복원하는 데 필요합니다.

전체 IAM 정책에서 이러한 권한을 보려면 IAM 사용의 예제 8을 참조하세요.

# AWS SDK를 사용한 DynamoDB용 코드 예제

다음 코드 예제에서는 DynamoDB를 AWS 소프트웨어 개발 키트(SDK)와 함께 사용하는 방법을 보여 줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

AWS 커뮤니티 기여는 AWS의 여러 팀에서 생성하고 유지 관리하는 예입니다. 피드백을 제공하려면 연결된 리포지토리에 제공된 메커니즘을 사용합니다.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

시작

## Hello DynamoDB

다음 코드 예제에서는 DynamoDB를 사용하여 시작하는 방법을 보여 줍니다.

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;
```

```
namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables =
 dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
```

```
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListTables를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

CMakeLists.txt CMake 파일의 코드입니다.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS dynamodb)

# Set this project's name.
project("hello_dynamodb")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)
```

```
# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_dynamodb.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_dynamodb.cpp 소스 파일의 코드입니다.

```
#include <aws/core/Aws.h>
#include <aws/dynamodb/DynamoDBClient.h>
#include <aws/dynamodb/model/ListTablesRequest.h>
#include <iostream>

/*
```

```
 *  A "Hello DynamoDB" starter application which initializes an Amazon DynamoDB
 (DynamoDB) client and lists the
 *  DynamoDB tables.
 *
 *  main function
 *
 *  Usage: 'hello_dynamodb'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.

    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::DynamoDB::DynamoDBClient dynamodbClient(clientConfig);
        Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
        listTablesRequest.SetLimit(50);
        do {
            const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamodbClient.ListTables(
                    listTablesRequest);
            if (!outcome.IsSuccess()) {
                std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
                result = 1;
                break;
            }

            for (const auto &tableName: outcome.GetResult().GetTableNames()) {
                std::cout << tableName << std::endl;
            }

            listTablesRequest.SetExclusiveStartTableName(
                    outcome.GetResult().GetLastEvaluatedTableName());

        } while (!listTablesRequest.GetExclusiveStartTableName().empty());
```

```
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 ListTables를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
```

```
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
```

```
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 ListTables를 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK for JavaScript에서 DynamoDB를 사용하는 방법에 대한 자세한 내용은 JavaScript를 사용한 DynamoDB 프로그래밍을 참조하세요.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 ListTables를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import boto3

# Create a DynamoDB client using the default credentials and region
dynamodb = boto3.client("dynamodb")

# Initialize a paginator for the list_tables operation
paginator = dynamodb.get_paginator("list_tables")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(Limit=10)

# List the tables in the current AWS account
print("Here are the DynamoDB tables in your account:")

# Use pagination to list all tables
table_names = []

for page in page_iterator:
    for table_name in page.get("TableNames", []):
        print(f"- {table_name}")
        table_names.append(table_name)

if not table_names:
    print("You don't have any DynamoDB tables in your account.")
else:
    print(f"\nFound {len(table_names)} tables.")
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 ListTables를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
require 'aws-sdk-dynamodb'
require 'logger'

# DynamoDBManager is a class responsible for managing DynamoDB operations
# such as listing all tables in the current AWS account.
class DynamoDBManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all DynamoDB tables in the current AWS account.
  def list_tables
    @logger.info('Here are the DynamoDB tables in your account:')

    paginator = @client.list_tables(limit: 10)
    table_names = []

    paginator.each_page do |page|
      page.table_names.each do |table_name|
        @logger.info("- #{table_name}")
        table_names << table_name
      end
    end

    if table_names.empty?
      @logger.info("You don't have any DynamoDB tables in your account.")
    else
      @logger.info("\nFound #{table_names.length} tables.")
    end
  end
```

```
end

if $PROGRAM_NAME == __FILE__
  dynamodb_client = Aws::DynamoDB::Client.new
  manager = DynamoDBManager.new(dynamodb_client)
  manager.list_tables
end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 ListTables를 참조하세요.

코드 예제

- AWS SDK를 사용한 DynamoDB용 기본 예제
  - Hello DynamoDB
  - AWS SDK와 함께 사용되는 DynamoDB의 기본 기능 알아보기
  - AWS SDK를 사용한 DynamoDB에 대한 작업
    - AWS SDK와 함께 BatchExecuteStatement사용
    - AWS SDK 또는 CLI와 함께 BatchGetItem 사용
    - AWS SDK 또는 CLI와 함께 BatchWriteItem 사용
    - AWS SDK 또는 CLI와 함께 CreateTable 사용
    - AWS SDK 또는 CLI와 함께 DeleteItem 사용
    - AWS SDK 또는 CLI와 함께 DeleteTable 사용
    - AWS SDK 또는 CLI와 함께 DescribeTable 사용
    - AWS SDK 또는 CLI와 함께 DescribeTimeToLive 사용
    - AWS SDK와 함께 ExecuteStatement사용
    - AWS SDK 또는 CLI와 함께 GetItem 사용
    - AWS SDK 또는 CLI와 함께 ListTables 사용
    - AWS SDK 또는 CLI와 함께 PutItem 사용
    - AWS SDK 또는 CLI와 함께 Query 사용
    - AWS SDK 또는 CLI와 함께 Scan 사용
    - AWS SDK 또는 CLI와 함께 UpdateItem 사용
    - AWS SDK 또는 CLI와 함께 UpdateTable 사용
    - AWS SDK 또는 CLI와 함께 UpdateTimeToLive 사용

- AWS SDK를 사용한 DynamoDB 관련 시나리오
  - AWS SDK를 사용하여 DAX로 DynamoDB 읽기 가속화
  - AWS Command Line Interface v2를 사용한 고급 DynamoDB 글로벌 보조 인덱스 시나리오 작업
  - DynamoDB 테이블에 데이터를 제출하기 위한 애플리케이션 구축
  - AWS SDK를 사용하여 DynamoDB에서 단일 속성과 여러 값 비교
  - AWS SDK를 사용하여 TTL로 DynamoDB 항목을 조건부로 업데이트
  - AWS SDK를 사용하여 로컬 DynamoDB 인스턴스에 연결
  - AWS SDK를 사용하여 DynamoDB에서 표현식 연산자 수 계산
  - COVID-19 데이터를 추적하는 API Gateway REST API 생성
  - Step Functions를 사용하여 메신저 애플리케이션 생성
  - 사용자가 레이블을 사용하여 사진을 관리할 수 있는 사진 자산 관리 애플리케이션 만들기
  - AWS SDK를 사용하여 글로벌 보조 인덱스가 포함된 DynamoDB 테이블 만들기
  - AWS SDK를 사용하여 원 처리량 설정으로 DynamoDB 테이블 만들기
  - DynamoDB 데이터를 추적하는 웹 애플리케이션 만들기
  - API Gateway를 사용하여 WebSocket 채팅 애플리케이션 생성
  - AWS SDK를 사용하여 TTL이 포함된 DynamoDB 항목 생성
  - AWS SDK를 사용하여 다중 리전 강력한 일관성을 갖춘 DynamoDB 글로벌 테이블을 만들고 관리하는 방법
  - AWS SDK를 사용하여 MREC를 보여주는 DynamoDB 글로벌 테이블 만들기 및 관리
  - AWS SDK와 함께 PartiQL DELETE 문을 사용하여 DynamoDB 데이터 삭제
  - AWS SDK를 사용하여 Amazon Rekognition을 통해 이미지에서 PPE 감지
  - AWS SDK와 함께 PartiQL INSERT 문을 사용하여 DynamoDB 데이터 삽입
  - 브라우저에서 Lambda 함수 간접 호출
  - AWS Command Line Interface v2를 사용하여 DynamoDB 글로벌 보조 인덱스 관리
  - AWS Command Line Interface v2를 사용하여 DynamoDB 리소스 기반 정책 관리
  - AWS SDK를 사용하여 Amazon DynamoDB의 성능 모니터링
  - AWS SDK로 고급 DynamoDB 쿼리 작업 수행
  - AWS SDK를 사용하여 DynamoDB에서 목록 작업 수행
  - AWS SDK를 사용하여 DynamoDB에서 맵 작업 수행
  - AWS SDK를 사용하여 DynamoDB에서 집합 작업 수행
  - PartiQL 문 배치 및 AWS SDK를 사용하여 DynamoDB 테이블 쿼리

- PartiQL 및 AWS SDK를 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 글로벌 보조 인덱스를 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 begins_with 조건을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 정렬 키에 날짜 범위를 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 복잡한 필터 표현식을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 동적 필터 표현식을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 필터 표현식 및 제한을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 중첩 속성을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 페이지 매김을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 강력하게 일관된 읽기를 사용하여 DynamoDB 테이블 쿼리
- AWS SDK와 함께 PartiQL SELECT 문을 사용하여 DynamoDB 데이터 쿼리
- AWS SDK를 사용하여 TTL 항목에 대한 DynamoDB 테이블 쿼리
- AWS SDK로 날짜 및 시간 패턴을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK를 사용하여 EXIF 및 기타 이미지 정보 저장
- AWS Command Line Interface v2를 사용하여 DynamoDB에 대한 속성 기반 액세스 제어 설정
- AWS SDK를 사용하여 DynamoDB에서 업데이트 표현식 순서 이해
- AWS SDK를 사용하여 원 처리량으로 DynamoDB 테이블 설정 업데이트
- AWS SDK를 사용하여 TTL이 포함된 DynamoDB 항목 업데이트
- AWS SDK에서 PartiQL UPDATE 문을 사용하여 DynamoDB 데이터 업데이트
- API Gateway를 사용하여 Lambda 함수 간접 호출
- Step Functions를 사용하여 Lambda 함수 간접 호출
- AWS SDK를 사용하여 DynamoDB용 문서 모델 사용
- AWS SDK를 사용하여 DynamoDB용 상위 수준 객체 지속성 모델 사용
- AWS SDK를 사용하여 DynamoDB에서 원자성 카운터 작업 사용
- AWS SDK를 사용하여 DynamoDB에서 조건부 작업 사용
- AWS SDK를 사용하여 DynamoDB에서 표현식 속성 이름 사용
- 예약된 이벤트를 사용하여 Lambda 함수 간접 호출
- AWS Command Line Interface v2를 사용한 DynamoDB 로컬 보조 인덱스 작업
- AWS Command Line Interface v2를 사용한 DynamoDB Streams 및 Time-to-Live 작업

- AWS Command Line Interface v2를 사용한 DynamoDB 글로벌 테이블 및 최종 일관성(MREC)을 갖춘 다중 리전 복제 작업
- AWS Command Line Interface v2를 사용한 DynamoDB 리소스 태그 지정 작업
- AWS Command Line Interface v2를 사용한 DynamoDB 테이블 암호화 작업
- DynamoDB의 서버리스 예시
  - DynamoDB 트리거에서 간접적으로 Lambda 함수 간접 호출
  - DynamoDB 트리거로 Lambda 함수에 대한 배치 항목 실패 보고
- DynamoDB에 대한 AWS 커뮤니티 기여
  - 서버리스 애플리케이션 빌드 및 테스트

# AWS SDK를 사용한 DynamoDB용 기본 예제

다음 코드 예제에서는 AWS SDK에서 Amazon DynamoDB의 기본 기능을 사용하는 방법을 보여줍니다.

예시

- Hello DynamoDB
- AWS SDK와 함께 사용되는 DynamoDB의 기본 기능 알아보기
- AWS SDK를 사용한 DynamoDB에 대한 작업
  - AWS SDK와 함께 BatchExecuteStatement사용
  - AWS SDK 또는 CLI와 함께 BatchGetItem 사용
  - AWS SDK 또는 CLI와 함께 BatchWriteItem 사용
  - AWS SDK 또는 CLI와 함께 CreateTable 사용
  - AWS SDK 또는 CLI와 함께 DeleteItem 사용
  - AWS SDK 또는 CLI와 함께 DeleteTable 사용
  - AWS SDK 또는 CLI와 함께 DescribeTable 사용
  - AWS SDK 또는 CLI와 함께 DescribeTimeToLive 사용
  - AWS SDK와 함께 ExecuteStatement사용
  - AWS SDK 또는 CLI와 함께 GetItem 사용
  - AWS SDK 또는 CLI와 함께 ListTables 사용
  - AWS SDK 또는 CLI와 함께 PutItem 사용
  - AWS SDK 또는 CLI와 함께 Query 사용

- AWS SDK 또는 CLI와 함께 Scan 사용
- AWS SDK 또는 CLI와 함께 UpdateItem 사용
- AWS SDK 또는 CLI와 함께 UpdateTable 사용
- AWS SDK 또는 CLI와 함께 UpdateTimeToLive 사용

## Hello DynamoDB

다음 코드 예제에서는 DynamoDB를 사용하여 시작하는 방법을 보여 줍니다.

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
```

```
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables =
 dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListTables를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

CMakeLists.txt CMake 파일의 코드입니다.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS dynamodb)

# Set this project's name.
project("hello_dynamodb")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})
```

```
if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_dynamodb.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_dynamodb.cpp 소스 파일의 코드입니다.

```
#include <aws/core/Aws.h>
#include <aws/dynamodb/DynamoDBClient.h>
#include <aws/dynamodb/model/ListTablesRequest.h>
#include <iostream>

/*
 *  A "Hello DynamoDB" starter application which initializes an Amazon DynamoDB
 (DynamoDB) client and lists the
 *  DynamoDB tables.
 *
 *  main function
 *
 *  Usage: 'hello_dynamodb'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
```

```
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::DynamoDB::DynamoDBClient dynamodbClient(clientConfig);
        Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
        listTablesRequest.SetLimit(50);
        do {
            const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamodbClient.ListTables(
                    listTablesRequest);
            if (!outcome.IsSuccess()) {
                std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
                result = 1;
                break;
            }

            for (const auto &tableName: outcome.GetResult().GetTableNames()) {
                std::cout << tableName << std::endl;
            }

            listTablesRequest.SetExclusiveStartTableName(
                    outcome.GetResult().GetLastEvaluatedTableName());

        } while (!listTablesRequest.GetExclusiveStartTableName().empty());
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 ListTables를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
```

```
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 ListTables를 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS SDK for JavaScript에서 DynamoDB를 사용하는 방법에 대한 자세한 내용은 JavaScript를 사용한 DynamoDB 프로그래밍을 참조하세요.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 ListTables를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import boto3

# Create a DynamoDB client using the default credentials and region
dynamodb = boto3.client("dynamodb")

# Initialize a paginator for the list_tables operation
paginator = dynamodb.get_paginator("list_tables")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(Limit=10)

# List the tables in the current AWS account
print("Here are the DynamoDB tables in your account:")

# Use pagination to list all tables
table_names = []

for page in page_iterator:
    for table_name in page.get("TableNames", []):
        print(f"- {table_name}")
        table_names.append(table_name)

if not table_names:
    print("You don't have any DynamoDB tables in your account.")
else:
    print(f"\nFound {len(table_names)} tables.")
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 ListTables를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
require 'aws-sdk-dynamodb'
require 'logger'

# DynamoDBManager is a class responsible for managing DynamoDB operations
# such as listing all tables in the current AWS account.
class DynamoDBManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all DynamoDB tables in the current AWS account.
  def list_tables
    @logger.info('Here are the DynamoDB tables in your account:')

    paginator = @client.list_tables(limit: 10)
    table_names = []

    paginator.each_page do |page|
      page.table_names.each do |table_name|
        @logger.info("- #{table_name}")
        table_names << table_name
      end
    end

    if table_names.empty?
      @logger.info("You don't have any DynamoDB tables in your account.")
    else
      @logger.info("\nFound #{table_names.length} tables.")
    end
  end
end

if $PROGRAM_NAME == __FILE__
  dynamodb_client = Aws::DynamoDB::Client.new
  manager = DynamoDBManager.new(dynamodb_client)
  manager.list_tables
end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 ListTables를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK와 함께 사용되는 DynamoDB의 기본 기능 알아보기

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 영화 데이터를 저장할 수 있는 테이블을 생성합니다.
- 테이블에 하나의 영화를 추가하고 가져오고 업데이트합니다.
- 샘플 JSON 파일에서 테이블에 영화 데이터를 씁니다.
- 특정 연도에 개봉된 영화를 쿼리합니다.
- 특정 연도 범위 동안 개봉된 영화를 스캔합니다.
- 테이블에서 영화를 삭제한 다음, 테이블을 삭제합니다.

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// This example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
public class DynamoDbBasics
```

```
{
    public static bool IsInteractive = true;

    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
                    .AddTransient<DynamoDbWrapper>())
            .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper =
 host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

        Console.WriteLine(success
            ? $"\nTable: {tableName} successfully created."
            : $"\nCould not create {tableName}.");

        WaitForEnter();

        // Add a single new movie to the table.
        var newMovie = new Movie
```

```
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };

        success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();

        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous"
+
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie:
{newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
```

```
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
        };

        success = await dynamoDbWrapper.DeleteItemAsync(tableName,
movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
        }
```

```
        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
        var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName,
startYear, endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await dynamoDbWrapper.DeleteTableAsync(tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is
complete.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
```

```
    {
        if (!IsInteractive)
        {
            return;
        }

        Console.Clear();
        Console.WriteLine();
        Console.Write(new string(' ', 28));
        Console.WriteLine("DynamoDB Basics Example");
        Console.WriteLine(SepBar);
        Console.WriteLine("This demo application shows the basics of using
DynamoDB with the AWS SDK.");
        Console.WriteLine(SepBar);
        Console.WriteLine("The application does the following:");
        Console.WriteLine("\t1. Creates a table with partition: year and
sort:title.");
        Console.WriteLine("\t2. Adds a single movie to the table.");
        Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
        Console.WriteLine("\t4. Updates the rating and plot of the movie that was
just added.");
        Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
        Console.WriteLine("\t6. Deletes a movie.");
        Console.WriteLine("\t7. Uses QueryAsync to return all movies released in
a given year.");
        Console.WriteLine("\t8. Uses ScanAsync to return all movies released
within a range of years.");
        Console.WriteLine("\t9. Finally, it deletes the table that was just
created.");
        WaitForEnter();
    }

    /// <summary>
    /// Simple method to wait for the Enter key to be pressed.
    /// </summary>
    private static void WaitForEnter()
    {
        if (IsInteractive)
        {
            Console.WriteLine("\nPress <Enter> to continue.");
            Console.WriteLine(SepBar);
            _ = Console.ReadLine();
        }
    }
```

```
}
```

영화 데이터를 포함할 테이블을 생성합니다.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
```

```
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
 {tableName}. {ex.Message}");
            throw;
        }
    }
```

테이블에 하나의 영화를 추가합니다.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
```

```
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
 {ex.Message}");
            throw;
        }
    }
```

테이블에서 하나의 항목을 업데이트합니다.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
```

```
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
```

```
            throw;
        }
    }
```

영화 테이블에서 하나의 항목을 가져옵니다.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

영화 테이블에 항목 배치를 씁니다.

```
    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the DynamoDB table.
    /// </summary>
    /// <param name="movieFileName">The name of the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public List<Movie> ImportMovies(string movieFileName)
    {
        var moviesList = new List<Movie>();
        if (!File.Exists(movieFileName))
        {
            return moviesList;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
```

```
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of
IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();
```

```
            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch
write operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

테이블에서 하나의 항목을 삭제합니다.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
```

```
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

테이블에서 특정 연도에 릴리스된 영화를 쿼리합니다.

```
    /// <summary>
    /// Queries the table for movies released in a particular year and
    /// then displays the information for the movies returned.
    /// </summary>
    /// <param name="tableName">The name of the table to query.</param>
    /// <param name="year">The release year for which we want to
    /// view movies.</param>
```

```
    /// <returns>The number of movies that match the query.</returns>
    public async Task<int> QueryMoviesAsync(string tableName, int year)
    {
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);
```

```
            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
 movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
 {ex.Message}");
            throw;
        }
    }
```

테이블에서 특정 연도 범위 동안 릴리스된 영화를 스캔합니다.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
```

```
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"An error occurred while scanning table.
 {ex.Message}");
            throw;
        }
    }
```

영화 테이블을 삭제합니다.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
 deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
 table {tableName}. {ex.Message}");
            return false;
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Bash

Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDB 시작 시나리오입니다.

```
###############################################################################
# function dynamodb_getting_started_movies
```

```
#
# Scenario to create an Amazon DynamoDB table and perform a series of operations
 on the table.
#
# Returns:
#       0 - If successful.
#       1 - If an error occurred.
###############################################################################
function dynamodb_getting_started_movies() {

  source ./dynamodb_operations.sh

  key_schema_json_file="dynamodb_key_schema.json"
  attribute_definitions_json_file="dynamodb_attr_def.json"
  item_json_file="movie_item.json"
  key_json_file="movie_key.json"
  batch_json_file="batch.json"
  attribute_names_json_file="attribute_names.json"
  attributes_values_json_file="attribute_values.json"

  echo_repeat "*" 88
  echo
  echo "Welcome to the Amazon DynamoDB getting started demo."
  echo
  echo_repeat "*" 88
  echo

  local table_name
  echo -n "Enter a name for a new DynamoDB table: "
  get_input
  table_name=$get_input_result

  echo '[
  {"AttributeName": "year", "KeyType": "HASH"},
   {"AttributeName": "title", "KeyType": "RANGE"}
  ]' >"$key_schema_json_file"

  echo '[
  {"AttributeName": "year", "AttributeType": "N"},
   {"AttributeName": "title", "AttributeType": "S"}
  ]' >"$attribute_definitions_json_file"

  if dynamodb_create_table -n "$table_name" -a "$attribute_definitions_json_file"
 \
```

```
  -k "$key_schema_json_file" 1>/dev/null; then
  echo "Created a DynamoDB table named $table_name"
else
  errecho "The table failed to create. This demo will exit."
  clean_up
  return 1
fi

echo "Waiting for the table to become active...."

if dynamodb_wait_table_active -n "$table_name"; then
  echo "The table is now active."
else
  errecho "The table failed to become active. This demo will exit."
  cleanup "$table_name"
  return 1
fi

echo
echo_repeat "*" 88
echo

echo -n "Enter the title of a movie you want to add to the table: "
get_input
local added_title
added_title=$get_input_result

local added_year
get_int_input "What year was it released? "
added_year=$get_input_result

local rating
get_float_input "On a scale of 1 - 10, how do you rate it? " "1" "10"
rating=$get_input_result

local plot
echo -n "Summarize the plot for me: "
get_input
plot=$get_input_result

echo '{
  "year": {"N" :"'"$added_year"'"},
  "title": {"S" :  "'"$added_title"'"},
```

```
    "info": {"M" : {"plot": {"S" : "'"$plot"'"}, "rating":
{"N" :"'"$rating"'"} } }
  }' >"$item_json_file"

 if dynamodb_put_item -n "$table_name" -i "$item_json_file"; then
   echo "The movie '$added_title' was successfully added to the table
'$table_name'."
 else
   errecho "Put item failed. This demo will exit."
   clean_up "$table_name"
   return 1
 fi

 echo
 echo_repeat "*" 88
 echo

 echo "Let's update your movie '$added_title'."
 get_float_input "You rated it $rating, what new rating would you give it? " "1"
"10"
 rating=$get_input_result

 echo -n "You summarized the plot as '$plot'."
 echo "What would you say now? "
 get_input
 plot=$get_input_result

 echo '{
   "year": {"N" :"'"$added_year"'"},
   "title": {"S" :  "'"$added_title"'"}
   }' >"$key_json_file"

 echo '{
   ":r": {"N" :"'"$rating"'"},
   ":p": {"S" : "'"$plot"'"}
  }' >"$item_json_file"

 local update_expression="SET info.rating = :r, info.plot = :p"

 if dynamodb_update_item -n "$table_name" -k "$key_json_file" -e
"$update_expression" -v "$item_json_file"; then
   echo "Updated '$added_title' with new attributes."
 else
   errecho "Update item failed. This demo will exit."
```

```
    clean_up "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo "We will now use batch write to upload 150 movie entries into the table."

  local batch_json
  for batch_json in movie_files/movies_*.json; do
    echo "{ \"$table_name\" : $(<"$batch_json") }" >"$batch_json_file"
    if dynamodb_batch_write_item -i "$batch_json_file" 1>/dev/null; then
      echo "Entries in $batch_json added to table."
    else
      errecho "Batch write failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi
  done

  local title="The Lord of the Rings: The Fellowship of the Ring"
  local year="2001"

  if get_yes_no_input "Let's move on...do you want to get info about '$title'?
(y/n) "; then
    echo '{
  "year": {"N" :"'"$year"'"},
  "title": {"S" :  "'"$title"'"}
  }' >"$key_json_file"
    local info
    info=$(dynamodb_get_item -n "$table_name" -k "$key_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Get item failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$info"
  fi
```

```
  local ask_for_year=true
  while [[ "$ask_for_year" == true ]]; do
    echo "Let's get a list of movies released in a given year."
    get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
    year=$get_input_result
    echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

    echo '{
    ":v": {"N" :"'"$year"'"}
    }' >"$attributes_values_json_file"

    response=$(dynamodb_query -n "$table_name" -k "#n=:v" -a
"$attribute_names_json_file" -v "$attributes_values_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Query table failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$response"

    if ! get_yes_no_input "Try another year? (y/n) "; then
      ask_for_year=false
    fi
  done

  echo "Now let's scan for movies released in a range of years. Enter a year: "
  get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
  local start=$get_input_result

  get_int_input "Enter another year: " "1972" "2018"
  local end=$get_input_result

  echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

  echo '{
```

```
    ":v1": {"N" : "'"$start"'"},
    ":v2": {"N" : "'"$end"'"}
    }' >"$attributes_values_json_file"

 response=$(dynamodb_scan -n "$table_name" -f "#n BETWEEN :v1 AND :v2" -a
"$attribute_names_json_file" -v "$attributes_values_json_file")

 # shellcheck disable=SC2181
 if [[ ${?} -ne 0 ]]; then
   errecho "Scan table failed. This demo will exit."
   clean_up "$table_name"
   return 1
 fi

 echo "Here is what I found:"
 echo "$response"

 echo
 echo_repeat "*" 88
 echo

 echo "Let's remove your movie '$added_title' from the table."

 if get_yes_no_input "Do you want to remove '$added_title'? (y/n) "; then
   echo '{
 "year": {"N" :"'"$added_year"'"},
 "title": {"S" :  "'"$added_title"'"}
 }' >"$key_json_file"

   if ! dynamodb_delete_item -n "$table_name" -k "$key_json_file"; then
     errecho "Delete item failed. This demo will exit."
     clean_up "$table_name"
     return 1
   fi
 fi

 if get_yes_no_input "Do you want to delete the table '$table_name'? (y/n) ";
then
   if ! clean_up "$table_name"; then
     return 1
   fi
 else
   if ! clean_up; then
     return 1
```

```
    fi
  fi

  return 0
}
```

이 시나리오에 사용된 DynamoDB 함수입니다.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }
```

```
 # Retrieve the calling parameters.
 while getopts "n:a:k:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     a) attribute_definitions="${OPTARG}" ;;
     k) key_schema="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_definitions" ]]; then
   errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
   usage
   return 1
 fi

 if [[ -z "$key_schema" ]]; then
   errecho "ERROR: You must provide a key schema json file path the -k
parameter."
   usage
   return 1
 fi

 iecho "Parameters:\n"
 iecho "    table_name:   $table_name"
 iecho "    attribute_definitions:   $attribute_definitions"
 iecho "    key_schema:   $key_schema"
 iecho ""
```

```
  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}

###############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
#       1 - If it fails.
###############################################################################
function dynamodb_describe_table {
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
```

```
}

# Retrieve the calling parameters.
while getopts "n:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

local table_status
  table_status=$(
    aws dynamodb describe-table \
      --table-name "$table_name" \
      --output text \
      --query 'Table.TableStatus'
  )

 local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log "$error_code"
    errecho "ERROR: AWS reports describe-table operation failed.$table_status"
    return 1
  fi

echo "$table_status"

return 0
```

```
}

###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
```

```
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}

########################################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
```

```
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
    echo " -e update expression  -- An expression that defines one or more
 attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
  }

  while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
```

```
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi
if [[ -z "$update_expression" ]]; then
  errecho "ERROR: You must provide an update expression with the -e parameter."
  usage
  return 1
fi

if [[ -z "$values" ]]; then
  errecho "ERROR: You must provide a values json file path the -v parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho "    keys:   $keys"
iecho "    update_expression:   $update_expression"
iecho "    values:   $values"

response=$(aws dynamodb update-item \
  --table-name "$table_name" \
  --key file://"$keys" \
  --update-expression "$update_expression" \
  --expression-attribute-values file://"$values")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
```

```
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
```

```
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}

###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
```

```
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
  }
  query=""
  while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi
```

```
  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text \
      --query "$query")
  else
    response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}

###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
```

```
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
    echo " -a attribute_names -- Path to JSON file containing the attribute
 names."
    echo " -v attribute_values -- Path to JSON file containing the attribute
 values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:k:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) key_condition_expression="${OPTARG}" ;;
      a) attribute_names="${OPTARG}" ;;
      v) attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
```

```
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$key_condition_expression" ]]; then
    errecho "ERROR: You must provide a key condition expression with the -k
parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
```

```
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.
```

```
 # ######################################
 # Function usage explanation
 ######################################
 function usage() {
   echo "function dynamodb_scan"
   echo "Scan a DynamoDB table."
   echo " -n table_name  -- The name of the table."
   echo " -f filter_expression  -- The filter expression."
   echo " -a expression_attribute_names -- Path to JSON file containing the
expression attribute names."
   echo " -v expression_attribute_values -- Path to JSON file containing the
expression attribute values."
   echo " [-p projection_expression]  -- Optional projection expression."
   echo ""
 }

 while getopts "n:f:a:v:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     f) filter_expression="${OPTARG}" ;;
     a) expression_attribute_names="${OPTARG}" ;;
     v) expression_attribute_values="${OPTARG}" ;;
     p) projection_expression="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$filter_expression" ]]; then
   errecho "ERROR: You must provide a filter expression with the -f parameter."
```

```
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_names" ]]; then
    errecho "ERROR: You must provide expression attribute names with the -a
parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_values" ]]; then
    errecho "ERROR: You must provide expression attribute values with the -v
parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values")
  else
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
```

```
}

################################################################################
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
################################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to delete."
    echo ""
  }
  while getopts "n:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
```

```
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho ""

  response=$(aws dynamodb delete-item \
    --table-name "$table_name" \
    --key file://"$keys")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
```

```
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""
```

```
  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

이 시나리오에 사용된 유틸리티 함수입니다.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
```

```
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem

- PutItem
- Query
- Scan
- UpdateItem

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    {
        Aws::Client::ClientConfiguration clientConfig;
        //  1. Create a table with partition: year (N) and sort: title (S).
 (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::dynamodbGettingStartedScenario(clientConfig);

            // 9. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }
    }

//! Scenario to modify and query a DynamoDB table.
/*!
  \sa dynamodbGettingStartedScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::dynamodbGettingStartedScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;
    std::cout << "Welcome to the Amazon DynamoDB getting started demo." <<
 std::endl;
```

```
   std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
             << std::endl;

   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   // 2. Add a new movie.
   Aws::String title;
   float rating;
   int year;
   Aws::String plot;
   {
       title = askQuestion(
               "Enter the title of a movie you want to add to the table: ");
       year = askQuestionForInt("What year was it released? ");
       rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                         1, 10);
       plot = askQuestion("Summarize the plot for me: ");

       Aws::DynamoDB::Model::PutItemRequest putItemRequest;
       putItemRequest.SetTableName(MOVIE_TABLE_NAME);

       putItemRequest.AddItem(YEAR_KEY,

Aws::DynamoDB::Model::AttributeValue().SetN(year));
       putItemRequest.AddItem(TITLE_KEY,

Aws::DynamoDB::Model::AttributeValue().SetS(title));

       // Create attribute for the info map.
       Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       ratingAttribute->SetN(rating);
       infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       plotAttribute->SetS(plot);
       infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
```

```
        putItemRequest.AddItem(INFO_KEY, infoMapAttribute);

        Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
                putItemRequest);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add an item: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
              << std::endl;

    // 3. Update the rating and plot of the movie by using an update expression.
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);
        plot = askQuestion(Aws::String("You summarized the plot as '") + plot +
                           "'.\nWhat would you say now? ");

        Aws::DynamoDB::Model::UpdateItemRequest request;
        request.SetTableName(MOVIE_TABLE_NAME);
        request.AddKey(TITLE_KEY,
Aws::DynamoDB::Model::AttributeValue().SetS(title));
        request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(year));
        std::stringstream expressionStream;
        expressionStream << "set " << INFO_KEY << "." << RATING_KEY << " =:r, "
                         << INFO_KEY << "." << PLOT_KEY << " =:p";
        request.SetUpdateExpression(expressionStream.str());
        request.SetExpressionAttributeValues({
                                                     {":r",
Aws::DynamoDB::Model::AttributeValue().SetN(
                                                             rating)},
                                                     {":p",
Aws::DynamoDB::Model::AttributeValue().SetS(
                                                             plot)}
                                             });

        request.SetReturnValues(Aws::DynamoDB::Model::ReturnValue::UPDATED_NEW);
```

```
        const Aws::DynamoDB::Model::UpdateItemOutcome &result =
dynamoClient.UpdateItem(
                request);
        if (!result.IsSuccess()) {
            std::cerr << "Error updating movie " + result.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;

    // 4. Put 250 movies in the table from moviedata.json.
    {
        std::cout << "Adding movies from a json file to the database." <<
std::endl;
        const size_t MAX_SIZE_FOR_BATCH_WRITE = 25;
        const size_t MOVIES_TO_WRITE = 10 * MAX_SIZE_FOR_BATCH_WRITE;
        Aws::String jsonString = getMovieJSON();
        if (!jsonString.empty()) {
            Aws::Utils::Json::JsonValue json(jsonString);
            Aws::Utils::Array<Aws::Utils::Json::JsonView> movieJsons =
json.View().AsArray();
            Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;

            // To add movies with a cross-section of years, use an appropriate
increment
            // value for iterating through the database.
            size_t increment = movieJsons.GetLength() / MOVIES_TO_WRITE;
            for (size_t i = 0; i < movieJsons.GetLength(); i += increment) {
                writeRequests.push_back(Aws::DynamoDB::Model::WriteRequest());
                Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
putItems = movieJsonViewToAttributeMap(
                        movieJsons[i]);
                Aws::DynamoDB::Model::PutRequest putRequest;
                putRequest.SetItem(putItems);
                writeRequests.back().SetPutRequest(putRequest);
                if (writeRequests.size() == MAX_SIZE_FOR_BATCH_WRITE) {
                    Aws::DynamoDB::Model::BatchWriteItemRequest request;
                    request.AddRequestItems(MOVIE_TABLE_NAME, writeRequests);
                    const Aws::DynamoDB::Model::BatchWriteItemOutcome &outcome =
dynamoClient.BatchWriteItem(
                            request);
```

```
                    if (!outcome.IsSuccess()) {
                        std::cerr << "Unable to batch write movie data: "
                                  << outcome.GetError().GetMessage()
                                  << std::endl;
                        writeRequests.clear();
                        break;
                    }
                    else {
                        std::cout << "Added batch of " << writeRequests.size()
                                  << " movies to the database."
                                  << std::endl;
                    }
                    writeRequests.clear();
                }
            }
        }
    }

    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;

    // 5. Get a movie by Key (partition + sort).
    {
        Aws::String titleToGet("King Kong");
        Aws::String answer = askQuestion(Aws::String(
                "Let's move on...Would you like to get info about '" + titleToGet
+
                "'? (y/n) "));
        if (answer == "y") {
            Aws::DynamoDB::Model::GetItemRequest request;
            request.SetTableName(MOVIE_TABLE_NAME);
            request.AddKey(TITLE_KEY,

Aws::DynamoDB::Model::AttributeValue().SetS(titleToGet));
            request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(1933));

            const Aws::DynamoDB::Model::GetItemOutcome &result =
dynamoClient.GetItem(
                    request);
            if (!result.IsSuccess()) {
                std::cerr << "Error " << result.GetError().GetMessage();
            }
            else {
```

```
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = result.GetResult().GetItem();
                if (!item.empty()) {
                    std::cout << "\nHere's what I found:" << std::endl;
                    printMovieInfo(item);
                }
                else {
                    std::cout << "\nThe movie was not found in the database."
                              << std::endl;
                }
            }
        }
    }

    // 6. Use Query with a key condition expression to return all movies
    //    released in a given year.
    Aws::String doAgain = "n";
    do {
        Aws::DynamoDB::Model::QueryRequest req;

        req.SetTableName(MOVIE_TABLE_NAME);

        // "year" is a DynamoDB reserved keyword and must be replaced with an
        // expression attribute name.
        req.SetKeyConditionExpression("#dynobase_year = :valueToMatch");
        req.SetExpressionAttributeNames({{"#dynobase_year", YEAR_KEY}});

        int yearToMatch = askQuestionForIntRange(
                "\nLet's get a list of movies released in"
                " a given year. Enter a year between 1972 and 2018 ",
                1972, 2018);
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
        attributeValues.emplace(":valueToMatch",
                                Aws::DynamoDB::Model::AttributeValue().SetN(
                                        yearToMatch));
        req.SetExpressionAttributeValues(attributeValues);

        const Aws::DynamoDB::Model::QueryOutcome &result =
dynamoClient.Query(req);
        if (result.IsSuccess()) {
            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
            if (!items.empty()) {
```

```
                std::cout << "\nThere were " << items.size()
                          << " movies in the database from "
                          << yearToMatch << "." << std::endl;
                for (const auto &item: items) {
                    printMovieInfo(item);
                }
                doAgain = "n";
            }
            else {
                std::cout << "\nNo movies from " << yearToMatch
                          << " were found in the database"
                          << std::endl;
                doAgain = askQuestion(Aws::String("Try another year? (y/n) "));
            }
        }
        else {
            std::cerr << "Failed to Query items: " <<
result.GetError().GetMessage()
                      << std::endl;
        }

    } while (doAgain == "y");

    //  7. Use Scan to return movies released within a range of years.
    //     Show how to paginate data using ExclusiveStartKey. (Scan +
FilterExpression)
    {
        int startYear = askQuestionForIntRange("\nNow let's scan a range of years
"
                                               "for movies in the database. Enter
a start year: ",
                                               1972, 2018);
        int endYear = askQuestionForIntRange("\nEnter an end year: ",
                                             startYear, 2018);
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
        do {
            Aws::DynamoDB::Model::ScanRequest scanRequest;
            scanRequest.SetTableName(MOVIE_TABLE_NAME);
            scanRequest.SetFilterExpression(
                    "#dynobase_year >= :startYear AND #dynobase_year
<= :endYear");
            scanRequest.SetExpressionAttributeNames({{"#dynobase_year",
YEAR_KEY}});
```

```
            Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
            attributeValues.emplace(":startYear",
                                    Aws::DynamoDB::Model::AttributeValue().SetN(
                                            startYear));
            attributeValues.emplace(":endYear",
                                    Aws::DynamoDB::Model::AttributeValue().SetN(
                                            endYear));
            scanRequest.SetExpressionAttributeValues(attributeValues);

            if (!exclusiveStartKey.empty()) {
                scanRequest.SetExclusiveStartKey(exclusiveStartKey);
            }

            const Aws::DynamoDB::Model::ScanOutcome &result = dynamoClient.Scan(
                    scanRequest);
            if (result.IsSuccess()) {
                const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
                if (!items.empty()) {
                    std::stringstream stringStream;
                    stringStream << "\nFound " << items.size() << " movies in one
scan."
                                 << " How many would you like to see? ";
                    size_t count = askQuestionForInt(stringStream.str());
                    for (size_t i = 0; i < count && i < items.size(); ++i) {
                        printMovieInfo(items[i]);
                    }
                }
                else {
                    std::cout << "\nNo movies in the database between " <<
startYear <<
                              " and " << endYear << "." << std::endl;
                }

                exclusiveStartKey = result.GetResult().GetLastEvaluatedKey();
                if (!exclusiveStartKey.empty()) {
                    std::cout << "Not all movies were retrieved. Scanning for
more."
                              << std::endl;
                }
                else {
                    std::cout << "All movies were retrieved with this scan."
```

```
                                << std::endl;
                    }
                }
                else {
                    std::cerr << "Failed to Scan movies: "
                              << result.GetError().GetMessage() << std::endl;
                }
            } while (!exclusiveStartKey.empty());
        }

        // 8. Delete a movie. (DeleteItem)
        {
            std::stringstream stringStream;
            stringStream << "\nWould you like to delete the movie " << title
                         << " from the database? (y/n) ";
            Aws::String answer = askQuestion(stringStream.str());
            if (answer == "y") {
                Aws::DynamoDB::Model::DeleteItemRequest request;
                request.AddKey(YEAR_KEY,
     Aws::DynamoDB::Model::AttributeValue().SetN(year));
                request.AddKey(TITLE_KEY,
                               Aws::DynamoDB::Model::AttributeValue().SetS(title));
                request.SetTableName(MOVIE_TABLE_NAME);

                const Aws::DynamoDB::Model::DeleteItemOutcome &result =
     dynamoClient.DeleteItem(
                        request);
                if (result.IsSuccess()) {
                    std::cout << "\nRemoved \"" << title << "\" from the database."
                              << std::endl;
                }
                else {
                    std::cerr << "Failed to delete the movie: "
                              << result.GetError().GetMessage()
                              << std::endl;
                }
            }
        }

        return true;
    }

    //! Routine to convert a JsonView object to an attribute map.
    /*!
```

```
  \sa movieJsonViewToAttributeMap()
  \param jsonView: Json view object.
  \return map: Map that can be used in a DynamoDB request.
 */
Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
AwsDoc::DynamoDB::movieJsonViewToAttributeMap(
        const Aws::Utils::Json::JsonView &jsonView) {
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> result;

    if (jsonView.KeyExists(YEAR_KEY)) {
        result[YEAR_KEY].SetN(jsonView.GetInteger(YEAR_KEY));
    }
    if (jsonView.KeyExists(TITLE_KEY)) {
        result[TITLE_KEY].SetS(jsonView.GetString(TITLE_KEY));
    }
    if (jsonView.KeyExists(INFO_KEY)) {
        Aws::Map<Aws::String, const
 std::shared_ptr<Aws::DynamoDB::Model::AttributeValue>> infoMap;
        Aws::Utils::Json::JsonView infoView = jsonView.GetObject(INFO_KEY);
        if (infoView.KeyExists(RATING_KEY)) {
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetN(infoView.GetDouble(RATING_KEY));
            infoMap.emplace(std::make_pair(RATING_KEY, attributeValue));
        }
        if (infoView.KeyExists(PLOT_KEY)) {
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetS(infoView.GetString(PLOT_KEY));
            infoMap.emplace(std::make_pair(PLOT_KEY, attributeValue));
        }

        result[INFO_KEY].SetM(infoMap);
    }

    return result;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
```

```
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
 std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
 dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
```

```
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
```

```
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
```

```
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

대화식 시나리오를 실행하여 테이블을 생성하고 테이블에 대한 작업을 수행합니다.

```
import (
 "context"
 "fmt"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunMovieScenario is an interactive example that shows you how to use the AWS
 SDK for Go
// to create and use an Amazon DynamoDB table that stores data about movies.
//
//  1. Create a table that can hold movie data.
//  2. Put, get, and update a single movie in the table.
//  3. Write movie data to the table from a sample JSON file.
//  4. Query for movies that were released in a given year.
//  5. Scan for movies that were released in a range of years.
//  6. Delete a movie from the table.
//  7. Delete the table.
//
// This example creates a DynamoDB service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
//
// The specified movie sampler is used to get sample data from a URL that is
 loaded
// into the named table.
func RunMovieScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner,
 tableName string,
 movieSampler actions.IMovieSampler) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()
```

```
log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the Amazon DynamoDB getting started demo.")
log.Println(strings.Repeat("-", 88))

tableBasics := actions.TableBasics{TableName: tableName,
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig)}

exists, err := tableBasics.TableExists(ctx)
if err != nil {
 panic(err)
}
if !exists {
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}

var customMovie actions.Movie
customMovie.Title = questioner.Ask("Enter a movie title to add to the table:",
 demotools.NotEmpty{})
customMovie.Year = questioner.AskInt("What year was it released?",
 demotools.NotEmpty{}, demotools.InIntRange{Lower: 1900, Upper: 2030})
customMovie.Info = map[string]interface{}{}
customMovie.Info["rating"] = questioner.AskFloat64(
 "Enter a rating between 1 and 10:",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
customMovie.Info["plot"] = questioner.Ask("What's the plot? ",
 demotools.NotEmpty{})
err = tableBasics.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's update your movie. You previously rated it %v.\n",
customMovie.Info["rating"])
customMovie.Info["rating"] = questioner.AskFloat64(
```

```
 "What new rating would you give it?",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
log.Printf("You summarized the plot as '%v'.\n", customMovie.Info["plot"])
customMovie.Info["plot"] = questioner.Ask("What would you say now?",
 demotools.NotEmpty{})
attributes, err := tableBasics.UpdateMovie(ctx, customMovie)
if err == nil {
 log.Printf("Updated %v with new values.\n", customMovie.Title)
 for _, attVal := range attributes {
  for valKey, val := range attVal {
   log.Printf("\t%v: %v\n", valKey, val)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting movie data from %v and adding 250 movies to the table...\n",
 movieSampler.GetURL())
movies := movieSampler.GetSampleMovies()
written, err := tableBasics.AddMovieBatch(ctx, movies, 250)
if err != nil {
 panic(err)
} else {
 log.Printf("Added %v movies to the table.\n", written)
}

show := 10
if show > written {
 show = written
}
log.Printf("The first %v movies in the table are:", show)
for index, movie := range movies[:show] {
 log.Printf("\t%v. %v\n", index+1, movie.Title)
}
movieIndex := questioner.AskInt(
 "Enter the number of a movie to get info about it: ",
 demotools.InIntRange{Lower: 1, Upper: show},
)
movie, err := tableBasics.GetMovie(ctx, movies[movieIndex-1].Title,
movies[movieIndex-1].Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))
```

```
log.Println("Let's get a list of movies released in a given year.")
releaseYear := questioner.AskInt("Enter a year between 1972 and 2018: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018},
)
releases, err := tableBasics.Query(ctx, releaseYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released in %v!\n", releaseYear)
 } else {
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Println("Now let's scan for movies released in a range of years.")
startYear := questioner.AskInt("Enter a year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
endYear := questioner.AskInt("Enter another year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
releases, err = tableBasics.Scan(ctx, startYear, endYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released between %v and %v!\n",
startYear, endYear)
 } else {
  log.Printf("Found %v movies. In this list, the plot is <nil> because "+
   "we used a projection expression when scanning for items to return only "+
   "the title, year, and rating.\n", len(releases))
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

var tables []string
if questioner.AskBool("Do you want to list all of your tables? (y/n) ", "y") {
 tables, err = tableBasics.ListTables(ctx)
 if err == nil {
  log.Printf("Found %v tables:", len(tables))
  for _, table := range tables {
```

```
    log.Printf("\t%v", table)
   }
  }
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Let's remove your movie '%v'.\n", customMovie.Title)
 if questioner.AskBool("Do you want to delete it from the table? (y/n) ", "y") {
  err = tableBasics.DeleteMovie(ctx, customMovie)
 }
 if err == nil {
  log.Printf("Deleted %v.\n", customMovie.Title)
 }

 if questioner.AskBool("Delete the table, too? (y/n)", "y") {
  err = tableBasics.DeleteTable(ctx)
 } else {
  log.Println("Don't forget to delete the table when you're done or you might " +
   "incur charges on your account.")
 }
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

DynamoDB 작업을 호출하는 구문과 메서드를 생성합니다.

```
import (
 "context"
```

```
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}
```

```
// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}



// ListTables lists the DynamoDB table names for the current account.
```

```
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
```

```
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
```

```
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx,
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
```

```
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
```

```
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
```

```
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDB 테이블을 생성합니다.

```
    // Create a table with a Sort key.
    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());
```

```
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE)
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
```

```
            System.exit(1);
        }
    }
```

헬퍼 함수를 생성하여 샘플 JSON 파일을 다운로드하고 추출합니다.

```
    // Load data into the table.
    public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
            .dynamoDbClient(ddb)
            .build();

        DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
        ObjectNode currentNode;
        int t = 0;
        while (iter.hasNext()) {
            // Only add 200 Movies to the table.
            if (t == 200)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            Movies movies = new Movies();
            movies.setYear(year);
            movies.setTitle(title);
            movies.setInfo(info);

            // Put the data into the Amazon DynamoDB Movie table.
            mappedTable.putItem(movies);
            t++;
        }
    }
```

테이블에서 항목을 가져옵니다.

```
    public static void getItem(DynamoDbClient ddb) {

        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put("year", AttributeValue.builder()
            .n("1933")
            .build());

        keyToGet.put("title", AttributeValue.builder()
            .s("King Kong")
            .build());

        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName("Movies")
            .build();

        try {
            Map<String, AttributeValue> returnedItem =
ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

전체 예제는 다음과 같습니다.

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 * <p>
 * This Java example performs these tasks:
 * <p>
 * 1. Creates the Amazon DynamoDB Movie table with partition and sort key.
 * 2. Puts data into the Amazon DynamoDB table from a JSON document using the
 * Enhanced client.
 * 3. Gets data from the Movie table.
 * 4. Adds a new item.
 * 5. Updates an item.
 * 6. Uses a Scan to query items using the Enhanced client.
 * 7. Queries all items where the year is 2013 using the Enhanced Client.
 * 8. Deletes the table.
 */

public class Scenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws IOException {
        String tableName = "Movies";
        String fileName = "../../../resources/sample_files/movies.json";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon DynamoDB example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println(
            "1. Creating an Amazon DynamoDB table named Movies with a key named
 year and a sort key named title.");
```

```
        createTable(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Loading data into the Amazon DynamoDB table.");
        loadData(ddb, tableName, fileName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Getting data from the Movie table.");
        getItem(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Putting a record into the Amazon DynamoDB
table.");
        putRecord(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Updating a record.");
        updateTableItem(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Scanning the Amazon DynamoDB table.");
        scanMovies(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Querying the Movies released in 2013.");
        queryTable(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        System.out.println(DASHES);

        ddb.close();
    }

    // Create a table with a Sort key.
    public static void createTable(DynamoDbClient ddb, String tableName) {
```

```
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE)
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
```

```
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Query the table.
    public static void queryTable(DynamoDbClient ddb) {
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

            DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
            QueryConditional queryConditional = QueryConditional
                .keyEqualTo(Key.builder()
                    .partitionValue(2013)
                    .build());

            // Get items in the table and write out the ID value.
            Iterator<Movies> results =
custTable.query(queryConditional).items().iterator();
            String result = "";

            while (results.hasNext()) {
                Movies rec = results.next();
                System.out.println("The title of the movie is " +
rec.getTitle());
                System.out.println("The movie information  is " + rec.getInfo());
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

```
   // Scan the table.
   public static void scanMovies(DynamoDbClient ddb, String tableName) {
       System.out.println("******* Scanning all movies.\n");
       try {
           DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
               .dynamoDbClient(ddb)
               .build();

           DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
           Iterator<Movies> results = custTable.scan().items().iterator();
           while (results.hasNext()) {
               Movies rec = results.next();
               System.out.println("The movie title is " + rec.getTitle());
               System.out.println("The movie year is " + rec.getYear());
           }

       } catch (DynamoDbException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }

   // Load data into the table.
   public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
       DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
           .dynamoDbClient(ddb)
           .build();

       DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
       JsonParser parser = new JsonFactory().createParser(new File(fileName));
       com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
       Iterator<JsonNode> iter = rootNode.iterator();
       ObjectNode currentNode;
       int t = 0;
       while (iter.hasNext()) {
           // Only add 200 Movies to the table.
           if (t == 200)
               break;
```

```
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            Movies movies = new Movies();
            movies.setYear(year);
            movies.setTitle(title);
            movies.setInfo(info);

            // Put the data into the Amazon DynamoDB Movie table.
            mappedTable.putItem(movies);
            t++;
        }
    }

    // Update the record to include show only directors.
    public static void updateTableItem(DynamoDbClient ddb, String tableName) {
        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put("year", AttributeValue.builder().n("1933").build());
        itemKey.put("title", AttributeValue.builder().s("King Kong").build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put("info", AttributeValueUpdate.builder()
            .value(AttributeValue.builder().s("{\"directors\":[\"Merian C. Cooper
\",\"Ernest B. Schoedsack\"]")
                .build())
            .action(AttributeAction.PUT)
            .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(itemKey)
            .attributeUpdates(updatedValues)
            .build();

        try {
            ddb.updateItem(request);
        } catch (ResourceNotFoundException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
```

```
            System.exit(1);
        }

        System.out.println("Item was updated!");
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{
        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    public static void putRecord(DynamoDbClient ddb) {
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

            DynamoDbTable<Movies> table = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));

            // Populate the Table.
            Movies record = new Movies();
            record.setYear(2020);
            record.setTitle("My Movie2");
            record.setInfo("no info");
            table.putItem(record);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Added a new movie to the table.");
```

```
    }

    public static void getItem(DynamoDbClient ddb) {

        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put("year", AttributeValue.builder()
            .n("1933")
            .build());

        keyToGet.put("title", AttributeValue.builder()
            .s("King Kong")
            .build());

        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName("Movies")
            .build();

        try {
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 항목을 참조하세요.

- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { readFileSync } from "node:fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
```

```
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
```

```
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });

  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
 1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so
 'title'
      // is required.
      title: "The Evil Dead",
      // Every other attribute is optional.
```

```
      info: {
        genres: ["Horror"],
      },
    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/
latest/developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
```

```
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
  });
  const updateResponse = await docClient.send(updateCommand);
  log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

  /**
   * Delete a movie from the table.
   */

  log("Deleting a single movie from the table.");
  const deleteCommand = new DeleteCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
  });
  await client.send(deleteCommand);
  log("Movie deleted.");

  /**
   * Upload a batch of movies.
   */

  log("Adding movies from local JSON file.");
  const file = readFileSync(
    `${dirname}../../../../resources/sample_files/movies.json`,
  );
  const movies = JSON.parse(file.toString());
  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);
  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        [tableName]: putRequests,
      },
    });
```

```
    await docClient.send(command);
  }
  log("Movies added.");

  /**
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log("Scan for movies released between 1980 and 1990");
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
```

```
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression.
 Scan will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  log(`Deleting table ${tableName}.`);
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable

- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDB 테이블을 생성합니다.

```
suspend fun createScenarioTable(
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
```

```
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}
```

헬퍼 함수를 생성하여 샘플 JSON 파일을 다운로드하고 추출합니다.

```
// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
```

```
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}

suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}
```

테이블에서 항목을 가져옵니다.

```
suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
```

```
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

전체 예제는 다음과 같습니다.

```
suspend fun main() {
    val tableName = "Movies"
    val fileName = "../../../resources/sample_files/movies.json"
    val partitionAlias = "#a"

    println("Creating an Amazon DynamoDB table named Movies with a key named id
 and a sort key named title.")
    createScenarioTable(tableName, "year")
    loadData(tableName, fileName)
    getMovie(tableName, "year", "1933")
    scanMovies(tableName)
    val count = queryMovieTable(tableName, "year", partitionAlias)
    println("There are $count Movies released in 2013.")
    deletIssuesTable(tableName)
}

suspend fun createScenarioTable(
    tableNameVal: String,
```

```
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}
```

```
// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}

suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }
```

```
    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}

suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}

suspend fun deletIssuesTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

```
suspend fun queryMovieTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = "year"

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.N("2013")

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}

suspend fun scanMovies(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
namespace DynamoDb\Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;
use DynamoDb\DynamoDBService;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithDynamoDB
{
    public function run()
    {
        echo("\n");
```

```
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB getting started demo using PHP!
\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDBService();

        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);
```

```
        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
|| $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];
        $attributes = ["rating" =>
            [
                'AttributeName' => 'rating',
                'AttributeType' => 'N',
                'Value' => $rating,
            ],
            'plot' => [
                'AttributeName' => 'plot',
                'AttributeType' => 'S',
                'Value' => $plot,
            ]
        ];
        $service->updateItemAttributesByKey($tableName, $key, $attributes);
        echo "Movie added and updated.";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);


        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";
```

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

        $movie = $service->getItemByKey($tableName, $key);
        echo "Ok, you have rated {$movie['Item']['title']['S']} as a
 {$movie['Item']['rating']['N']}\n";

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;
```

```
        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}
```

- API 세부 정보는 AWS SDK for PHP API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDB 테이블을 캡슐화하는 클래스를 생성합니다.

```
from decimal import Decimal
from io import BytesIO
import json
import logging
import os
from pprint import pprint
import requests
from zipfile import ZipFile
import boto3
from boto3.dynamodb.conditions import Key
from botocore.exceptions import ClientError
from question import Question

logger = logging.getLogger(__name__)

class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
```

```
                "Kelly Preston",
                "John C. Reilly"
            ]
        }
    }
"""

def __init__(self, dyn_resource):
    """
    :param dyn_resource: A Boto3 DynamoDB resource.
    """
    self.dyn_resource = dyn_resource
    # The table variable is set during the scenario in the call to
    # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
    self.table = None


def exists(self, table_name):
    """
    Determines whether a table exists. As a side effect, stores the table in
    a member variable.

    :param table_name: The name of the table to check.
    :return: True when the table exists; otherwise, False.
    """
    try:
        table = self.dyn_resource.Table(table_name)
        table.load()
        exists = True
    except ClientError as err:
        if err.response["Error"]["Code"] == "ResourceNotFoundException":
            exists = False
        else:
            logger.error(
                "Couldn't check for existence of %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
    else:
        self.table = table
    return exists
```

```
    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table


    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
```

```
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tables


    def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
least
                       the keys required by the schema that was specified when
the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
```

```
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
        }
        try:
            done = False
            start_key = None
            while not done:
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
```

```
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies


    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

헬퍼 함수를 생성하여 샘플 JSON 파일을 다운로드하고 추출합니다.

```
def get_sample_movie_data(movie_file_name):
    """
    Gets sample movie data, either from a local file or by first downloading it
 from
    the Amazon DynamoDB developer guide.

    :param movie_file_name: The local file name where the movie data is stored in
 JSON format.
    :return: The movie data as a dict.
    """
    if not os.path.isfile(movie_file_name):
        print(f"Downloading {movie_file_name}...")
        movie_content = requests.get(
            "https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip"
        )
        movie_zip = ZipFile(BytesIO(movie_content.content))
        movie_zip.extractall()

    try:
        with open(movie_file_name) as movie_file:
            movie_data = json.load(movie_file, parse_float=Decimal)
    except FileNotFoundError:
        print(
            f"File {movie_file_name} not found. You must first download the file
 to "
            "run this demo. See the README for instructions."
        )
        raise
    else:
        # The sample file lists over 4000 movies, return only the first 250.
        return movie_data[:250]
```

대화식 시나리오를 실행하여 테이블을 생성하고 테이블에 대한 작업을 수행합니다.

```
def run_scenario(table_name, movie_file_name, dyn_resource):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB getting started demo.")
    print("-" * 88)

    movies = Movies(dyn_resource)
    movies_exists = movies.exists(table_name)
    if not movies_exists:
        print(f"\nCreating table {table_name}...")
        movies.create_table(table_name)
        print(f"\nCreated table {movies.table.name}.")

    my_movie = Question.ask_questions(
        [
            Question(
                "title", "Enter the title of a movie you want to add to the
 table: "
            ),
            Question("year", "What year was it released? ", Question.is_int),
            Question(
                "rating",
                "On a scale of 1 - 10, how do you rate it? ",
                Question.is_float,
                Question.in_range(1, 10),
            ),
            Question("plot", "Summarize the plot for me: "),
        ]
    )
    movies.add_movie(**my_movie)
    print(f"\nAdded '{my_movie['title']}' to '{movies.table.name}'.")
    print("-" * 88)

    movie_update = Question.ask_questions(
        [
            Question(
                "rating",
                f"\nLet's update your movie.\nYou rated it {my_movie['rating']},
 what new "
                f"rating would you give it? ",
                Question.is_float,
                Question.in_range(1, 10),
```

```
            ),
            Question(
                "plot",
                f"You summarized the plot as '{my_movie['plot']}'.\nWhat would
you say now? ",
            ),
        ]
    )
    my_movie.update(movie_update)
    updated = movies.update_movie(**my_movie)
    print(f"\nUpdated '{my_movie['title']}' with new attributes:")
    pprint(updated)
    print("-" * 88)

    if not movies_exists:
        movie_data = get_sample_movie_data(movie_file_name)
        print(f"\nReading data from '{movie_file_name}' into your table.")
        movies.write_batch(movie_data)
        print(f"\nWrote {len(movie_data)} movies into {movies.table.name}.")
    print("-" * 88)

    title = "The Lord of the Rings: The Fellowship of the Ring"
    if Question.ask_question(
        f"Let's move on...do you want to get info about '{title}'? (y/n) ",
        Question.is_yesno,
    ):
        movie = movies.get_movie(title, 2001)
        print("\nHere's what I found:")
        pprint(movie)
    print("-" * 88)

    ask_for_year = True
    while ask_for_year:
        release_year = Question.ask_question(
            f"\nLet's get a list of movies released in a given year. Enter a year
between "
            f"1972 and 2018: ",
            Question.is_int,
            Question.in_range(1972, 2018),
        )
        releases = movies.query_movies(release_year)
        if releases:
            print(f"There were {len(releases)} movies released in
{release_year}:")
```

```
            for release in releases:
                print(f"\t{release['title']}")
            ask_for_year = False
        else:
            print(f"I don't know about any movies released in {release_year}!")
            ask_for_year = Question.ask_question(
                "Try another year? (y/n) ", Question.is_yesno
            )
    print("-" * 88)

    years = Question.ask_questions(
        [
            Question(
                "first",
                f"\nNow let's scan for movies released in a range of years. Enter
a year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
            Question(
                "second",
                "Now enter another year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
        ]
    )
    releases = movies.scan_movies(years)
    if releases:
        count = Question.ask_question(
            f"\nFound {len(releases)} movies. How many do you want to see? ",
            Question.is_int,
            Question.in_range(1, len(releases)),
        )
        print(f"\nHere are your {count} movies:\n")
        pprint(releases[:count])
    else:
        print(
            f"I don't know about any movies released between {years['first']} "
            f"and {years['second']}."
        )
    print("-" * 88)

    if Question.ask_question(
```

```
        f"\nLet's remove your movie from the table. Do you want to remove "
        f"'{my_movie['title']}'? (y/n)",
        Question.is_yesno,
    ):
        movies.delete_movie(my_movie["title"], my_movie["year"])
        print(f"\nRemoved '{my_movie['title']}' from the table.")
    print("-" * 88)

    if Question.ask_question(f"\nDelete the table? (y/n) ", Question.is_yesno):
        movies.delete_table()
        print(f"Deleted {table_name}.")
    else:
        print(
            "Don't forget to delete the table when you're done or you might incur
 "
            "charges on your account."
        )

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        run_scenario(
            "doc-example-table-movies", "moviedata.json",
 boto3.resource("dynamodb")
        )
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

이 시나리오에서는 다음 헬퍼 클래스를 사용하여 명령 프롬프트에서 질문을 합니다.

```
class Question:
    """
    A helper class to ask questions at a command prompt and validate and convert
    the answers.
    """

    def __init__(self, key, question, *validators):
        """
        :param key: The key that is used for storing the answer in a dict, when
```

```
                    multiple questions are asked in a set.
        :param question: The question to ask.
        :param validators: The answer is passed through the list of validators
until
                           one fails or they all pass. Validators may also
convert the
                           answer to another form, such as from a str to an int.
        """
        self.key = key
        self.question = question
        self.validators = Question.non_empty, *validators

    @staticmethod
    def ask_questions(questions):
        """
        Asks a set of questions and stores the answers in a dict.

        :param questions: The list of questions to ask.
        :return: A dict of answers.
        """
        answers = {}
        for question in questions:
            answers[question.key] = Question.ask_question(
                question.question, *question.validators
            )
        return answers

    @staticmethod
    def ask_question(question, *validators):
        """
        Asks a single question and validates it against a list of validators.
        When an answer fails validation, the complaint is printed and the
question
        is asked again.

        :param question: The question to ask.
        :param validators: The list of validators that the answer must pass.
        :return: The answer, converted to its final form by the validators.
        """
        answer = None
        while answer is None:
            answer = input(question)
            for validator in validators:
                answer, complaint = validator(answer)
```

```
                if answer is None:
                    print(complaint)
                    break
        return answer

    @staticmethod
    def non_empty(answer):
        """
        Validates that the answer is not empty.
        :return: The non-empty answer, or None.
        """
        return answer if answer != "" else None, "I need an answer. Please?"

    @staticmethod
    def is_yesno(answer):
        """
        Validates a yes/no answer.
        :return: True when the answer is 'y'; otherwise, False.
        """
        return answer.lower() == "y", ""

    @staticmethod
    def is_int(answer):
        """
        Validates that the answer can be converted to an int.
        :return: The int answer; otherwise, None.
        """
        try:
            int_answer = int(answer)
        except ValueError:
            int_answer = None
        return int_answer, f"{answer} must be a valid integer."

    @staticmethod
    def is_letter(answer):
        """
        Validates that the answer is a letter.
        :return The letter answer, converted to uppercase; otherwise, None.
        """
        return (
            answer.upper() if answer.isalpha() else None,
            f"{answer} must be a single letter.",
        )
```

```
    @staticmethod
    def is_float(answer):
        """
        Validate that the answer can be converted to a float.
        :return The float answer; otherwise, None.
        """
        try:
            float_answer = float(answer)
        except ValueError:
            float_answer = None
        return float_answer, f"{answer} must be a valid float."

    @staticmethod
    def in_range(lower, upper):
        """
        Validate that the answer is within a range. The answer must be of a type
that can
        be compared to the lower and upper bounds.
        :return: The answer, if it is within the range; otherwise, None.
        """

        def _validate(answer):
            return (
                answer if lower <= answer <= upper else None,
                f"{answer} must be between {lower} and {upper}.",
            )

        return _validate
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem

- Query
- Scan
- UpdateItem

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDB 테이블을 캡슐화하는 클래스를 생성합니다.

```
# Creates an Amazon DynamoDB table that can be used to store movie data.
# The table uses the release year of the movie as the partition key and the
# title as the sort key.
#
# @param table_name [String] The name of the table to create.
# @return [Aws::DynamoDB::Table] The newly created table.
def create_table(table_name)
  @table = @dynamo_resource.create_table(
    table_name: table_name,
    key_schema: [
      { attribute_name: 'year', key_type: 'HASH' }, # Partition key
      { attribute_name: 'title', key_type: 'RANGE' } # Sort key
    ],
    attribute_definitions: [
      { attribute_name: 'year', attribute_type: 'N' },
      { attribute_name: 'title', attribute_type: 'S' }
    ],
    billing_mode: 'PAY_PER_REQUEST'
  )
  @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
  @table
rescue Aws::DynamoDB::Errors::ServiceError => e
  @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
  raise
end
```

헬퍼 함수를 생성하여 샘플 JSON 파일을 다운로드하고 추출합니다.

```
  # Gets sample movie data, either from a local file or by first downloading it
 from
  # the Amazon DynamoDB Developer Guide.
  #
  # @param movie_file_name [String] The local file name where the movie data is
 stored in JSON format.
  # @return [Hash] The movie data as a Hash.
  def fetch_movie_data(movie_file_name)
    if !File.file?(movie_file_name)
      @logger.debug("Downloading #{movie_file_name}...")
      movie_content = URI.open(
        'https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip'
      )
      movie_json = ''
      Zip::File.open_buffer(movie_content) do |zip|
        zip.each do |entry|
          movie_json = entry.get_input_stream.read
        end
      end
    else
      movie_json = File.read(movie_file_name)
    end
    movie_data = JSON.parse(movie_json)
    # The sample file lists over 4000 movies. This returns only the first 250.
    movie_data.slice(0, 250)
  rescue StandardError => e
    puts("Failure downloading movie data:\n#{e}")
    raise
  end
```

대화식 시나리오를 실행하여 테이블을 생성하고 테이블에 대한 작업을 수행합니다.

```
  table_name = "doc-example-table-movies-#{rand(10**4)}"
  scaffold = Scaffold.new(table_name)
  dynamodb_wrapper = DynamoDBBasics.new(table_name)

  new_step(1, 'Create a new DynamoDB table if none already exists.')
```

```
 unless scaffold.exists?(table_name)
   puts("\nNo such table: #{table_name}. Creating it...")
   scaffold.create_table(table_name)
   print "Done!\n".green
 end

 new_step(2, 'Add a new record to the DynamoDB table.')
 my_movie = {}
 my_movie[:title] = CLI::UI::Prompt.ask('Enter the title of a movie to add to
the table. E.g. The Matrix')
 my_movie[:year] = CLI::UI::Prompt.ask('What year was it released? E.g.
1989').to_i
 my_movie[:rating] = CLI::UI::Prompt.ask('On a scale of 1 - 10, how do you rate
it? E.g. 7').to_i
 my_movie[:plot] = CLI::UI::Prompt.ask('Enter a brief summary of the plot. E.g.
A man awakens to a new reality.')
 dynamodb_wrapper.add_item(my_movie)
 puts("\nNew record added:")
 puts JSON.pretty_generate(my_movie).green
 print "Done!\n".green

 new_step(3, 'Update a record in the DynamoDB table.')
 my_movie[:rating] = CLI::UI::Prompt.ask("Let's update the movie you added with
a new rating, e.g. 3:").to_i
 response = dynamodb_wrapper.update_item(my_movie)
 puts("Updated '#{my_movie[:title]}' with new attributes:")
 puts JSON.pretty_generate(response).green
 print "Done!\n".green

 new_step(4, 'Get a record from the DynamoDB table.')
 puts("Searching for #{my_movie[:title]} (#{my_movie[:year]})...")
 response = dynamodb_wrapper.get_item(my_movie[:title], my_movie[:year])
 puts JSON.pretty_generate(response).green
 print "Done!\n".green

 new_step(5, 'Write a batch of items into the DynamoDB table.')
 download_file = 'moviedata.json'
 puts("Downloading movie database to #{download_file}...")
 movie_data = scaffold.fetch_movie_data(download_file)
 puts("Writing movie data from #{download_file} into your table...")
 scaffold.write_batch(movie_data)
 puts("Records added: #{movie_data.length}.")
 print "Done!\n".green
```

```
 new_step(5, 'Query for a batch of items by key.')
 loop do
   release_year = CLI::UI::Prompt.ask('Enter a year between 1972 and 2018, e.g.
1999:').to_i
   results = dynamodb_wrapper.query_items(release_year)
   if results.any?
     puts("There were #{results.length} movies released in #{release_year}:")
     results.each do |movie|
       print "\t #{movie['title']}".green
     end
     break
   else
     continue = CLI::UI::Prompt.ask("Found no movies released in
#{release_year}! Try another year? (y/n)")
     break unless continue.eql?('y')
   end
 end
 print "\nDone!\n".green

 new_step(6, 'Scan for a batch of items using a filter expression.')
 years = {}
 years[:start] = CLI::UI::Prompt.ask('Enter a starting year between 1972 and
2018:')
 years[:end] = CLI::UI::Prompt.ask('Enter an ending year between 1972 and
2018:')
 releases = dynamodb_wrapper.scan_items(years)
 if !releases.empty?
   puts("Found #{releases.length} movies.")
   count = Question.ask(
     'How many do you want to see? ', method(:is_int), in_range(1,
releases.length)
   )
   puts("Here are your #{count} movies:")
   releases.take(count).each do |release|
     puts("\t#{release['title']}")
   end
 else
   puts("I don't know about any movies released between #{years[:start]} "\
        "and #{years[:end]}.")
 end
 print "\nDone!\n".green

 new_step(7, 'Delete an item from the DynamoDB table.')
```

```
  answer = CLI::UI::Prompt.ask("Do you want to remove '#{my_movie[:title]}'? (y/
n) ")
  if answer.eql?('y')
    dynamodb_wrapper.delete_item(my_movie[:title], my_movie[:year])
    puts("Removed '#{my_movie[:title]}' from the table.")
    print "\nDone!\n".green
  end

  new_step(8, 'Delete the DynamoDB table.')
  answer = CLI::UI::Prompt.ask('Delete the table? (y/n)')
  if answer.eql?('y')
    scaffold.delete_table
    puts("Deleted #{table_name}.")
  else
    puts("Don't forget to delete the table when you're done!")
  end
  print "\nThanks for watching!\n".green
rescue Aws::Errors::ServiceError
  puts('Something went wrong with the demo.')
rescue Errno::ENOENT
  true
end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

## SAP ABAP

### SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    " Create an Amazon Dynamo DB table.

    TRY.
        DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
        DATA(lo_dyn) = /aws1/cl_dyn_factory=>create( lo_session ).
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        DATA(oo_result) = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
```

```
      " It throws exception if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
 { lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.

    " Describe table
    TRY.
        DATA(lo_table) = lo_dyn->describetable( iv_tablename = iv_table_name ).
        DATA(lv_tablename) = lo_table->get_table( )->ask_tablename( ).
        MESSAGE 'The table name is ' && lv_tablename TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table does not exist' TYPE 'E'.
    ENDTRY.

    " Put items into the table.
    TRY.
        DATA(lo_resp_putitem) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1975' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.5' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s = 'Star
 Wars' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1978' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
```

```
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '8.1' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Speed' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1994' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.9' }| ) ) )
          ) ).
        " TYPE REF TO ZCL_AWS1_dyn_PUT_ITEM_OUTPUT
        MESSAGE '3 rows inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Get item from table.
    TRY.
        DATA(lo_resp_getitem) = lo_dyn->getitem(
          iv_tablename               = iv_table_name
          it_key                     = VALUE /aws1/cl_dynattributevalue=>tt_key(
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ) ).
        DATA(lt_attr) = lo_resp_getitem->get_item( ).
        DATA(lo_title) = lt_attr[ key = 'title' ]-value.
        DATA(lo_year) = lt_attr[ key = 'year' ]-value.
        DATA(lo_rating) = lt_attr[ key = 'year' ]-value.
```

```
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Query item from table.
    TRY.
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
             ( NEW /aws1/cl_dynattributevalue( iv_n = '1975' ) ) ).
        DATA(lt_keyconditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
            it_attributevaluelist = lt_attributelist
            iv_comparisonoperator = |EQ|
          ) ) ) ).
        DATA(lo_query_result) = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_keyconditions ).
        DATA(lt_items) = lo_query_result->get_items( ).
        READ TABLE lo_query_result->get_items( ) INTO DATA(lt_item) INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Scan items from table.
    TRY.
        DATA(lo_scan_result) = lo_dyn->scan( iv_tablename = iv_table_name ).
        lt_items = lo_scan_result->get_items( ).
        " Read the first item and display the attributes.
        READ TABLE lo_query_result->get_items( ) INTO lt_item INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
```

```
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Update items from table.
    TRY.
        DATA(lt_attributeupdates) = VALUE /aws1/
cl_dynattrvalueupdate=>tt_attributeupdates(
          ( VALUE /aws1/cl_dynattrvalueupdate=>ts_attributeupdates_maprow(
          key = 'rating' value = NEW /aws1/cl_dynattrvalueupdate(
            io_value  = NEW /aws1/cl_dynattributevalue( iv_n = '7.6' )
            iv_action = |PUT| ) ) ) ).
        DATA(lt_key) = VALUE /aws1/cl_dynattributevalue=>tt_key(
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 '1980' ) ) ) ).
        DATA(lo_resp) = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = lt_key
          it_attributeupdates = lt_attributeupdates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Delete table.
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
```

```
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted as it is in use' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

SDK for Swift에 대한 DynamoDB 호출을 처리하는 Swift 클래스입니다.

```
import AWSDynamoDB
import Foundation

/// An enumeration of error codes representing issues that can arise when using
/// the `MovieTable` class.
enum MoviesError: Error {
    /// The specified table wasn't found or couldn't be created.
    case TableNotFound
```

```
    /// The specified item wasn't found or couldn't be created.
    case ItemNotFound
    /// The Amazon DynamoDB client is not properly initialized.
    case UninitializedClient
    /// The table status reported by Amazon DynamoDB is not recognized.
    case StatusUnknown
    /// One or more specified attribute values are invalid or missing.
    case InvalidAttributes
}

/// A class representing an Amazon DynamoDB table containing movie
/// information.
public class MovieTable {
    var ddbClient: DynamoDBClient?
    let tableName: String

    /// Create an object representing a movie table in an Amazon DynamoDB
    /// database.
    ///
    /// - Parameters:
    ///   - region: The optional Amazon Region to create the database in.
    ///   - tableName: The name to assign to the table. If not specified, a
    ///     random table name is generated automatically.
    ///
    /// > Note: The table is not necessarily available when this function
    /// returns. Use `tableExists()` to check for its availability, or
    /// `awaitTableActive()` to wait until the table's status is reported as
    /// ready to use by Amazon DynamoDB.
    ///
    init(region: String? = nil, tableName: String) async throws {
        do {
            let config = try await DynamoDBClient.DynamoDBClientConfiguration()
            if let region = region {
                config.region = region
            }

            self.ddbClient = DynamoDBClient(config: config)
            self.tableName = tableName

            try await self.createTable()
        } catch {
            print("ERROR: ", dump(error, name: "Initializing Amazon
 DynamoDBClient client"))
            throw error
```

```
        }
    }

    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }


    /// Check to see if the table exists online yet.
    ///
    /// - Returns: `true` if the table exists, or `false` if not.
    ///
    func tableExists() async throws -> Bool {
```

```
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DescribeTableInput(
                tableName: tableName
            )
            let output = try await client.describeTable(input: input)
            guard let description = output.table else {
                throw MoviesError.TableNotFound
            }

            return description.tableName == self.tableName
        } catch {
            print("ERROR: tableExists:", dump(error))
            throw error
        }
    }


    ///
    /// Waits for the table to exist and for its status to be active.
    ///
    func awaitTableActive() async throws {
        while try (await self.tableExists() == false) {
            do {
                let duration = UInt64(0.25 * 1_000_000_000) // Convert .25
seconds to nanoseconds.
                try await Task.sleep(nanoseconds: duration)
            } catch {
                print("Sleep error:", dump(error))
            }
        }

        while try (await self.getTableStatus() != .active) {
            do {
                let duration = UInt64(0.25 * 1_000_000_000) // Convert .25
seconds to nanoseconds.
                try await Task.sleep(nanoseconds: duration)
            } catch {
                print("Sleep error:", dump(error))
            }
        }
```

```
    }


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }


    /// Get the table's status.
    ///
    /// - Returns: The table status, as defined by the
    ///   `DynamoDBClientTypes.TableStatus` enum.
    ///
    func getTableStatus() async throws -> DynamoDBClientTypes.TableStatus {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DescribeTableInput(
                tableName: self.tableName
            )
            let output = try await client.describeTable(input: input)
            guard let description = output.table else {
                throw MoviesError.TableNotFound
            }
            guard let status = description.tableStatus else {
                throw MoviesError.StatusUnknown
            }
```

```
            return status
        } catch {
            print("ERROR: getTableStatus:", dump(error))
            throw error
        }
    }


    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Create a Swift `URL` and use it to load the file into a `Data`
            // object. Then decode the JSON into an array of `Movie` objects.

            let fileUrl = URL(fileURLWithPath: jsonPath)
            let jsonData = try Data(contentsOf: fileUrl)

            var movieList = try JSONDecoder().decode([Movie].self, from:
jsonData)

            // Truncate the list to the first 200 entries or so for this example.

            if movieList.count > 200 {
                movieList = Array(movieList[...199])
            }

            // Before sending records to the database, break the movie list into
            // 25-entry chunks, which is the maximum size of a batch item
request.

            let count = movieList.count
            let chunks = stride(from: 0, to: count, by: 25).map {
                Array(movieList[$0 ..< Swift.min($0 + 25, count)])
            }

            // For each chunk, create a list of write request records and
populate
```

```
            // them with `PutRequest` requests, each specifying one movie from
the
            // chunk. Once the chunk's items are all in the `PutRequest` list,
            // send them to Amazon DynamoDB using the
            // `DynamoDBClient.batchWriteItem()` function.

            for chunk in chunks {
                var requestList: [DynamoDBClientTypes.WriteRequest] = []

                for movie in chunk {
                    let item = try await movie.getAsItem()
                    let request = DynamoDBClientTypes.WriteRequest(
                        putRequest: .init(
                            item: item
                        )
                    )
                    requestList.append(request)
                }

                let input = BatchWriteItemInput(requestItems: [tableName:
requestList])
                _ = try await client.batchWriteItem(input: input)
            }
        } catch {
            print("ERROR: populate:", dump(error))
            throw error
        }
    }


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
            let item = try await movie.getAsItem()
```

```
            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    /// Given a movie's details, add a movie to the Amazon DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title as a `String`.
    ///   - year: The release year of the movie (`Int`).
    ///   - rating: The movie's rating if available (`Double`; default is
    ///     `nil`).
    ///   - plot: A summary of the movie's plot (`String`; default is `nil`,
    ///     indicating no plot summary is available).
    ///
    func add(title: String, year: Int, rating: Double? = nil,
             plot: String? = nil) async throws
    {
        do {
            let movie = Movie(title: title, year: year, rating: rating, plot:
plot)
            try await self.add(movie: movie)
        } catch {
            print("ERROR: add with fields:", dump(error))
            throw error
        }
    }


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
```

```
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }


    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
```

```
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }


    /// Return an array of `Movie` objects released in the specified range of
    /// years.
    ///
    /// - Parameters:
    ///   - firstYear: The first year of movies to return.
```

```
    ///   - lastYear: The last year of movies to return.
    ///   - startKey: A starting point to resume processing; always use `nil`.
    ///
    /// - Returns: An array of `Movie` objects describing the matching movies.
    ///
    /// > Note: The `startKey` parameter is used by this function when
    ///   recursively calling itself, and should always be `nil` when calling
    ///   directly.
    ///
    func getMovies(firstYear: Int, lastYear: Int,
                   startKey: [Swift.String: DynamoDBClientTypes.AttributeValue]?
= nil)
        async throws -> [Movie]
    {
        do {
            var movieList: [Movie] = []

            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = ScanInput(
                consistentRead: true,
                exclusiveStartKey: startKey,
                expressionAttributeNames: [
                    "#y": "year" // `year` is a reserved word, so use `#y`
instead.
                ],
                expressionAttributeValues: [
                    ":y1": .n(String(firstYear)),
                    ":y2": .n(String(lastYear))
                ],
                filterExpression: "#y BETWEEN :y1 AND :y2",
                tableName: self.tableName
            )

            let pages = client.scanPaginated(input: input)

            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }
```

```
                // Build an array of `Movie` objects for the returned items.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList

        } catch {
            print("ERROR: getMovies with scan:", dump(error))
            throw error
        }
    }


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
[:]
```

```
            if rating != nil {
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }


    /// Delete a movie, given its title and release year.
    ///
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
```

```
    ///
    func delete(title: String, year: Int) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            _ = try await client.deleteItem(input: input)
        } catch {
            print("ERROR: delete:", dump(error))
            throw error
        }
    }
}
```

MovieTable 클래스에서 동영상을 나타내는 데 사용하는 구조입니다.

```
import Foundation
import AWSDynamoDB

/// The optional details about a movie.
public struct Details: Codable {
    /// The movie's rating, if available.
    var rating: Double?
    /// The movie's plot, if available.
    var plot: String?
}

/// A structure describing a movie. The `year` and `title` properties are
/// required and are used as the key for Amazon DynamoDB operations. The
/// `info` sub-structure's two properties, `rating` and `plot`, are optional.
public struct Movie: Codable {
    /// The year in which the movie was released.
    var year: Int
```

```
    /// The movie's title.
    var title: String
    /// A `Details` object providing the optional movie rating and plot
    /// information.
    var info: Details

    /// Create a `Movie` object representing a movie, given the movie's
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - rating: The movie's rating (optional `Double`).
    ///   - plot: The movie's plot (optional `String`)
    init(title: String, year: Int, rating: Double? = nil, plot: String? = nil) {
        self.title = title
        self.year = year

        self.info = Details(rating: rating, plot: plot)
    }

    /// Create a `Movie` object representing a movie, given the movie's
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - info: The optional rating and plot information for the movie in a
    ///     `Details` object.
    init(title: String, year: Int, info: Details?){
        self.title = title
        self.year = year

        if info != nil {
            self.info = info!
        } else {
            self.info = Details(rating: nil, plot: nil)
        }
    }

    ///
    /// Return a new `MovieTable` object, given an array mapping string to Amazon
    /// DynamoDB attribute values.
    ///
```

```
    /// - Parameter item: The item information provided to the form used by
    ///   DynamoDB. This is an array of strings mapped to
    ///   `DynamoDBClientTypes.AttributeValue` values.
    init(withItem item: [Swift.String:DynamoDBClientTypes.AttributeValue]) throws
{
        // Read the attributes.

        guard let titleAttr = item["title"],
              let yearAttr = item["year"] else {
            throw MoviesError.ItemNotFound
        }
        let infoAttr = item["info"] ?? nil

        // Extract the values of the title and year attributes.

        if case .s(let titleVal) = titleAttr {
            self.title = titleVal
        } else {
            throw MoviesError.InvalidAttributes
        }

        if case .n(let yearVal) = yearAttr {
            self.year = Int(yearVal)!
        } else {
            throw MoviesError.InvalidAttributes
        }

        // Extract the rating and/or plot from the `info` attribute, if
        // they're present.

        var rating: Double? = nil
        var plot: String? = nil

        if infoAttr != nil, case .m(let infoVal) = infoAttr {
            let ratingAttr = infoVal["rating"] ?? nil
            let plotAttr = infoVal["plot"] ?? nil

            if ratingAttr != nil, case .n(let ratingVal) = ratingAttr {
                rating = Double(ratingVal) ?? nil
            }
            if plotAttr != nil, case .s(let plotVal) = plotAttr {
                plot = plotVal
            }
        }
```

```
        self.info = Details(rating: rating, plot: plot)
    }

    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
}
```

MovieTable 클래스를 사용하여 DynamoDB 데이터베이스에 액세스하는 프로그램입니다.

```
import ArgumentParser
import ClientRuntime
import Foundation

import AWSDynamoDB


@testable import MovieList

extension String {
    // Get the directory if the string is a file path.
    func directory() -> String {
        guard let lastIndex = lastIndex(of: "/") else {
            print("Error: String directory separator not found.")
            return ""
        }
        return String(self[...lastIndex])
    }
}

struct ExampleCommand: ParsableCommand {
    @Argument(help: "The path of the sample movie data JSON file.")
    var jsonPath: String = #file.directory() + "../../../../../resources/
sample_files/movies.json"

    @Option(help: "The AWS Region to run AWS API calls in.")
    var awsRegion: String?

    @Option(
        help: ArgumentHelp("The level of logging for the Swift SDK to perform."),
        completion: .list([
            "critical",
            "debug",
            "error",
            "info",
            "notice",
            "trace",
            "warning"
        ])
    )
    var logLevel: String = "error"

    /// Configuration details for the command.
```

```
    static var configuration = CommandConfiguration(
        commandName: "basics",
        abstract: "A basic scenario demonstrating the usage of Amazon DynamoDB.",
        discussion: """
        An example showing how to use Amazon DynamoDB to perform a series of
        common database activities on a simple movie database.
        """
    )

    /// Called by ``main()`` to asynchronously run the AWS example.
    func runAsync() async throws {
        print("Welcome to the AWS SDK for Swift basic scenario for Amazon
DynamoDB!")

        //========================================================================
        // 1. Create the table. The Amazon DynamoDB table is represented by
        //    the `MovieTable` class.
        //========================================================================

        let tableName = "ddb-movies-sample-\(Int.random(in: 1 ... Int.max))"

        print("Creating table \"\(tableName)\"...")

        let movieDatabase = try await MovieTable(region: awsRegion,
                                                 tableName: tableName)

        print("\nWaiting for table to be ready to use...")
        try await movieDatabase.awaitTableActive()

        //========================================================================
        // 2. Add a movie to the table.
        //========================================================================

        print("\nAdding a movie...")
        try await movieDatabase.add(title: "Avatar: The Way of Water", year:
2022)
        try await movieDatabase.add(title: "Not a Real Movie", year: 2023)

        //========================================================================
        // 3. Update the plot and rating of the movie using an update
        //    expression.
        //========================================================================

        print("\nAdding details to the added movie...")
```

```
        _ = try await movieDatabase.update(title: "Avatar: The Way of Water",
year: 2022,
                                           rating: 9.2, plot: "It's a sequel.")

        //===========================================================================
        // 4. Populate the table from the JSON file.
        //===========================================================================

        print("\nPopulating the movie database from JSON...")
        try await movieDatabase.populate(jsonPath: jsonPath)

        //===========================================================================
        // 5. Get a specific movie by key. In this example, the key is a
        //    combination of `title` and `year`.
        //===========================================================================

        print("\nLooking for a movie in the table...")
        let gotMovie = try await movieDatabase.get(title: "This Is the End",
year: 2013)

        print("Found the movie \"\(gotMovie.title)\", released in
\(gotMovie.year).")
        print("Rating: \(gotMovie.info.rating ?? 0.0).")
        print("Plot summary: \(gotMovie.info.plot ?? "None.")")

        //===========================================================================
        // 6. Delete a movie.
        //===========================================================================

        print("\nDeleting the added movie...")
        try await movieDatabase.delete(title: "Avatar: The Way of Water", year:
2022)

        //===========================================================================
        // 7. Use a query with a key condition expression to return all movies
        //    released in a given year.
        //===========================================================================

        print("\nGetting movies released in 1994...")
        let movieList = try await movieDatabase.getMovies(fromYear: 1994)
        for movie in movieList {
            print("    \(movie.title)")
        }
```

```
        //====================================================================
        // 8. Use `scan()` to return movies released in a range of years.
        //====================================================================

        print("\nGetting movies released between 1993 and 1997...")
        let scannedMovies = try await movieDatabase.getMovies(firstYear: 1993,
 lastYear: 1997)
        for movie in scannedMovies {
            print("    \(movie.title) (\(movie.year))")
        }

        //====================================================================
        // 9. Delete the table.
        //====================================================================

        print("\nDeleting the table...")
        try await movieDatabase.deleteTable()
    }
}

@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- API 세부 정보는 AWS SDK for Swift API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable

- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용한 DynamoDB에 대한 작업

다음 코드 예제에서는 AWS SDK를 통해 개별 DynamoDB 작업을 수행하는 방법을 보여줍니다. 각 예제에는 GitHub에 대한 링크가 포함되어 있습니다. 여기에서 코드 설정 및 실행에 대한 지침을 찾을 수 있습니다.

이들 발췌문은 DynamoDB API를 호출하며, 컨텍스트에서 실행되어야 하는 더 큰 프로그램에서 발췌한 코드입니다. AWS SDK를 사용한 DynamoDB 관련 시나리오 에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

다음 예제에는 가장 일반적으로 사용되는 작업만 포함되어 있습니다. 전체 목록은 Amazon DynamoDB API 참조를 참조하세요.

예시

- AWS SDK와 함께 BatchExecuteStatement사용
- AWS SDK 또는 CLI와 함께 BatchGetItem 사용
- AWS SDK 또는 CLI와 함께 BatchWriteItem 사용
- AWS SDK 또는 CLI와 함께 CreateTable 사용
- AWS SDK 또는 CLI와 함께 DeleteItem 사용
- AWS SDK 또는 CLI와 함께 DeleteTable 사용
- AWS SDK 또는 CLI와 함께 DescribeTable 사용
- AWS SDK 또는 CLI와 함께 DescribeTimeToLive 사용
- AWS SDK와 함께 ExecuteStatement사용
- AWS SDK 또는 CLI와 함께 GetItem 사용
- AWS SDK 또는 CLI와 함께 ListTables 사용

- AWS SDK 또는 CLI와 함께 PutItem 사용
- AWS SDK 또는 CLI와 함께 Query 사용
- AWS SDK 또는 CLI와 함께 Scan 사용
- AWS SDK 또는 CLI와 함께 UpdateItem 사용
- AWS SDK 또는 CLI와 함께 UpdateTable 사용
- AWS SDK 또는 CLI와 함께 UpdateTimeToLive 사용

## AWS SDK와 함께 **BatchExecuteStatement**사용

다음 코드 예시는 BatchExecuteStatement의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- PartiQL DELETE를 사용하여 데이터 삭제
- PartiQL INSERT를 사용하여 데이터 삽입
- PartiQL 문 배치를 사용하여 테이블 쿼리
- PartiQL SELECT를 사용하여 데이터 쿼리
- PartiQL UPDATE를 사용하여 데이터 업데이트

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

INSERT 문 배치를 사용하여 항목을 추가합니다.

```
        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
```

```
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
```

```
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
```

```
            }
            else
            {
                return null!;
            }
        }
```

SELECT 문 배치를 사용하여 항목을 가져옵니다.

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
 = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
```

```
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    if (r.Item.Any())
                    {

Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                    }
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }
```

UPDATE 문 배치를 사용하여 항목을 업데이트합니다.

```
        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
```

```
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
 = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
```

```
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

DELETE 문 배치를 사용하여 항목을 삭제합니다.

```
        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
 year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
```

```
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 BatchExecuteStatement를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

INSERT 문 배치를 사용하여 항목을 추가합니다.

```
    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
              << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
```

```
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

            // Create attribute for the info map.
            Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
= Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            ratingAttribute->SetN(ratings[i]);
            infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            plotAttribute->SetS(plots[i]);
            infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
            attributes.push_back(infoMapAttribute);
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
```

SELECT 문 배치를 사용하여 항목을 가져옵니다.

```
   // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
   {
       Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
               titles.size());
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);
           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
           statements[i].SetParameters(attributes);
       }

       Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

       request.SetStatements(statements);

       Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
               request);
       if (outcome.IsSuccess()) {
           const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

           const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

           for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
               const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

               printMovieInfo(item);
           }
       }
```

```
        else {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

UPDATE 문 배치를 사용하여 항목을 업데이트합니다.

```
   // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

   for (size_t i = 0; i < titles.size(); ++i) {
       ratings[i] = askQuestionForFloatRange(
               Aws::String("\nLet's update your the movie, \"") + titles[i] +
               ".\nYou rated it  " + std::to_string(ratings[i])
               + ", what new rating would you give it? ", 1, 10);
   }

   std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

   {
       Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
               titles.size());

       std::stringstream sqlStream;
       sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                 << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                 << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";


       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);

           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));
```

```
attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);
        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

DELETE 문 배치를 사용하여 항목을 삭제합니다.

```
    // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;
```

```
        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
 dynamoClient.BatchExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movies: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 BatchExecuteStatement를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

예제를 위한 함수 수신기 구문을 정의합니다.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
```

```
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

INSERT 문 배치를 사용하여 항목을 추가합니다.

```
// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
 return err
}
```

SELECT 문 배치를 사용하여 항목을 가져옵니다.

```
// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
```

```
 }
 return outMovies, err
}
```

UPDATE 문 배치를 사용하여 항목을 업데이트합니다.

```
// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

DELETE 문 배치를 사용하여 항목을 삭제합니다.

```
// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
```

```
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 BatchExecuteStatement를 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

PartiQL을 사용하여 항목 배치를 생성합니다.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL을 사용하여 항목 배치를 가져옵니다.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
```

```
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL을 사용하여 항목 배치를 업데이트합니다.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
```

```
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL을 사용하여 항목 배치를 삭제합니다.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
  });

  const response = await docClient.send(command);
```

```
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 BatchExecuteStatement를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }

        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function insertItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
```

```
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 BatchExecuteStatement를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
 with
                           each statement. This list must be in the same order as
 the
                           statements.
        :return: The responses returned from running the statements, if any.
        """
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
```

```
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
table "
                    "does not exist."
                )
            else:
                logger.error(
                    "Couldn't execute batch of PartiQL statements. Here's why:
%s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 BatchExecuteStatement를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

PartiQL을 사용하여 항목 배치를 읽습니다.

```
class DynamoDBPartiQLBatch
  attr_reader :dynamo_resource, :table
```

```
  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Selects a batch of items from a table using PartiQL
  #
  # @param batch_titles [Array] Collection of movie titles
  # @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
  def batch_execute_select(batch_titles)
    request_items = batch_titles.map do |title, year|
      {
        statement: "SELECT * FROM \"#{@table.name}\" WHERE title=? and year=?",
        parameters: [title, year]
      }
    end
    @dynamodb.client.batch_execute_statement({ statements: request_items })
  end
```

PartiQL을 사용하여 항목 배치를 삭제합니다.

```
class DynamoDBPartiQLBatch
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Deletes a batch of items from a table using PartiQL
  #
  # @param batch_titles [Array] Collection of movie titles
  # @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
  def batch_execute_write(batch_titles)
    request_items = batch_titles.map do |title, year|
      {
        statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
        parameters: [title, year]
      }
```

```
    end
    @dynamodb.client.batch_execute_statement({ statements: request_items })
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 BatchExecuteStatement를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **BatchGetItem** 사용

다음 코드 예시는 BatchGetItem의 사용 방법을 보여 줍니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace LowLevelBatchGet
{
    public class LowLevelBatchGet
    {
        private static readonly string _table1Name = "Forum";
        private static readonly string _table2Name = "Thread";

        public static async void
 RetrieveMultipleItemsBatchGet(AmazonDynamoDBClient client)
        {
            var request = new BatchGetItemRequest
            {
```

```
                RequestItems = new Dictionary<string, KeysAndAttributes>()
            {
                { _table1Name,
                  new KeysAndAttributes
                  {
                      Keys = new List<Dictionary<string, AttributeValue> >()
                      {
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon DynamoDB"
                          } }
                          },
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon S3"
                          } }
                          }
                      }
                  }},
                {
                    _table2Name,
                    new KeysAndAttributes
                    {
                        Keys = new List<Dictionary<string, AttributeValue> >()
                        {
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 1"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 2"
                                  } }
```

```
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon S3"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "S3 Thread 1"
                                  } }
                            }
                        }
                    }
                }
            };

            BatchGetItemResponse response;
            do
            {
                Console.WriteLine("Making request");
                response = await client.BatchGetItemAsync(request);

                // Check the response.
                var responses = response.Responses; // Attribute list in the
response.

                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
                }

                // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
```

```
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue>
 attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }

 Console.WriteLine("************************************************");
        }

        static void Main()
        {
            var client = new AmazonDynamoDBClient();

            RetrieveMultipleItemsBatchGet(client);
        }
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 BatchGetItem을 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_batch_get_item
#
# This function gets a batch of items from a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the keys of the items to get.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_batch_get_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_get_item"
    echo "Get a batch of items from a DynamoDB table."
    echo " -i item  -- Path to json file containing the keys of the items to
 get."
    echo ""
  }

  while getopts "i:h" option; do
```

```
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  response=$(aws dynamodb batch-get-item \
    --request-items file://"$item")
  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-get-item operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
```

```
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 BatchGetItem을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Batch get items from different Amazon DynamoDB tables.
/*!
  \sa batchGetItem()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::batchGetItem(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::BatchGetItemRequest request;

    // Table1: Forum.
    Aws::String table1Name = "Forum";
    Aws::DynamoDB::Model::KeysAndAttributes table1KeysAndAttributes;

    // Table1: Projection expression.
    table1KeysAndAttributes.SetProjectionExpression("#n, Category, Messages,
 #v");

    // Table1: Expression attribute names.
    Aws::Http::HeaderValueCollection headerValueCollection;
    headerValueCollection.emplace("#n", "Name");
    headerValueCollection.emplace("#v", "Views");
    table1KeysAndAttributes.SetExpressionAttributeNames(headerValueCollection);

    // Table1: Set key name, type, and value to search.
    std::vector<Aws::String> nameValues = {"Amazon DynamoDB", "Amazon S3"};
    for (const Aws::String &name: nameValues) {
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
        Aws::DynamoDB::Model::AttributeValue key;
        key.SetS(name);
```

```
        keys.emplace("Name", key);
        table1KeysAndAttributes.AddKeys(keys);
    }

    Aws::Map<Aws::String, Aws::DynamoDB::Model::KeysAndAttributes> requestItems;
    requestItems.emplace(table1Name, table1KeysAndAttributes);

    // Table2: ProductCatalog.
    Aws::String table2Name = "ProductCatalog";
    Aws::DynamoDB::Model::KeysAndAttributes table2KeysAndAttributes;
    table2KeysAndAttributes.SetProjectionExpression("Title, Price, Color");

    // Table2: Set key name, type, and value to search.
    std::vector<Aws::String> idValues = {"102", "103", "201"};
    for (const Aws::String &id: idValues) {
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
        Aws::DynamoDB::Model::AttributeValue key;
        key.SetN(id);
        keys.emplace("Id", key);
        table2KeysAndAttributes.AddKeys(keys);
    }

    requestItems.emplace(table2Name, table2KeysAndAttributes);

    bool result = true;
    do {  // Use a do loop to handle pagination.
        request.SetRequestItems(requestItems);
        const Aws::DynamoDB::Model::BatchGetItemOutcome &outcome =
dynamoClient.BatchGetItem(
                request);

        if (outcome.IsSuccess()) {
            for (const auto &responsesMapEntry:
outcome.GetResult().GetResponses()) {
                Aws::String tableName = responsesMapEntry.first;
                const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &tableResults = responsesMapEntry.second;
                std::cout << "Retrieved " << tableResults.size()
                          << " responses for table '" << tableName << "'.\n"
                          << std::endl;
                if (tableName == "Forum") {

                    std::cout << "Name | Category | Message | Views" <<
std::endl;
```

```
                    for (const Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                        std::cout << item.at("Name").GetS() << " | ";
                        std::cout << item.at("Category").GetS() << " | ";
                        std::cout << (item.count("Message") == 0 ? "" : item.at(
                                "Messages").GetN()) << " | ";
                        std::cout << (item.count("Views") == 0 ? "" : item.at(
                                "Views").GetN()) << std::endl;
                    }
                }
                else {
                    std::cout << "Title | Price | Color" << std::endl;
                    for (const Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                        std::cout << item.at("Title").GetS() << " | ";
                        std::cout << (item.count("Price") == 0 ? "" : item.at(
                                "Price").GetN());
                        if (item.count("Color")) {
                            std::cout << " | ";
                            for (const
 std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> &listItem: item.at(
                                    "Color").GetL())
                                std::cout << listItem->GetS() << " ";
                        }
                        std::cout << std::endl;
                    }
                }
                std::cout << std::endl;
            }

            // If necessary, repeat request for remaining items.
            requestItems = outcome.GetResult().GetUnprocessedKeys();
        }
        else {
            std::cerr << "Batch get item failed: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
            break;
        }
    } while (!requestItems.empty());

    return result;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 BatchGetItem을 참조하세요.

CLI

AWS CLI

테이블에서 여러 항목을 검색하는 방법

다음 `batch-get-items` 예시에서는 `GetItem` 요청 3개의 배치를 사용하여 `MusicCollection` 테이블에서 여러 항목을 읽고 작업에 사용된 읽기 용량 단위 수를 요청합니다. 이 명령은 `AlbumTitle` 속성만 반환합니다.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json \
    --return-consumed-capacity TOTAL
```

`request-items.json`의 콘텐츠:

```
{
    "MusicCollection": {
        "Keys": [
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Call Me Today"}
            },
            {
                "Artist": {"S": "Acme Band"},
                "SongTitle": {"S": "Happy Day"}
            },
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Scared of My Shadow"}
            }
        ],
        "ProjectionExpression":"AlbumTitle"
    }
}
```

출력:

```
{
    "Responses": {
        "MusicCollection": [
            {
                "AlbumTitle": {
                    "S": "Somewhat Famous"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Blue Sky Blues"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Louder Than Ever"
                }
            }
        ]
    },
    "UnprocessedKeys": {},
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 1.5
        }
    ]
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 배치 작업을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 BatchGetItem을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

서비스 클라이언트를 사용하여 배치 항목을 가져오는 방법을 보여줍니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class BatchReadItems {
    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        getBatchItems(dynamoDbClient, tableName);
    }

    public static void getBatchItems(DynamoDbClient dynamoDbClient, String
 tableName) {
        // Define the primary key values for the items you want to retrieve.
```

```
        Map<String, AttributeValue> key1 = new HashMap<>();
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());

        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();

        // Make the batchGetItem request.
        BatchGetItemResponse batchGetItemResponse =
 dynamoDbClient.batchGetItem(batchGetItemRequest);

        // Extract and print the retrieved items.
        Map<String, List<Map<String, AttributeValue>>> responses =
 batchGetItemResponse.responses();
        if (responses.containsKey(tableName)) {
            List<Map<String, AttributeValue>> musicItems =
 responses.get(tableName);
            for (Map<String, AttributeValue> item : musicItems) {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            }
        } else {
            System.out.println("No items retrieved.");
        }
    }
}
```

서비스 클라이언트와 페이지네이터를 사용하여 배치 항목을 가져오는 방법을 보여줍니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class BatchGetItemsPaginator {

    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        getBatchItemsPaginator(dynamoDbClient, tableName) ;
    }

    public static void getBatchItemsPaginator(DynamoDbClient dynamoDbClient,
 String tableName) {
        // Define the primary key values for the items you want to retrieve.
        Map<String, AttributeValue> key1 = new HashMap<>();
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());
```

```
        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();

        // Use batchGetItemPaginator for paginated requests.
        dynamoDbClient.batchGetItemPaginator(batchGetItemRequest).stream()
            .flatMap(response -> response.responses().getOrDefault(tableName,
 Collections.emptyList()).stream())
            .forEach(item -> {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            });
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 BatchGetItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 BatchGet을 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { BatchGetCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
```

```
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
  });

  const response = await docClient.send(command);
  console.log(response.Responses.Books);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 BatchGetItem을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
```

```
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};

ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 BatchGetItem을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: DynamoDB 테이블 'Music' 및 'Songs'에서 노래 제목이 'Somewhere Down The Road'인 항목을 가져옵니다.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list = New-Object
 'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
 Amazon.DynamoDBv2.Model.AttributeValue]]'
```

```
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem
```

출력:

```
Name                           Value
----                           -----
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 BatchGetItem을 참조하세요.

Tools for PowerShell V5

예 1: DynamoDB 테이블 'Music' 및 'Songs'에서 노래 제목이 'Somewhere Down The Road'인 항목을 가져옵니다.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
```

```
$list = New-Object
 'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
 Amazon.DynamoDBv2.Model.AttributeValue]]'
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem
```

출력:

```
Name                           Value
----                           -----
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 BatchGetItem을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import decimal
import json
import logging
import os
import pprint
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
dynamodb = boto3.resource("dynamodb")

MAX_GET_SIZE = 100  # Amazon DynamoDB rejects a get batch larger than 100 items.


def do_batch_get(batch_keys):
    """
    Gets a batch of items from Amazon DynamoDB. Batches can contain keys from
    more than one table.

    When Amazon DynamoDB cannot process all items in a batch, a set of
 unprocessed
    keys is returned. This function uses an exponential backoff algorithm to
 retry
    getting the unprocessed keys until all are retrieved or the specified
    number of tries is reached.

    :param batch_keys: The set of keys to retrieve. A batch can contain at most
 100
                       keys. Otherwise, Amazon DynamoDB returns an error.
    :return: The dictionary of retrieved items grouped under their respective
             table names.
```

```
    """
    tries = 0
    max_tries = 5
    sleepy_time = 1  # Start with 1 second of sleep, then exponentially increase.
    retrieved = {key: [] for key in batch_keys}
    while tries < max_tries:
        response = dynamodb.batch_get_item(RequestItems=batch_keys)
        # Collect any retrieved items and retry unprocessed keys.
        for key in response.get("Responses", []):
            retrieved[key] += response["Responses"][key]
        unprocessed = response["UnprocessedKeys"]
        if len(unprocessed) > 0:
            batch_keys = unprocessed
            unprocessed_count = sum(
                [len(batch_key["Keys"]) for batch_key in batch_keys.values()]
            )
            logger.info(
                "%s unprocessed keys returned. Sleep, then retry.",
unprocessed_count
            )
            tries += 1
            if tries < max_tries:
                logger.info("Sleeping for %s seconds.", sleepy_time)
                time.sleep(sleepy_time)
                sleepy_time = min(sleepy_time * 2, 32)
        else:
            break

    return retrieved
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 BatchGetItem를 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    /// Gets an array of `Movie` objects describing all the movies in the
    /// specified list. Any movies that aren't found in the list have no
    /// corresponding entry in the resulting array.
    ///
    /// - Parameters
    ///     - keys: An array of tuples, each of which specifies the title and
    ///       release year of a movie to fetch from the table.
    ///
    /// - Returns:
    ///     - An array of `Movie` objects describing each match found in the
    ///     table.
    ///
    /// - Throws:
    ///     - `MovieError.ClientUninitialized` if the DynamoDB client has not
    ///     been initialized.
    ///     - DynamoDB errors are thrown without change.
    func batchGet(keys: [(title: String, year: Int)]) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MovieError.ClientUninitialized
            }

            var movieList: [Movie] = []
            var keyItems: [[Swift.String: DynamoDBClientTypes.AttributeValue]] =
 []

            // Convert the list of keys into the form used by DynamoDB.

            for key in keys {
```

```
                let item: [Swift.String: DynamoDBClientTypes.AttributeValue] = [
                    "title": .s(key.title),
                    "year": .n(String(key.year))
                ]
                keyItems.append(item)
            }

            // Create the input record for `batchGetItem()`. The list of
requested
            // items is in the `requestItems` property. This array contains one
            // entry for each table from which items are to be fetched. In this
            // example, there's only one table containing the movie data.
            //
            // If we wanted this program to also support searching for matches
            // in a table of book data, we could add a second `requestItem`
            // mapping the name of the book table to the list of items we want to
            // find in it.
            let input = BatchGetItemInput(
                requestItems: [
                    self.tableName: .init(
                        consistentRead: true,
                        keys: keyItems
                    )
                ]
            )

            // Fetch the matching movies from the table.

            let output = try await client.batchGetItem(input: input)

            // Get the set of responses. If there aren't any, return the empty
            // movie list.

            guard let responses = output.responses else {
                return movieList
            }

            // Get the list of matching items for the table with the name
            // `tableName`.

            guard let responseList = responses[self.tableName] else {
                return movieList
            }
```

```
            // Create `Movie` items for each of the matching movies in the table
            // and add them to the `MovieList` array.

            for response in responseList {
                try movieList.append(Movie(withItem: response))
            }

            return movieList
        } catch {
            print("ERROR: batchGet", dump(error))
            throw error
        }
    }
```

- API 세부 정보는 Swift용 AWS SDK API 참조의 BatchGetItem을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **`BatchWriteItem`** 사용

다음 코드 예시는 `BatchWriteItem`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 기본 사항 알아보기

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

영화 테이블에 항목 배치를 씁니다.

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

    // Now return the first 250 entries.
    if (allMovies != null && allMovies.Any())
    {
        moviesList = allMovies.GetRange(0, 250);
    }
    return moviesList;
}

/// <summary>
/// Writes 250 items to the movie table.
/// </summary>
/// <param name="movieFileName">A string containing the full path to
/// the JSON file containing movie data.</param>
/// <param name="tableName">The name of the table to write items to.</param>
/// <returns>A long integer value representing the number of movies
/// imported from the JSON file.</returns>
public async Task<long> BatchWriteItemsAsync(
    string movieFileName, string tableName)
{
```

```
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of
IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch
write operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
```

```
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 BatchWriteItem을 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
```

```
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}
```

### 이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
```

```
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 BatchWriteItem을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Batch write items from a JSON file.
/*!
  \sa batchWriteItem()
  \param jsonFilePath: JSON file path.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

/*
 * The input for this routine is a JSON file that you can download from the
 following URL:
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
SampleData.html.
```

```
 *
 * The JSON data uses the BatchWriteItem API request syntax. The JSON strings are
 * converted to AttributeValue objects. These AttributeValue objects will then
 generate
 * JSON strings when constructing the BatchWriteItem request, essentially
 outputting
 * their input.
 *
 * This is perhaps an artificial example, but it demonstrates the APIs.
 */

bool AwsDoc::DynamoDB::batchWriteItem(const Aws::String &jsonFilePath,
                                      const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    std::ifstream fileStream(jsonFilePath);

    if (!fileStream) {
        std::cerr << "Error: could not open file '" << jsonFilePath << "'."
                  << std::endl;
    }

    std::stringstream stringStream;
    stringStream << fileStream.rdbuf();
    Aws::Utils::Json::JsonValue jsonValue(stringStream);

    Aws::DynamoDB::Model::BatchWriteItemRequest batchWriteItemRequest;
    Aws::Map<Aws::String, Aws::Utils::Json::JsonView> level1Map =
 jsonValue.View().GetAllObjects();
    for (const auto &level1Entry: level1Map) {
        const Aws::Utils::Json::JsonView &entriesView = level1Entry.second;
        const Aws::String &tableName = level1Entry.first;
        // The JSON entries at this level are as follows:
        //  key - table name
        //  value - list of request objects
        if (!entriesView.IsListType()) {
            std::cerr << "Error: JSON file entry '"
                      << tableName << "' is not a list." << std::endl;
            continue;
        }

        Aws::Utils::Array<Aws::Utils::Json::JsonView> entries =
 entriesView.AsArray();

        Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;
```

```
        if (AwsDoc::DynamoDB::addWriteRequests(tableName, entries,
                                               writeRequests)) {
            batchWriteItemRequest.AddRequestItems(tableName, writeRequests);
        }
    }

    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::BatchWriteItemOutcome outcome =
 dynamoClient.BatchWriteItem(
            batchWriteItemRequest);

    if (outcome.IsSuccess()) {
        std::cout << "DynamoDB::BatchWriteItem was successful." << std::endl;
    }
    else {
        std::cerr << "Error with DynamoDB::BatchWriteItem. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return outcome.IsSuccess();
}

//! Convert requests in JSON format to a vector of WriteRequest objects.
/*!
  \sa addWriteRequests()
  \param tableName: Name of the table for the write operations.
  \param requestsJson: Request data in JSON format.
  \param writeRequests: Vector to receive the WriteRequest objects.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::addWriteRequests(const Aws::String &tableName,
                                        const
 Aws::Utils::Array<Aws::Utils::Json::JsonView> &requestsJson,

 Aws::Vector<Aws::DynamoDB::Model::WriteRequest> &writeRequests) {
    for (size_t i = 0; i < requestsJson.GetLength(); ++i) {
        const Aws::Utils::Json::JsonView &requestsEntry = requestsJson[i];
        if (!requestsEntry.IsObject()) {
            std::cerr << "Error: incorrect requestsEntry type "
                      << requestsEntry.WriteReadable() << std::endl;
            return false;
```

```
        }

        Aws::Map<Aws::String, Aws::Utils::Json::JsonView> requestsMap =
 requestsEntry.GetAllObjects();

        for (const auto &request: requestsMap) {
            const Aws::String &requestType = request.first;
            const Aws::Utils::Json::JsonView &requestJsonView = request.second;

            if (requestType == "PutRequest") {
                if (!requestJsonView.ValueExists("Item")) {
                    std::cerr << "Error: item key missing for requests "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }
                Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 attributes;
                if (!getAttributeObjectsMap(requestJsonView.GetObject("Item"),
                                            attributes)) {
                    std::cerr << "Error getting attributes "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }

                Aws::DynamoDB::Model::PutRequest putRequest;
                putRequest.SetItem(attributes);
                writeRequests.push_back(
                        Aws::DynamoDB::Model::WriteRequest().WithPutRequest(
                                putRequest));
            }
            else {
                std::cerr << "Error: unimplemented request type '" << requestType
                          << "'." << std::endl;
            }
        }
    }

    return true;
}

//! Generate a map of AttributeValue objects from JSON records.
/*!
  \sa getAttributeObjectsMap()
  \param jsonView: JSONView of attribute records.
```

```
  \param writeRequests: Map to receive the AttributeValue objects.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::getAttributeObjectsMap(const Aws::Utils::Json::JsonView
 &jsonView,
                                         Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue> &attributes) {
    Aws::Map<Aws::String, Aws::Utils::Json::JsonView> objectsMap =
 jsonView.GetAllObjects();
    for (const auto &entry: objectsMap) {
        const Aws::String &attributeKey = entry.first;
        const Aws::Utils::Json::JsonView &attributeJsonView = entry.second;

        if (!attributeJsonView.IsObject()) {
            std::cerr << "Error: attribute not an object "
                      << attributeJsonView.WriteReadable() << std::endl;
            return false;
        }

        attributes.emplace(attributeKey,

 Aws::DynamoDB::Model::AttributeValue(attributeJsonView));
    }

    return true;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 BatchWriteItem을 참조하세요.

## CLI

### AWS CLI

#### 테이블에 여러 항목을 추가하는 방법

다음 `batch-write-item` 예시에서는 `PutItem` 요청 3개의 배치를 사용하여 `MusicCollection` 테이블에 새 항목 3개를 추가합니다. 또한 작업에 사용된 쓰기 용량 단위 수와 작업에서 수정된 모든 항목 모음에 대한 정보도 요청합니다.

```
aws dynamodb batch-write-item \
```

```
    --request-items file://request-items.json \
    --return-consumed-capacity INDEXES \
    --return-item-collection-metrics SIZE
```

request-items.json의 콘텐츠:

```
{
    "MusicCollection": [
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
                    "SongTitle": {"S": "Call Me Today"},
                    "AlbumTitle": {"S": "Somewhat Famous"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "Acme Band"},
                    "SongTitle": {"S": "Happy Day"},
                    "AlbumTitle": {"S": "Songs About Life"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
                    "SongTitle": {"S": "Scared of My Shadow"},
                    "AlbumTitle": {"S": "Blue Sky Blues"}
                }
            }
        }
    ]
}
```

출력:

```
{
    "UnprocessedItems": {},
    "ItemCollectionMetrics": {
```

```
        "MusicCollection": [
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "No One You Know"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            },
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "Acme Band"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            }
        ]
    },
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 6.0,
            "Table": {
                "CapacityUnits": 3.0
            },
            "LocalSecondaryIndexes": {
                "AlbumTitleIndex": {
                    "CapacityUnits": 3.0
                }
            }
        }
    ]
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 배치 작업을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 BatchWriteItem을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
```

```
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx,
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
```

```
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 BatchWriteItem을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

서비스 클라이언트를 사용하여 테이블에 많은 항목을 삽입합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutRequest;
import software.amazon.awssdk.services.dynamodb.model.WriteRequest;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class BatchWriteItems {
    public static void main(String[] args){
        final String usage = """
```

```
                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        addBatchItems(dynamoDbClient, tableName);
    }

    public static void addBatchItems(DynamoDbClient dynamoDbClient, String
tableName) {
        // Specify the updates you want to perform.
        List<WriteRequest> writeRequests = new ArrayList<>();

        // Set item 1.
        Map<String, AttributeValue> item1Attributes = new HashMap<>();
        item1Attributes.put("Artist",
AttributeValue.builder().s("Artist1").build());
        item1Attributes.put("Rating", AttributeValue.builder().s("5").build());
        item1Attributes.put("Comments", AttributeValue.builder().s("Great
song!").build());
        item1Attributes.put("SongTitle",
AttributeValue.builder().s("SongTitle1").build());

writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item1Attri

        // Set item 2.
        Map<String, AttributeValue> item2Attributes = new HashMap<>();
        item2Attributes.put("Artist",
AttributeValue.builder().s("Artist2").build());
        item2Attributes.put("Rating", AttributeValue.builder().s("4").build());
        item2Attributes.put("Comments", AttributeValue.builder().s("Nice
melody.").build());
        item2Attributes.put("SongTitle",
AttributeValue.builder().s("SongTitle2").build());

writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item2Attri
```

```
        try {
            // Create the BatchWriteItemRequest.
            BatchWriteItemRequest batchWriteItemRequest =
 BatchWriteItemRequest.builder()
                .requestItems(Map.of(tableName, writeRequests))
                .build();

            // Execute the BatchWriteItem operation.
            BatchWriteItemResponse batchWriteItemResponse =
 dynamoDbClient.batchWriteItem(batchWriteItemRequest);

            // Process the response.
            System.out.println("Batch write successful: " +
 batchWriteItemResponse);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

향상된 클라이언트를 사용하여 테이블에 많은 항목을 삽입합니다.

```
import com.example.dynamodb.Customer;
import com.example.dynamodb.Music;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import
 software.amazon.awssdk.enhanced.dynamodb.model.BatchWriteItemEnhancedRequest;
import software.amazon.awssdk.enhanced.dynamodb.model.WriteBatch;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
```

```
/*
 * Before running this code example, create an Amazon DynamoDB table named
 Customer with these columns:
 *   - id - the id of the record that is the key
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table
 *
 * Also, ensure that you have set up your development environment, including your
 credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class EnhancedBatchWriteItems {
        public static void main(String[] args) {
                Region region = Region.US_EAST_1;
                DynamoDbClient ddb = DynamoDbClient.builder()
                                .region(region)
                                .build();
                DynamoDbEnhancedClient enhancedClient =
 DynamoDbEnhancedClient.builder()
                                .dynamoDbClient(ddb)
                                .build();
                putBatchRecords(enhancedClient);
                ddb.close();
        }

        public static void putBatchRecords(DynamoDbEnhancedClient enhancedClient)
 {
                try {
                        DynamoDbTable<Customer> customerMappedTable =
 enhancedClient.table("Customer",
                                        TableSchema.fromBean(Customer.class));
                        DynamoDbTable<Music> musicMappedTable =
 enhancedClient.table("Music",
                                        TableSchema.fromBean(Music.class));
                        LocalDate localDate = LocalDate.parse("2020-04-07");
                        LocalDateTime localDateTime = localDate.atStartOfDay();
                        Instant instant =
 localDateTime.toInstant(ZoneOffset.UTC);
```

```
                        Customer record2 = new Customer();
                        record2.setCustName("Fred Pink");
                        record2.setId("id110");
                        record2.setEmail("fredp@noserver.com");
                        record2.setRegistrationDate(instant);

                        Customer record3 = new Customer();
                        record3.setCustName("Susan Pink");
                        record3.setId("id120");
                        record3.setEmail("spink@noserver.com");
                        record3.setRegistrationDate(instant);

                        Customer record4 = new Customer();
                        record4.setCustName("Jerry orange");
                        record4.setId("id101");
                        record4.setEmail("jorange@noserver.com");
                        record4.setRegistrationDate(instant);

                        BatchWriteItemEnhancedRequest
batchWriteItemEnhancedRequest = BatchWriteItemEnhancedRequest
                                        .builder()
                                        .writeBatches(

WriteBatch.builder(Customer.class) // add items to the Customer

         // table

.mappedTableResource(customerMappedTable)

.addPutItem(builder -> builder.item(record2))

.addPutItem(builder -> builder.item(record3))

.addPutItem(builder -> builder.item(record4))
                                                                        .build(),

WriteBatch.builder(Music.class) // delete an item from the Music

      // table

.mappedTableResource(musicMappedTable)

.addDeleteItem(builder -> builder.key(
```

```
        Key.builder().partitionValue(

                        "Famous Band")

                        .build()))
                                                                        .build())
                                        .build();

                        // Add three items to the Customer table and delete one
 item from the Music
                        // table.

 enhancedClient.batchWriteItem(batchWriteItemEnhancedRequest);
                        System.out.println("done");

                } catch (DynamoDbException e) {
                        System.err.println(e.getMessage());
                        System.exit(1);
                }
        }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 BatchWriteItem을 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 BatchWrite를 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
```

```
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "node:fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year'
 is recommended
        // to account for duplicate titles.
        BatchWriteMoviesTable: putRequests,
      },
    });

    await docClient.send(command);
```

```
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 BatchWriteItem을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
```

```
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 BatchWriteItem을 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public function writeBatch(string $TableName, array $Batch, int $depth = 2)
    {
        if (--$depth <= 0) {
            throw new Exception("Max depth exceeded. Please try with fewer batch
 items or increase depth.");
        }

        $marshal = new Marshaler();
        $total = 0;
        foreach (array_chunk($Batch, 25) as $Items) {
            foreach ($Items as $Item) {
                $BatchWrite['RequestItems'][$TableName][] = ['PutRequest' =>
['Item' => $marshal->marshalItem($Item)]];
            }
            try {
```

```
                echo "Batching another " . count($Items) . " for a total of " .
 ($total += count($Items)) . " items!\n";
                $response = $this->dynamoDbClient->batchWriteItem($BatchWrite);
                $BatchWrite = [];
            } catch (Exception $e) {
                echo "uh oh...";
                echo $e->getMessage();
                die();
            }
            if ($total >= 250) {
                echo "250 movies is probably enough. Right? We can stop there.
\n";
                break;
            }
        }
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 BatchWriteItem을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 새 항목을 생성하거나, DynamoDB 테이블 Music 및 Songs의 새 항목으로 기존 항목을 바꿉니다.

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 10.0
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item
```

출력:

```
$requestItem = @{
```

```
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 BatchWriteItem을 참조하세요.

Tools for PowerShell V5

예 1: 새 항목을 생성하거나, DynamoDB 테이블 Music 및 Songs의 새 항목으로 기존 항목을 바꿉니다.

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 10.0
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item
```

출력:

```
$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 BatchWriteItem을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
least
                       the keys required by the schema that was specified when
the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 BatchWriteItem를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Fills an Amazon DynamoDB table with the specified data. Items are sent in
  # batches of 25 until all items are written.
  #
  # @param movies [Enumerable] The data to put in the table. Each item must
 contain at least
  #                            the keys required by the schema that was specified
 when the
  #                            table was created.
  def write_batch(movies)
    index = 0
    slice_size = 25
    while index < movies.length
      movie_items = []
      movies[index, slice_size].each do |movie|
        movie_items.append({ put_request: { item: movie } })
      end
      @dynamo_resource.client.batch_write_item({ request_items: { @table.name =>
 movie_items } })
      index += slice_size
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts(
      "Couldn't load data into table #{@table.name}. Here's why:"
    )
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 BatchWriteItem을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Create a Swift `URL` and use it to load the file into a `Data`
            // object. Then decode the JSON into an array of `Movie` objects.

            let fileUrl = URL(fileURLWithPath: jsonPath)
            let jsonData = try Data(contentsOf: fileUrl)

            var movieList = try JSONDecoder().decode([Movie].self, from:
 jsonData)

            // Truncate the list to the first 200 entries or so for this example.

            if movieList.count > 200 {
                movieList = Array(movieList[...199])
            }

            // Before sending records to the database, break the movie list into
            // 25-entry chunks, which is the maximum size of a batch item
 request.
```

```
            let count = movieList.count
            let chunks = stride(from: 0, to: count, by: 25).map {
                Array(movieList[$0 ..< Swift.min($0 + 25, count)])
            }

            // For each chunk, create a list of write request records and
populate
            // them with `PutRequest` requests, each specifying one movie from
the
            // chunk. Once the chunk's items are all in the `PutRequest` list,
            // send them to Amazon DynamoDB using the
            // `DynamoDBClient.batchWriteItem()` function.

            for chunk in chunks {
                var requestList: [DynamoDBClientTypes.WriteRequest] = []

                for movie in chunk {
                    let item = try await movie.getAsItem()
                    let request = DynamoDBClientTypes.WriteRequest(
                        putRequest: .init(
                            item: item
                        )
                    )
                    requestList.append(request)
                }

                let input = BatchWriteItemInput(requestItems: [tableName:
requestList])
                _ = try await client.batchWriteItem(input: input)
            }
        } catch {
            print("ERROR: populate:", dump(error))
            throw error
        }
    }
```

- API 세부 정보는 Swift용 AWS SDK API 참조의 BatchWriteItem을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **CreateTable** 사용

다음 코드 예시는 CreateTable의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 기본 사항 알아보기
- DAX로 읽기 가속화
- 고급 글로벌 보조 인덱스 시나리오
- 글로벌 보조 인덱스가 있는 테이블 만들기
- 웜 처리량이 활성화된 테이블 만들기
- MREC를 보여주는 글로벌 테이블 만들기 및 관리
- MRSC 글로벌 테이블 만들기 및 관리
- 글로벌 보조 인덱스 관리
- 리소스 기반 정책 관리
- 속성 기반 액세스 제어 설정
- 글로벌 테이블 및 다중 리전 복제 최종 일관성(MREC) 작업
- 로컬 보조 인덱스로 작업
- 리소스 태그 지정 작업
- Streams 및 Time-to-Live 작업
- 테이블 암호화 작업

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
```

```
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
```

```
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
{tableName}. {ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateTable을 참조하세요.

Bash

Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
```

```
   echo " -a attribute_definitions -- JSON file path of a list of attributes and
their types."
   echo " -k key_schema -- JSON file path of a list of attributes and their key
types."
   echo ""
 }

 # Retrieve the calling parameters.
 while getopts "n:a:k:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     a) attribute_definitions="${OPTARG}" ;;
     k) key_schema="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_definitions" ]]; then
   errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
   usage
   return 1
 fi

 if [[ -z "$key_schema" ]]; then
   errecho "ERROR: You must provide a key schema json file path the -k
parameter."
   usage
   return 1
```

```
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
```

```
# This function outputs everything sent to it to STDERR (standard error output).
#########################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#########################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#########################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 CreateTable을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Create an Amazon DynamoDB table.
/*!
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;

    Aws::DynamoDB::Model::CreateTableRequest request;

    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);
```

```
    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

테이블이 활성화될 때까지 대기하는 코드입니다.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
```

```
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 CreateTable을 참조하세요.

## CLI

### AWS CLI

#### 예 1: 태그가 포함된 테이블을 생성하는 방법

다음 `create-table` 예시에서는 지정된 속성과 키 스키마를 사용하여 이름이 `MusicCollection`인 테이블을 생성합니다. 이 테이블은 프로비저닝된 처리량을 사용하며, 기본 AWS 소유 CMK를 사용하여 저장 시 암호화됩니다. 이 명령은 또한 키가 `Owner`이고 값이 `blueTeam`인 태그를 테이블에 적용합니다.

```
aws dynamodb create-table \
    --table-name MusicCollection \
```

```
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
```

```
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 2: 온디맨드 모드에서 테이블을 생성하는 방법

다음 예시에서는 프로비저닝된 처리량 모드가 아닌 온디맨드 모드를 사용하여 이름이 MusicCollection인 테이블을 생성합니다. 이는 예상치 못한 워크로드가 있는 테이블에 유용합니다.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
```

```
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 3: 고객 관리형 CMK로 테이블을 생성하고 암호화하는 방법

다음 예시에서는 이름이 MusicCollection인 테이블을 만들고 고객 관리형 CMK를 사용하여 이를 암호화합니다.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 4: 로컬 보조 인덱스가 있는 테이블을 생성하는 방법

다음 `MusicCollection` 예시에서는 지정된 속성과 키 스키마를 사용하여 이름이 `AlbumTitleIndex`인 로컬 보조 인덱스가 있는 이라는 테이블을 생성합니다.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
```

```
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
```

```
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 5: 글로벌 보조 인덱스가 있는 테이블을 생성하는 방법

다음 예시에서는 이름이 GameTitleIndex인 글로벌 보조 인덱스가 있는 GameScores라는 테이블을 생성합니다. 기본 테이블은 파티션 키가 UserId이고 정렬 키가 GameTitle이므로 특정 게임의 개별 사용자 최고 점수를 효율적으로 찾을 수 있는 반면 GSI는 파티션 키가 GameTitle이고 정렬 키가 TopScore이므로 특정 게임의 전체 최고 점수를 빠르게 찾을 수 있습니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
```

```
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
```

```
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 6: 글로벌 보조 인덱스가 있는 테이블 여러 개를 한 번에 생성하는 방법

다음 예시에서는 두 개의 글로벌 보조 인덱스가 있는 GameScores라는 테이블을 생성합니다. GSI 스키마는 명령줄이 아닌 파일을 통해 전달됩니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

gsi.json의 콘텐츠:

```
[
    {
        "IndexName": "GameTitleIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
```

```
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
```

```
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
```

```
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 7: Streams가 활성화된 테이블을 생성하는 방법

다음 예시에서는 DynamoDB Streams가 활성화된 GameScores라는 테이블을 생성합니다. 각 항목의 새 이미지와 이전 이미지가 모두 스트림에 작성됩니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
```

```
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 기본 작업을 참조하세요.

예 8: Keys-Only Stream이 활성화된 테이블을 생성하는 방법

다음 예시에서는 DynamoDB Streams가 활성화된 GameScores라는 테이블을 생성합니다. 수정된 항목의 키 속성만 스트림에 작성됩니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
```

```
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB Streams에 대한 변경 데이터 캡처를 참조하세요.

예 9: Standard-Infrequent Access 클래스를 사용하는 테이블을 생성하는 방법

다음 예시에서는 이름이 GameScores인 테이블을 생성하고 Standard-Infrequent Access(DynamoDB Standard-IA) 테이블 클래스를 할당합니다. 이 테이블 클래스는 가장 비용이 많이 드는 스토리지에 최적화되어 있습니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "DeletionProtectionEnabled": false
    }
}
```


자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 클래스를 참조하세요.

예 10: 삭제 방지가 활성화된 테이블을 생성하는 방법

다음 예시에서는 이름이 GameScores인 테이블을 생성하고 삭제 방지를 활성화합니다.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
```

```
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 삭제 보호 기능 사용을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 CreateTable을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
```

```
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
```

```
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 CreateTable을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
```

```
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
    public static void main(String[] args) {
        final String usage = """

            Usage:
                <tableName> <key>

            Where:
                tableName - The Amazon DynamoDB table to create (for example,
 Music3).
                key - The key for the Amazon DynamoDB table (for example,
 Artist).
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        System.out.println("Creating an Amazon DynamoDB table " + tableName + "
 with a simple primary key: " + key);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        String result = createTable(ddb, tableName, key);
        System.out.println("New table is " + result);
        ddb.close();
    }

    public static String createTable(DynamoDbClient ddb, String tableName, String
 key) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        CreateTableRequest request = CreateTableRequest.builder()
            .attributeDefinitions(AttributeDefinition.builder()
                .attributeName(key)
```

```
                .attributeType(ScalarAttributeType.S)
                .build())
            .keySchema(KeySchemaElement.builder()
                .attributeName(key)
                .keyType(KeyType.HASH)
                .build())
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
 scales based on traffic.
            .tableName(tableName)
            .build();

        String newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription().tableName();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateTable을 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    BillingMode: "PAY_PER_REQUEST",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 CreateTable을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
```

```
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 CreateTable을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createNewTable(
    tableNameVal: String,
    key: String,
): String? {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
```

```
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef)
            keySchema = listOf(keySchemaVal)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        var tableArn: String
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 CreateTable를 참조하세요.

## PHP

### SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블을 생성합니다.

```
        $tableName = "ddb_demo_table_$uuid";
```

```
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
'WriteCapacityUnits' => 10],
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 CreateTable을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 이 예시는 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만듭니다. 테이블을 구성하는 데 사용되는 스키마는 표시된 대로 또는 -Schema 파라미터를 사용하여 지정한 대로 각 cmdlet에 파이프로 연결할 수 있습니다.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

예 2: 이 예시는 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만듭니다. 로컬 보조 인덱스도 정의됩니다. 로컬 보조 인덱스의 키는 테이블의 프라이머리 해시 키(ForumName)에서 자동으로 설정됩니다. 테이블을 구성하는 데 사용되는 스키마는 표시된 대로 또는 -Schema 파라미터를 사용하여 지정한 대로 각 cmdlet에 파이프로 연결할 수 있습니다.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

예 3: 이 예시는 단일 파이프라인을 사용하여 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만드는 방법을 보여줍니다. Add-DDBKeySchema 및 Add-DDBIndexSchema는 파이프라인 또는 -Schema 파라미터에서 TableSchema 객체가 제공되지 않는 경우 자동으로 새 TableSchema 객체를 만듭니다.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 CreateTable을 참조하세요.

Tools for PowerShell V5

예 1: 이 예시는 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만듭니다. 테이블을 구성하는 데 사용되는 스키마는 표시된 대로 또는 -Schema 파라미터를 사용하여 지정한 대로 각 cmdlet에 파이프로 연결할 수 있습니다.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, Subject}
```

```
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

예 2: 이 예시는 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만듭니다. 로컬 보조 인덱스도 정의됩니다. 로컬 보조 인덱스의 키는 테이블의 프라이머리 해시 키(ForumName)에서 자동으로 설정됩니다. 테이블을 구성하는 데 사용되는 스키마는 표시된 대로 또는 -Schema 파라미터를 사용하여 지정한 대로 각 cmdlet에 파이프로 연결할 수 있습니다.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

예 3: 이 예시는 단일 파이프라인을 사용하여 프라이머리 키가 'ForumName'(키 유형 해시) 및 'Subject'(키 유형 범위)로 구성된 Thread라는 테이블을 만드는 방법을 보여줍니다. Add-DDBKeySchema 및 Add-DDBIndexSchema는 파이프라인 또는 -Schema 파라미터에서 TableSchema 객체가 제공되지 않는 경우 자동으로 새 TableSchema 객체를 만듭니다.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
```

```
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

출력:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 CreateTable을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

영화 데이터를 저장할 테이블을 생성합니다.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
```

```
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
```

```
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 CreateTable를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end
```

```
  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        { attribute_name: 'year', key_type: 'HASH' }, # Partition key
        { attribute_name: 'title', key_type: 'RANGE' } # Sort key
      ],
      attribute_definitions: [
        { attribute_name: 'year', attribute_type: 'N' },
        { attribute_name: 'title', attribute_type: 'S' }
      ],
      billing_mode: 'PAY_PER_REQUEST'
    )
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 CreateTable을 참조하세요.

## Rust

### SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
pub async fn create_table(
    client: &Client,
```

```
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
        .attribute_name(&a_name)
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 CreateTable을 참조하세요.

## SAP ABAP

### SDK for SAP ABAP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
      " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
{ lo_resourceinuseex->av_err_msg }|.
```

```
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 CreateTable을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
 "year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
 "title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
 keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
 keyType: .range)
                ],
```

```
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }
```

- API 세부 정보는 Swift용 AWS SDK API 참조의 CreateTable을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **`DeleteItem`** 사용

다음 코드 예시는 `DeleteItem`의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 기본 사항 알아보기
- 조건부 작업 사용

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
```

```
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteItem을 참조하세요.

Bash

Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
```

```
    echo " -k keys  -- Path to json file containing the keys that identify the
item to delete."
    echo ""
 }
 while getopts "n:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
 fi

 if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
 fi

 iecho "Parameters:\n"
 iecho "    table_name:   $table_name"
 iecho "    keys:   $keys"
 iecho ""

 response=$(aws dynamodb delete-item \
    --table-name "$table_name" \
    --key file://"$keys")

 local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}
```

이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
```

```
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 DeleteItem을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Delete an item from an Amazon DynamoDB table.
```

```
/*!
  \sa deleteItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::deleteItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteItemRequest request;

    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteItemOutcome &outcome =
 dynamoClient.DeleteItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item \"" << partitionValue << "\" deleted!" << std::endl;
    }
    else {
        std::cerr << "Failed to delete item: " << outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

테이블이 활성화될 때까지 대기하는 코드입니다.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
```

```
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DeleteItem을 참조하세요.

CLI

AWS CLI

예 1: 항목을 삭제하는 방법

다음 delete-item 예시에서는 MusicCollection 테이블에서 항목을 삭제하고 삭제된 항목에 대한 세부 정보와 요청에 사용된 용량을 요청합니다.

```
aws dynamodb delete-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-values ALL_OLD \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

key.json의 콘텐츠:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Scared of My Shadow"}
}
```

출력:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Blue Sky Blues"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Scared of My Shadow"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 2.0
    },
    "ItemCollectionMetrics": {
```

```
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 쓰기를 참조하세요.

예 2: 조건부로 항목을 삭제하는 방법

다음 예시에서는 ProductCategory가 Sporting Goods 또는 Gardening Supplies이고 가격이 500에서 600 사이일 때만 ProductCatalog 테이블에서 항목을 삭제합니다. 삭제된 항목에 대한 세부 정보가 반환됩니다.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (#P
 between :lo and :hi)" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

names.json의 콘텐츠:

```
{
    "#P": "Price"
}
```

values.json의 콘텐츠:

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
```

```
}
```

출력:

```
{
    "Attributes": {
        "Id": {
            "N": "456"
        },
        "Price": {
            "N": "550"
        },
        "ProductCategory": {
            "S": "Sporting Goods"
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 쓰기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteItem을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
```

```
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DeleteItem을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyval>

                Where:
                    tableName - The Amazon DynamoDB table to delete the item from
 (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to delete
 (for example, Famous Band).
                """;
```

```
        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Deleting item \"%s\" from %s\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void deleteDynamoDBItem(DynamoDbClient ddb, String tableName,
 String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();

        try {
            ddb.deleteItem(deleteReq);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 DeleteCommand를 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteItem을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블에서 항목을 삭제합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

DynamoDB 문서 클라이언트를 사용하여 테이블에서 항목을 삭제합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteItem을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteDynamoDBItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
```

```
        DeleteItemRequest {
            tableName = tableNameVal
            key = keyToGet
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteItem(request)
        println("Item with key matching $keyVal was deleted")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteItem를 참조하세요.

## PHP

### SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

    public function deleteItemByKey(string $tableName, array $key)
    {
        $this->dynamoDbClient->deleteItem([
```

```
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 DeleteItem을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 제공된 키와 일치하는 DynamoDB 항목을 제거합니다.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 DeleteItem을 참조하세요.

Tools for PowerShell V5

예 1: 제공된 키와 일치하는 DynamoDB 항목을 제거합니다.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 DeleteItem을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

항목이 특정 기준을 충족하는 경우에만 삭제되도록 조건을 지정할 수 있습니다.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def delete_underrated_movie(self, title, year, rating):
        """
        Deletes a movie only if it is rated below a specified value. By using a
        condition expression in a delete operation, you can specify that an item
 is
        deleted only when it meets certain criteria.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        :param rating: The rating threshold to check before deleting the movie.
        """
        try:
            self.table.delete_item(
                Key={"year": year, "title": title},
                ConditionExpression="info.rating <= :val",
```

```
                ExpressionAttributeValues={":val": Decimal(str(rating))},
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't delete %s because its rating is greater than %s.",
                    title,
                    rating,
                )
            else:
                logger.error(
                    "Couldn't delete movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 DeleteItem를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end
```

```
  # Deletes a movie from the table.
  #
  # @param title [String] The title of the movie to delete.
  # @param year [Integer] The release year of the movie to delete.
  def delete_item(title, year)
    @table.delete_item(key: { 'year' => year, 'title' => title })
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete movie #{title}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 DeleteItem을 참조하세요.

## Rust

### SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
pub async fn delete_item(
    client: &Client,
    table: &str,
    key: &str,
    value: &str,
) -> Result<DeleteItemOutput, Error> {
    match client
        .delete_item()
        .table_name(table)
        .key(key, AttributeValue::S(value.into()))
        .send()
        .await
    {
        Ok(out) => {
            println!("Deleted item from table");
            Ok(out)
        }
```

```
        Err(e) => Err(Error::unhandled(e)),
    }
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DeleteItem을 참조하세요.

## SAP ABAP

### SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        DATA(lo_resp) = lo_dyn->deleteitem(
          iv_tablename                = iv_table_name
          it_key                      = it_key_input ).
        MESSAGE 'Deleted one item.' TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 DeleteItem을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    /// Delete a movie, given its title and release year.
    ///
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
    ///
    func delete(title: String, year: Int) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            _ = try await client.deleteItem(input: input)
        } catch {
            print("ERROR: delete:", dump(error))
            throw error
        }
    }
```

- API 세부 정보는 Swift용 AWS SDK API 참조의 DeleteItem을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **DeleteTable** 사용

다음 코드 예시는 DeleteTable의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 기본 사항 알아보기
- DAX로 읽기 가속화
- MRSC 글로벌 테이블 만들기 및 관리
- 글로벌 보조 인덱스 관리
- 글로벌 테이블 및 다중 리전 복제 최종 일관성(MREC) 작업

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
```

```
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteTable을 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
```

```
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
```

```
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 DeleteTable을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Delete an Amazon DynamoDB table.
/*!
  \sa deleteTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteTable(const Aws::String &tableName,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DeleteTable을 참조하세요.

CLI

AWS CLI

테이블을 삭제하는 방법

다음 `delete-table` 예시에서는 `MusicCollection` 테이블을 삭제합니다.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

출력:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 삭제를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteTable을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DeleteTable을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to delete (for example,
 Music3).

                **Warning** This program will delete the table that you specify!
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
```

```
        }

        String tableName = args[0];
        System.out.format("Deleting the Amazon DynamoDB table %s...\n",
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteTable을 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteTable을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
```

```
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteTable을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteTable를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
            ]);
        });
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 DeleteTable을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 지정된 테이블을 삭제합니다. 작업이 진행되기 전에 확인 메시지가 표시됩니다.

```
Remove-DDBTable -TableName "myTable"
```

예 2: 지정된 테이블을 삭제합니다. 작업이 진행되기 전에 확인 메시지가 표시되지 않습니다.

```
Remove-DDBTable -TableName "myTable" -Force
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 DeleteTable을 참조하세요.

Tools for PowerShell V5

예 1: 지정된 테이블을 삭제합니다. 작업이 진행되기 전에 확인 메시지가 표시됩니다.

```
Remove-DDBTable -TableName "myTable"
```

예 2: 지정된 테이블을 삭제합니다. 작업이 진행되기 전에 확인 메시지가 표시되지 않습니다.

```
Remove-DDBTable -TableName "myTable" -Force
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 DeleteTable을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
```

```
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 DeleteTable를 참조하세요.

## Ruby

### SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
```

```
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Deletes the table.
  def delete_table
    @table.delete
    @table = nil
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete table. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 DeleteTable을 참조하세요.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
pub async fn delete_table(client: &Client, table: &str) ->
 Result<DeleteTableOutput, Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
```

```
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DeleteTable을 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
TRY.
    lo_dyn->deletetable( iv_tablename = iv_table_name ).
    " Wait till the table is actually deleted.
    lo_dyn->get_waiter( )->tablenotexists(
      iv_max_wait_time = 200
      iv_tablename     = iv_table_name ).
    MESSAGE 'Table ' && iv_table_name && ' deleted.' TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
    MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
  CATCH /aws1/cx_dynresourceinuseex.
    MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 DeleteTable을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }
```

- API 세부 정보는 Swift용 AWS SDK API 참조의 DeleteTable을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **DescribeTable** 사용

다음 코드 예시는 DescribeTable의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 기본 사항 알아보기
- MREC를 보여주는 글로벌 테이블 만들기 및 관리
- MRSC 글로벌 테이블 만들기 및 관리
- 글로벌 테이블 및 다중 리전 복제 최종 일관성(MREC) 작업

- Streams 및 Time-to-Live 작업
- 테이블 암호화 작업

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
private static async Task GetTableInformation()
{
    Console.WriteLine("\n*** Retrieving table information ***");

    var response = await Client.DescribeTableAsync(new DescribeTableRequest
    {
        TableName = ExampleTableName
    });

    var table = response.Table;
    Console.WriteLine($"Name: {table.TableName}");
    Console.WriteLine($"# of items: {table.ItemCount}");

}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeTable을 참조하세요.

Bash

Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
#       1 - If it fails.
###############################################################################
function dynamodb_describe_table {
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
```

```
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  local table_status
    table_status=$(
      aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
        --query 'Table.TableStatus'
    )

   local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log "$error_code"
      errecho "ERROR: AWS reports describe-table operation failed.$table_status"
      return 1
    fi

  echo "$table_status"

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
```

```
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 DescribeTable을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Describe an Amazon DynamoDB table.
/*!
  \sa describeTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::describeTable(const Aws::String &tableName,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DescribeTableOutcome &outcome =
 dynamoClient.DescribeTable(
            request);

    if (outcome.IsSuccess()) {
        const Aws::DynamoDB::Model::TableDescription &td =
 outcome.GetResult().GetTable();
        std::cout << "Table name  : " << td.GetTableName() << std::endl;
        std::cout << "Table ARN   : " << td.GetTableArn() << std::endl;
        std::cout << "Status      : "
                  <<
 Aws::DynamoDB::Model::TableStatusMapper::GetNameForTableStatus(
                          td.GetTableStatus()) << std::endl;
        std::cout << "Item count  : " << td.GetItemCount() << std::endl;
        std::cout << "Size (bytes): " << td.GetTableSizeBytes() << std::endl;
```

```
        const Aws::DynamoDB::Model::ProvisionedThroughputDescription &ptd =
 td.GetProvisionedThroughput();
        std::cout << "Throughput" << std::endl;
        std::cout << "  Read Capacity : " << ptd.GetReadCapacityUnits() <<
 std::endl;
        std::cout << "  Write Capacity: " << ptd.GetWriteCapacityUnits() <<
 std::endl;

        const Aws::Vector<Aws::DynamoDB::Model::AttributeDefinition> &ad =
 td.GetAttributeDefinitions();
        std::cout << "Attributes" << std::endl;
        for (const auto &a: ad)
            std::cout << "  " << a.GetAttributeName() << " (" <<

 Aws::DynamoDB::Model::ScalarAttributeTypeMapper::GetNameForScalarAttributeType(
                                a.GetAttributeType()) <<
                     ")" << std::endl;
    }
    else {
        std::cerr << "Failed to describe table: " <<
 outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DescribeTable을 참조하세요.

## CLI

### AWS CLI

#### 테이블을 설명하는 방법

다음 `describe-table` 예시에서는 `MusicCollection` 테이블을 설명합니다.

```
aws dynamodb describe-table \
    --table-name MusicCollection
```

출력:

```
{
```

```
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "ACTIVE",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1421866952.062
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 설명을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DescribeTable을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
```

```
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DescribeTable을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import
 software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughputDescription;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
```

```
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to get information
 about (for example, Music3).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Getting description for %s\n\n", tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        describeDymamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void describeDymamoDBTable(DynamoDbClient ddb, String
 tableName) {
        DescribeTableRequest request = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            TableDescription tableInfo = ddb.describeTable(request).table();
            if (tableInfo != null) {
                System.out.format("Table name  : %s\n", tableInfo.tableName());
                System.out.format("Table ARN   : %s\n", tableInfo.tableArn());
                System.out.format("Status      : %s\n", tableInfo.tableStatus());
```

```
                System.out.format("Item count  : %d\n", tableInfo.itemCount());
                System.out.format("Size (bytes): %d\n",
 tableInfo.tableSizeBytes());

                ProvisionedThroughputDescription throughputInfo =
 tableInfo.provisionedThroughput();
                System.out.println("Throughput");
                System.out.format("  Read Capacity : %d\n",
 throughputInfo.readCapacityUnits());
                System.out.format("  Write Capacity: %d\n",
 throughputInfo.writeCapacityUnits());

                List<AttributeDefinition> attributes =
 tableInfo.attributeDefinitions();
                System.out.println("Attributes");
                for (AttributeDefinition a : attributes) {
                    System.out.format("  %s (%s)\n", a.attributeName(),
 a.attributeType());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("\nDone!");
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeTable을 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 DescribeTable을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 DescribeTable을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 지정된 테이블의 세부 정보를 반환합니다.

```
Get-DDBTable -TableName "myTable"
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 DescribeTable을 참조하세요.

Tools for PowerShell V5

예 1: 지정된 테이블의 세부 정보를 반환합니다.

```
Get-DDBTable -TableName "myTable"
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 DescribeTable을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def exists(self, table_name):
        """
        Determines whether a table exists. As a side effect, stores the table in
        a member variable.

        :param table_name: The name of the table to check.
        :return: True when the table exists; otherwise, False.
        """
        try:
            table = self.dyn_resource.Table(table_name)
```

```
            table.load()
            exists = True
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                exists = False
            else:
                logger.error(
                    "Couldn't check for existence of %s. Here's why: %s: %s",
                    table_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            self.table = table
        return exists
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 DescribeTable를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
```

```
end

# Determines whether a table exists. As a side effect, stores the table in
# a member variable.
#
# @param table_name [String] The name of the table to check.
# @return [Boolean] True when the table exists; otherwise, False.
def exists?(table_name)
  @dynamo_resource.client.describe_table(table_name: table_name)
  @logger.debug("Table #{table_name} exists")
rescue Aws::DynamoDB::Errors::ResourceNotFoundException
  @logger.debug("Table #{table_name} doesn't exist")
  false
rescue Aws::DynamoDB::Errors::ServiceError => e
  puts("Couldn't check for existence of #{table_name}:\n")
  puts("\t#{e.code}: #{e.message}")
  raise
end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 DescribeTable을 참조하세요.

## SAP ABAP

### SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
TRY.
    oo_result = lo_dyn->describetable( iv_tablename = iv_table_name ).
    DATA(lv_tablename) = oo_result->get_table( )->ask_tablename( ).
    DATA(lv_tablearn) = oo_result->get_table( )->ask_tablearn( ).
    DATA(lv_tablestatus) = oo_result->get_table( )->ask_tablestatus( ).
    DATA(lv_itemcount) = oo_result->get_table( )->ask_itemcount( ).
    MESSAGE 'The table name is ' && lv_tablename
        && '. The table ARN is ' && lv_tablearn
        && '. The tablestatus is ' && lv_tablestatus
        && '. Item count is ' && lv_itemcount TYPE 'I'.
```

```
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table ' && lv_tablename && ' does not exist' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 SAP ABAP용 AWS SDK API 참조의 DescribeTable을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **DescribeTimeToLive** 사용

다음 코드 예시는 DescribeTimeToLive의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- Streams 및 Time-to-Live 작업

CLI

AWS CLI

테이블의 Time to Live 설정을 보려면

다음 describe-time-to-live 예제에서는 MusicCollection 테이블의 Time to Live 설정을 표시합니다.

```
aws dynamodb describe-time-to-live \
    --table-name MusicCollection
```

출력:

```
{
    "TimeToLiveDescription": {
        "TimeToLiveStatus": "ENABLED",
        "AttributeName": "ttl"
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 Time to Live를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DescribeTimeToLive를 참조하세요.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용하여 기존 DynamoDB 테이블의 TTL 구성을 설명합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DescribeTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTimeToLiveResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.logging.Level;
import java.util.logging.Logger;

    public DescribeTimeToLiveResponse describeTTL(final String tableName, final
 Region region) {
        final DescribeTimeToLiveRequest request =
            DescribeTimeToLiveRequest.builder().tableName(tableName).build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.describeTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeTimeToLive를 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용하여 기존 DynamoDB 테이블의 TTL 구성을 설명합니다.

```
import { DynamoDBClient, DescribeTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const describeTTL = async (tableName, region) => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    try {
        const ttlDescription = await client.send(new
 DescribeTimeToLiveCommand({ TableName: tableName }));

        if (ttlDescription.TimeToLiveDescription.TimeToLiveStatus === 'ENABLED')
 {
            console.log("TTL is enabled for table %s.", tableName);
        } else {
            console.log("TTL is not enabled for table %s.", tableName);
        }

        return ttlDescription;
    } catch (e) {
        console.error(`Error describing table: ${e}`);
        throw e;
    }
}

// Example usage (commented out for testing)
// describeTTL('your-table-name', 'us-east-1');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 DescribeTimeToLive를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용하여 기존 DynamoDB 테이블의 TTL 구성을 설명합니다.

```
import boto3


def describe_ttl(table_name, region):
    """
    Describes TTL on an existing table, as well as a region.

    :param table_name: String representing the name of the table
    :param region: AWS Region of the table - example `us-east-1`
    :return: Time to live description.
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        ttl_description = dynamodb.describe_time_to_live(TableName=table_name)
        print(
            f"TimeToLive for table {table_name} is status
 {ttl_description['TimeToLiveDescription']['TimeToLiveStatus']}"
        )

        return ttl_description
    except Exception as e:
        print(f"Error describing table: {e}")
        raise


# Enter your own table name and AWS region
describe_ttl("your-table-name", "us-east-1")
```

- API 세부 정보는 AWS SDK for Python(Boto3) API 참조의 DescribeTimeToLive를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK와 함께 **ExecuteStatement**사용

다음 코드 예시는 ExecuteStatement의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- PartiQL DELETE를 사용하여 데이터 삭제
- PartiQL INSERT를 사용하여 데이터 삽입
- PartiQL을 사용하여 테이블 쿼리
- PartiQL SELECT를 사용하여 데이터 쿼리
- PartiQL UPDATE를 사용하여 데이터 업데이트

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

INSERT 문을 사용하여 항목을 추가합니다.

```
        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
```

```
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

SELECT 문을 사용하여 항목을 가져옵니다.

```
        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

SELECT 문을 사용하여 항목 목록을 가져옵니다.

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

UPDATE 문을 사용하여 항목을 업데이트합니다.

```
        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
```

```
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

DELETE 문을 사용하여 하나의 영화를 삭제합니다.

```
        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
```

```
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ExecuteStatement를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

INSERT 문을 사용하여 항목을 추가합니다.

```
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
    Aws::String title;
    float rating;
    int year;
    Aws::String plot;
    {
        title = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        year = askQuestionForInt("What year was it released? ");
        rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                          1, 10);
        plot = askQuestion("Summarize the plot for me: ");

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
```

```
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        request.SetStatement(sqlStream.str());

        // Create the parameter attributes.
        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        ratingAttribute->SetN(rating);
        infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        plotAttribute->SetS(plot);
        infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
        attributes.push_back(infoMapAttribute);
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add a movie: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
```

SELECT 문을 사용하여 항목을 가져옵니다.

```
   //  3. Get the data for the movie using a "Select" statement.
(ExecuteStatement)
   {
       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       request.SetStatement(sqlStream.str());

       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
               request);

       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to retrieve movie information: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
       else {
           // Print the retrieved movie information.
           const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

           if (items.size() == 1) {
               printMovieInfo(items[0]);
           }
           else {
               std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                         << " There should be only one movie." << std::endl;
           }
       }
   }
```

UPDATE 문을 사용하여 항목을 업데이트합니다.

```
   //  4. Update the data for the movie using an "Update" statement.
(ExecuteStatement)
   {
       rating = askQuestionForFloatRange(
               Aws::String("\nLet's update your movie.\nYou rated it  ") +
               std::to_string(rating)
               + ", what new rating would you give it? ", 1, 10);

       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                 << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                 << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

       request.SetStatement(sqlStream.str());

       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
               request);

       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to update a movie: "
                     << outcome.GetError().GetMessage();
           return false;
       }
   }
```

DELETE 문을 사용하여 항목을 삭제합니다.

```
   // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
   {
       Aws::DynamoDB::Model::ExecuteStatementRequest request;
```

```
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
 dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movie: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 ExecuteStatement를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

예제를 위한 함수 수신기 구문을 정의합니다.

```
import (
 "context"
 "fmt"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

INSERT 문을 사용하여 항목을 추가합니다.

```
// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}
```

SELECT 문을 사용하여 항목을 가져옵니다.

```
// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

SELECT 문을 사용하여 항목 목록을 가져오고 결과를 투영합니다.

```
// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
```

```
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}
```

UPDATE 문을 사용하여 항목을 업데이트합니다.

```
// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
```

```
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}
```

DELETE 문을 사용하여 항목을 삭제합니다.

```
// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
```

```
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 ExecuteStatement를 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

PartiQL을 사용하여 항목을 생성합니다.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL을 사용하여 항목을 가져옵니다.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL을 사용하여 항목을 업데이트합니다.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });
```

```
  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL을 사용하여 항목을 삭제합니다.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 ExecuteStatement를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 ExecuteStatement를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statement, params):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statement: The PartiQL statement.
        :param params: The list of PartiQL parameters. These are applied to the
                       statement in the order they are listed.
        :return: The items returned from the statement, if any.
        """
        try:
            output = self.dyn_resource.meta.client.execute_statement(
                Statement=statement, Parameters=params
            )
        except ClientError as err:
```

```
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute PartiQL '%s' because the table does not
 exist.",
                    statement,
                )
            else:
                logger.error(
                    "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                    statement,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 ExecuteStatement를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

PartiQL을 사용하여 항목을 한 개 선택합니다.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end
```

```
  # Gets a single record from a table using PartiQL.
  # Note: To perform more fine-grained selects,
  # use the Client.query instance method instead.
  #
  # @param title [String] The title of the movie to search.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def select_item_by_title(title)
    request = {
      statement: "SELECT * FROM \"#{@table.name}\" WHERE title=?",
      parameters: [title]
    }
    @dynamodb.client.execute_statement(request)
  end
```

PartiQL을 사용하여 항목을 한 개 업데이트합니다.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Updates a single record from a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def update_rating_by_title(title, year, rating)
    request = {
      statement: "UPDATE \"#{@table.name}\" SET info.rating=? WHERE title=? and
 year=?",
      parameters: [{ "N": rating }, title, year]
    }
    @dynamodb.client.execute_statement(request)
  end
```

PartiQL을 사용하여 항목을 한 개 추가합니다.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Adds a single record to a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param plot [String] The plot of the movie.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def insert_item(title, year, plot, rating)
    request = {
      statement: "INSERT INTO \"#{@table.name}\" VALUE {'title': ?, 'year': ?,
 'info': ?}",
      parameters: [title, year, { 'plot': plot, 'rating': rating }]
    }
    @dynamodb.client.execute_statement(request)
  end
```

PartiQL을 사용하여 항목을 한 개 삭제합니다.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Deletes a single record from a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
```

```
# @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
def delete_item_by_title(title, year)
  request = {
    statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
    parameters: [title, year]
  }
  @dynamodb.client.execute_statement(request)
end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 ExecuteStatement를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **`GetItem`** 사용

다음 코드 예시는 `GetItem`의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 기본 사항 알아보기
- DAX로 읽기 가속화
- MRSC 글로벌 테이블 만들기 및 관리
- 글로벌 테이블 및 다중 리전 복제 최종 일관성(MREC) 작업

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
```

```
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
 {ex.Message}");
            throw;
```

```
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetItem을 참조하세요.

Bash

Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
```

```
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
  }
  query=""
  while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text \
```

```
      --query "$query")
  else
    response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
```

```
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 GetItem을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Get an item from an Amazon DynamoDB table.
/*!
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
                               const Aws::String &partitionKey,
                               const Aws::String &partitionValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;

    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
```

```
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cerr << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 GetItem을 참조하세요.

## CLI

### AWS CLI

예 1: 테이블의 항목을 읽는 방법

다음 `get-item` 예시에서는 `MusicCollection` 테이블에서 항목을 검색합니다. 테이블에는 해시 및 범위 프라이머리 키(`Artist` 및 `SongTitle`)가 있으므로 이 두 속성을 모두 지정해야 합니다. 또한 이 명령은 작업에 사용된 읽기 용량에 대한 정보를 요청합니다.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

`key.json`의 콘텐츠:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

출력:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 읽기를 참조하세요.

예 2: 일관된 읽기를 사용하여 항목을 읽는 방법

다음 예시에서는 강력히 일관되 읽기를 사용하여 `MusicCollection` 테이블의 항목을 읽습니다.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

`key.json`의 콘텐츠:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

출력:

```
{
    "Item": {
```

```
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 읽기를 참조하세요.

예 3: 항목의 특정 속성을 검색하는 방법

다음 예시에서는 프로젝션 표현식을 사용하여 원하는 항목의 세 가지 속성만 검색합니다.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

names.json의 콘텐츠:

```
{
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

출력:

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
```

```
        "Title": {
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 읽기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 GetItem을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
```

```
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 GetItem을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDbClient를 사용하여 테이블에서 항목을 가져옵니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
```

```
                    tableName - The Amazon DynamoDB table from which an item is
retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).\s
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                """";

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
ddb.getItem(request).item();
            if (returnedItem.isEmpty())
```

```
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 GetItem을 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 GetCommand를 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
```

```
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 GetItem을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블에서 항목을 가져옵니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
```

```
  } else {
    console.log("Success", data.Item);
  }
});
```

DynamoDB 문서 클라이언트를 사용하여 테이블에서 항목을 가져옵니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 GetItem을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun getSpecificItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 GetItem를 참조하세요.

## PHP

### SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";
```

```
    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 GetItem을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 파티션 키 SongTitle과 정렬 키 Artist가 있는 DynamoDB 항목을 반환합니다.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 GetItem을 참조하세요.

Tools for PowerShell V5

예 1: 파티션 키 SongTitle과 정렬 키 Artist가 있는 DynamoDB 항목을 반환합니다.

```
$key = @{
```

```
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 GetItem을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
```

```
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 GetItem를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Gets movie data from the table for a specific movie.
  #
  # @param title [String] The title of the movie.
  # @param year [Integer] The release year of the movie.
  # @return [Hash] The data about the requested movie.
  def get_item(title, year)
    @table.get_item(key: { 'year' => year, 'title' => title })
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 GetItem을 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        oo_item = lo_dyn->getitem(
          iv_tablename                = iv_table_name
          it_key                      = it_key ).
        DATA(lt_attr) = oo_item->get_item( ).
        DATA(lo_title) = lt_attr[ key = 'title' ]-value.
        DATA(lo_year) = lt_attr[ key = 'year' ]-value.
        DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( )
          && 'Movie year is: ' && lo_year->get_n( )
          && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 GetItem을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB
```

```
    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }
```

- API 세부 정보는 Swift용 AWS SDK API 참조의 GetItem을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **ListTables** 사용

다음 코드 예시는 ListTables의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 속성 기반 액세스 제어 설정

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
private static async Task ListMyTables()
{
    Console.WriteLine("\n*** Listing tables ***");

    string lastTableNameEvaluated = null;
    do
    {
        var response = await Client.ListTablesAsync(new ListTablesRequest
        {
            Limit = 2,
            ExclusiveStartTableName = lastTableNameEvaluated
        });

        foreach (var name in response.TableNames)
        {
            Console.WriteLine(name);
        }

        lastTableNameEvaluated = response.LastEvaluatedTableName;
    } while (lastTableNameEvaluated != null);
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListTables를 참조하세요.

Bash

Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_list_tables
#
# This function lists all the tables in a DynamoDB.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_list_tables() {
  response=$(aws dynamodb list-tables \
    --output text \
    --query "TableNames")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  echo "$response" | tr -s "[:space:]" "\n"

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi
```

```
  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 ListTables를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! List the Amazon DynamoDB tables for the current AWS account.
/*!
  \sa listTables()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::listTables(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
    listTablesRequest.SetLimit(50);
    do {
        const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamoClient.ListTables(
                listTablesRequest);
        if (!outcome.IsSuccess()) {
            std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }

        for (const auto &tableName: outcome.GetResult().GetTableNames())
            std::cout << tableName << std::endl;
        listTablesRequest.SetExclusiveStartTableName(
```

```
                outcome.GetResult().GetLastEvaluatedTableName());

    } while (!listTablesRequest.GetExclusiveStartTableName().empty());

    return true;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 ListTables를 참조하세요.

CLI

AWS CLI

예 1: 테이블을 나열하는 방법

다음 `list-tables` 예시에서는 AWS 계정 및 리전과 연결된 모든 테이블을 나열합니다.

```
aws dynamodb list-tables
```

출력:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 이름 나열을 참조하세요.

예 2: 페이지 크기를 제한하는 방법

다음 예시에서는 모든 기존 테이블의 목록을 반환하지만 각 호출에서 항목을 하나만 검색하고, 필요한 경우 전체 목록을 가져오기 위해 여러 번 호출합니다. 페이지 크기 제한은 많은 리소스에서 list 명령을 실행할 때 유용합니다. 리소스가 많을 때 기본 페이지 크기인 1,000을 사용하면 '시간 초과' 오류가 발생할 수 있습니다.

```
aws dynamodb list-tables \
```

```
    --page-size 1
```

출력:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 이름 나열을 참조하세요.

예 3: 반환되는 항목 수를 제한하는 방법

다음 예시에서는 반환되는 항목 수를 2개로 제한합니다. 응답에는 결과의 다음 페이지를 검색하는 데 사용되는 NextToken 값이 포함됩니다.

```
aws dynamodb list-tables \
    --max-items 2
```

출력:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog"
    ],
    "NextToken":
 "abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9"
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 이름 나열을 참조하세요.

예 4: 결과의 다음 페이지를 검색하는 방법

다음 명령은 이전의 list-tables 명령 호출에서 얻은 NextToken 값을 사용하여 다른 결과 페이지를 검색합니다. 이 경우 응답에는 NextToken 값이 포함되어 있지 않으므로 결과의 끝에 도달했음을 알 수 있습니다.

```
aws dynamodb list-tables \
    --starting-
token abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9
```

출력:

```
{
    "TableNames": [
        "Reply",
        "Thread"
    ]
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 테이블 이름 나열을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 ListTables를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 ListTables를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }
```

```
                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 ListTables를 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});
```

```
export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 ListTables를 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 ListTables를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun listAllTables() {
    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.listTables(ListTablesRequest {})
        response.tableNames?.forEach { tableName ->
            println("Table name is $tableName")
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 ListTables를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public function listTables($exclusiveStartTableName = "", $limit = 100)
    {
        $this->dynamoDbClient->listTables([
            'ExclusiveStartTableName' => $exclusiveStartTableName,
            'Limit' => $limit,
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 ListTables를 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 서비스에 더 이상 테이블이 없다고 표시될 때까지 자동으로 반복하여 모든 테이블의 세부 정보를 반환합니다.

```
Get-DDBTableList
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 ListTables를 참조하세요.

Tools for PowerShell V5

예 1: 서비스에 더 이상 테이블이 없다고 표시될 때까지 자동으로 반복하여 모든 테이블의 세부 정보를 반환합니다.

```
Get-DDBTableList
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 ListTables를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
```

```
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tables
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 ListTables를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블이 존재하는지 확인합니다.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Determines whether a table exists. As a side effect, stores the table in
  # a member variable.
  #
  # @param table_name [String] The name of the table to check.
  # @return [Boolean] True when the table exists; otherwise, False.
  def exists?(table_name)
    @dynamo_resource.client.describe_table(table_name: table_name)
    @logger.debug("Table #{table_name} exists")
  rescue Aws::DynamoDB::Errors::ResourceNotFoundException
    @logger.debug("Table #{table_name} doesn't exist")
    false
  rescue Aws::DynamoDB::Errors::ServiceError => e
```

```
    puts("Couldn't check for existence of #{table_name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 ListTables를 참조하세요.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
pub async fn list_tables(client: &Client) -> Result<Vec<String>, Error> {
    let paginator = client.list_tables().into_paginator().items().send();
    let table_names = paginator.collect::<Result<Vec<_>, _>>().await?;

    println!("Tables:");

    for name in &table_names {
        println!("  {}", name);
    }

    println!("Found {} tables", table_names.len());
    Ok(table_names)
}
```

테이블이 존재하는지 확인합니다.

```
pub async fn table_exists(client: &Client, table: &str) -> Result<bool, Error> {
    debug!("Checking for table: {table}");
    let table_list = client.list_tables().send().await;

    match table_list {
        Ok(list) => Ok(list.table_names().contains(&table.into())),
```

```
        Err(e) => Err(e.into()),
    }
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 ListTables을 참조하세요.

## SAP ABAP

### SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        oo_result = lo_dyn->listtables( ).
        " You can loop over the oo_result to get table properties like this.
        LOOP AT oo_result->get_tablenames( ) INTO DATA(lo_table_name).
          DATA(lv_tablename) = lo_table_name->get_value( ).
        ENDLOOP.
        DATA(lv_tablecount) = lines( oo_result->get_tablenames( ) ).
        MESSAGE 'Found ' && lv_tablecount && ' tables' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 ListTables을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    /// Get a list of the DynamoDB tables available in the specified Region.
    ///
    /// - Returns: An array of strings listing all of the tables available
    ///   in the Region specified when the session was created.
    public func getTableList() async throws -> [String] {
        let input = ListTablesInput(
        )
        return try await session.listTables(input: input)
    }
```

- API 세부 정보는 Swift용 AWS SDK API 참조의 ListTables를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **PutItem** 사용

다음 코드 예시는 `PutItem`의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 기본 사항 알아보기
- DAX로 읽기 가속화
- 고급 글로벌 보조 인덱스 시나리오
- TTL을 사용하여 항목 생성

- MREC를 보여주는 글로벌 테이블 만들기 및 관리
- MRSC 글로벌 테이블 만들기 및 관리
- 조건부 작업 사용
- 글로벌 테이블 및 다중 리전 복제 최종 일관성(MREC) 작업
- Streams 및 Time-to-Live 작업

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
```

```
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 PutItem을 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
#############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
```

```
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
```

```
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}
```

이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
```

```
  fi
}

################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
```

```
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 PutItem을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Put an item in an Amazon DynamoDB table.
/*!
  \sa putItem()
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
                               const Aws::String &artistValue,
                               const Aws::String &albumTitleKey,
                               const Aws::String &albumTitleValue,
                               const Aws::String &awardsKey,
                               const Aws::String &awardsValue,
                               const Aws::String &songTitleKey,
                               const Aws::String &songTitleValue,
```

```
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);

    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

테이블이 활성화될 때까지 대기하는 코드입니다.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
```

```
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 PutItem을 참조하세요.

## CLI

### AWS CLI

#### 예 1: 테이블에 항목을 추가하는 방법

다음 `put-item` 예시에서는 MusicCollection 테이블에 새 항목을 추가합니다.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

`item.json`의 콘텐츠:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

출력:

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 쓰기를 참조하세요.

예 2: 테이블의 항목을 조건부로 덮어쓰는 방법

다음 put-item 예시에서는 기존 항목에 값이 Greatest Hits인 AlbumTitle 속성이 있는 경우에만 MusicCollection 테이블의 기존 항목을 덮어씁니다. 이 명령은 항목의 이전 값을 반환합니다.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

item.json의 콘텐츠:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

names.json의 콘텐츠:

```
{
    "#A": "AlbumTitle"
}
```

values.json의 콘텐츠:

```
{
    ":A": {"S": "Greatest Hits"}
}
```

출력:

```
{
    "Attributes": {
```

```
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
    }
}
```

키가 이미 있는 경우 다음과 같은 출력이 표시됩니다.

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 쓰기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 PutItem을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
```

```
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 PutItem을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDbClient를 사용하여 테이블에 항목 추가

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
 <awards> <awardsval> <Songtitle> <songtitleval>

                Where:
```

```
                    tableName - The Amazon DynamoDB table in which an item is
placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example,
Happy Day).
                **Warning** This program will  place an item that you specify
into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }
```

```
    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {

        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle,
 AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
 AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The
 request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 PutItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 PutCommand를 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 PutItem을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블에 항목을 추가합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

DynamoDB 문서 클라이언트를 사용하여 테이블에 항목을 추가합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 PutItem을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
    albumTitle: String,
    albumTitleValue: String,
```

```
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 PutItem를 참조하세요.

## PHP

### SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
```

```
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 PutItem을 참조하세요.

## PowerShell

### Tools for PowerShell V4

예 1: 새 항목을 생성하거나 새 항목으로 기존 항목을 바꿉니다.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 PutItem을 참조하세요.

Tools for PowerShell V5

예 1: 새 항목을 생성하거나 새 항목으로 기존 항목을 바꿉니다.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 PutItem을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
```

```
                "Kevin Costner",
                "Kelly Preston",
                "John C. Reilly"
            ]
        }
    }
"""

def __init__(self, dyn_resource):
    """
    :param dyn_resource: A Boto3 DynamoDB resource.
    """
    self.dyn_resource = dyn_resource
    # The table variable is set during the scenario in the call to
    # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
    self.table = None


def add_movie(self, title, year, plot, rating):
    """
    Adds a movie to the table.

    :param title: The title of the movie.
    :param year: The release year of the movie.
    :param plot: The plot summary of the movie.
    :param rating: The quality rating of the movie.
    """
    try:
        self.table.put_item(
            Item={
                "year": year,
                "title": title,
                "info": {"plot": plot, "rating": Decimal(str(rating))},
            }
        )
    except ClientError as err:
        logger.error(
            "Couldn't add movie %s to table %s. Here's why: %s: %s",
            title,
            self.table.name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 PutItem를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
        'year' => movie[:year],
        'title' => movie[:title],
        'info' => { 'plot' => movie[:plot], 'rating' => movie[:rating] }
      }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 PutItem을 참조하세요.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
```

```
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 PutItem을 참조하세요.

## SAP ABAP

### SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 PutItem을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }
```

```
    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- API 세부 정보는 Swift용 AWS SDK API 참조의 PutItem을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **`Query`** 사용

다음 코드 예시는 `Query`의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 기본 사항 알아보기
- DAX로 읽기 가속화
- 고급 글로벌 보조 인덱스 시나리오
- 단일 속성과 여러 값 비교
- 글로벌 보조 인덱스 관리
- 고급 쿼리 작업 수행
- begins_with 조건을 사용하여 테이블 쿼리
- 날짜 범위를 사용하여 테이블 쿼리
- 글로벌 보조 인덱스를 사용하여 테이블 쿼리
- 복잡한 필터 표현식을 사용하여 테이블 쿼리
- 동적 필터 표현식을 사용하여 테이블 쿼리
- 필터 표현식 및 제한을 사용하여 테이블 쿼리
- 중첩 속성을 사용하여 테이블 쿼리
- 페이지 매김을 사용하여 테이블 쿼리
- 강력하게 일관된 읽기를 사용하여 테이블 쿼리
- TTL 항목에 대한 쿼리
- 날짜 및 시간 패턴을 사용하여 테이블 쿼리
- 표현식 속성 이름 사용
- 로컬 보조 인덱스로 작업

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
```

```
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 Query를 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
```

```
   echo " -a attribute_names -- Path to JSON file containing the attribute
names."
   echo " -v attribute_values -- Path to JSON file containing the attribute
values."
   echo " [-p projection_expression]  -- Optional projection expression."
   echo ""
 }

 while getopts "n:k:a:v:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) key_condition_expression="${OPTARG}" ;;
     a) attribute_names="${OPTARG}" ;;
     v) attribute_values="${OPTARG}" ;;
     p) projection_expression="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$key_condition_expression" ]]; then
   errecho "ERROR: You must provide a key condition expression with the -k
parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_names" ]]; then
   errecho "ERROR: You must provide a attribute names with the -a parameter."
   usage
```

```
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
##############################################################################
# function errecho
#
```

```
# This function outputs everything sent to it to STDERR (standard error output).
#########################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#########################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#########################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 Query를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
 * specified on the command line, only the specified fields and values are
 * returned.
 */

bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::QueryRequest request;

    request.SetTableName(tableName);
```

```
   if (!projectionExpression.empty()) {
       request.SetProjectionExpression(projectionExpression);
   }

   // Set query key condition expression.
   request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

   // Set Expression AttributeValues.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
   attributeValues.emplace(":valueToMatch", partitionValue);

   request.SetExpressionAttributeValues(attributeValues);

   bool result = true;

   // "exclusiveStartKey" is used for pagination.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
   do {
       if (!exclusiveStartKey.empty()) {
           request.SetExclusiveStartKey(exclusiveStartKey);
           exclusiveStartKey.clear();
       }
       // Perform Query operation.
       const Aws::DynamoDB::Model::QueryOutcome &outcome =
dynamoClient.Query(request);
       if (outcome.IsSuccess()) {
           // Reference the retrieved items.
           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
           if (!items.empty()) {
               std::cout << "Number of items retrieved from Query: " <<
items.size()
                         << std::endl;
               // Iterate each item and print.
               for (const auto &item: items) {
                   std::cout
                           <<
"*************************************************"
                           << std::endl;
                   // Output each retrieved field and its value.
                   for (const auto &i: item)
```

```
                        std::cout << i.first << ": " << i.second.GetS() <<
 std::endl;
                }
            }
            else {
                std::cout << "No item found in table: " << tableName <<
 std::endl;
            }

            exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Query items: " <<
 outcome.GetError().GetMessage();
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 Query를 참조하세요.

CLI

AWS CLI

예 1: 테이블을 쿼리하는 방법

다음 query 예시에서는 MusicCollection 테이블의 항목을 쿼리합니다. 테이블에는 해시 및 범위 프라이머리 키(Artist 및 SongTitle)가 있지만 이 쿼리는 해시 키 값만 지정합니다. 'No One You Know'라는 아티스트의 노래 제목이 반환됩니다.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

expression-attributes.json의 콘텐츠:

```
{
    ":v1": {"S": "No One You Know"}
}
```

출력:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 쿼리 작업을 참조하세요.

예 2: 강력히 일관된 읽기를 사용하여 테이블을 쿼리하고 인덱스를 내림차순으로 탐색하는 방법

다음 예시에서는 첫 번째 예와 동일한 쿼리를 수행하지만 결과를 역순으로 반환하고 강력히 일관된 읽기를 사용합니다.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

expression-attributes.json의 콘텐츠:

```
{
    ":v1": {"S": "No One You Know"}
}
```

출력:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 쿼리 작업을 참조하세요.

예 3: 특정 결과를 필터링하는 방법

다음 예시에서는 MusicCollection을 쿼리하되 AlbumTitle 속성에 특정 값이 있는 결과를 제외합니다. 항목을 읽은 후에 필터가 적용되므로 ScannedCount 또는 ConsumedCapacity에는 영향을 주지 않는다는 점에 유의하세요.

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
```

```
    --return-consumed-capacity TOTAL
```

values.json의 콘텐츠:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

names.json의 콘텐츠:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

출력:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 쿼리 작업을 참조하세요.

예 4: 항목 수만 검색하는 방법

다음 예시에서는 쿼리와 일치하는 항목 수를 검색하지만 항목 자체는 검색하지 않습니다.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

expression-attributes.json의 콘텐츠:

```
{
    ":v1": {"S": "No One You Know"}
}
```

출력:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 쿼리 작업을 참조하세요.

예 5: 인덱스를 쿼리하는 방법

다음 예시에서는 로컬 보조 인덱스 AlbumTitleIndex를 쿼리합니다. 쿼리는 로컬 보조 인덱스로 프로젝션된 기본 테이블의 모든 속성을 반환합니다. 로컬 보조 인덱스 또는 글로벌 보조 인덱스를 쿼리할 때는 table-name 파라미터를 사용하여 기본 테이블의 이름도 제공해야 한다는 점에 유의하세요.

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

expression-attributes.json의 콘텐츠:

```
{
    ":v1": {"S": "No One You Know"}
}
```

출력:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
                "CapacityUnits": 0.5
            }
```

```
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 쿼리 작업을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 Query를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 Query를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDbClient를 사용하여 테이블을 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
```

```
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];

        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }
```

```
   public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
           String partitionAlias) {
       // Set up an alias for the partition key name in case it's a reserved
 word.
       HashMap<String, String> attrNameAlias = new HashMap<String, String>();
       attrNameAlias.put(partitionAlias, partitionKeyName);

       // Set up mapping of the partition name with the value.
       HashMap<String, AttributeValue> attrValues = new HashMap<>();
       attrValues.put(":" + partitionKeyName, AttributeValue.builder()
               .s(partitionKeyVal)
               .build());

       QueryRequest queryReq = QueryRequest.builder()
               .tableName(tableName)
               .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
               .expressionAttributeNames(attrNameAlias)
               .expressionAttributeValues(attrValues)
               .build();

       try {
           QueryResponse response = ddb.query(queryReq);
           return response.count();

       } catch (DynamoDbException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
       return -1;
   }
}
```

DynamoDbClient 및 보조 인덱스를 사용하여 테이블을 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
```

```
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
            QueryResponse response = ddb.query(request);
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 QueryCommand를 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
```

```
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
```

```
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
```

```
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 Query를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
```

```
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->query($query);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 Query를 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 지정된 SongTitle 및 Artist와 함께 DynamoDB 항목을 반환하는 쿼리를 간접 호출합니다.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

출력:

```
Name                           Value
----                           -----
```

```
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 Query를 참조하세요.

Tools for PowerShell V5

예 1: 지정된 SongTitle 및 Artist와 함께 DynamoDB 항목을 반환하는 쿼리를 간접 호출합니다.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 Query를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

키 조건 표현식을 사용하여 항목을 쿼리합니다.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
 self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]
```

데이터 하위 집합을 반환하도록 항목을 쿼리하고 프로젝션합니다.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
 titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
```

```
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Items"]
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 Query를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
      key_condition_expression: '#yr = :year',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: { ':year' => year }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.items
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 Query를 참조하세요.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

지정된 연도에 제작된 영화를 찾습니다.

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;

    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 Query를 참조하세요.

## SAP ABAP

### SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
```

```
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 Query을 참조하세요.

## Swift

### SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
```

```
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
```

```
            throw error
        }
    }
```

- API에 대한 세부 정보는 AWS Swift용 SDK API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **Scan** 사용

다음 코드 예시는 Scan의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 기본 사항 알아보기
- DAX로 읽기 가속화
- 단일 속성과 여러 값 비교

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
```

```
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
```

```
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
 table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
 {ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 Scan을 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
```

```
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
    echo " -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:f:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      f) filter_expression="${OPTARG}" ;;
      a) expression_attribute_names="${OPTARG}" ;;
      v) expression_attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
```

```
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$filter_expression" ]]; then
    errecho "ERROR: You must provide a filter expression with the -f parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_names" ]]; then
    errecho "ERROR: You must provide expression attribute names with the -a
parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_values" ]]; then
    errecho "ERROR: You must provide expression attribute values with the -v
parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values")
  else
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
```

```
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
```

```
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API 세부 정보는 AWS CLI 명령 참조의 Scan을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Scan an Amazon DynamoDB table.
/*!
  \sa scanTable()
  \param tableName: Name for the DynamoDB table.
  \param projectionExpression: An optional projection expression, ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
```

```
 */

bool AwsDoc::DynamoDB::scanTable(const Aws::String &tableName,
                                 const Aws::String &projectionExpression,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::ScanRequest request;
    request.SetTableName(tableName);

    if (!projectionExpression.empty())
        request.SetProjectionExpression(projectionExpression);

    Aws::Vector<Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>>
 all_items;
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 last_evaluated_key; // Used for pagination;
    do {
        if (!last_evaluated_key.empty()) {
            request.SetExclusiveStartKey(last_evaluated_key);
        }
        const Aws::DynamoDB::Model::ScanOutcome &outcome =
 dynamoClient.Scan(request);
        if (outcome.IsSuccess()) {
            // Reference the retrieved items.
            const Aws::Vector<Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
            all_items.insert(all_items.end(), items.begin(), items.end());

            last_evaluated_key = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Scan items: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }

    } while (!last_evaluated_key.empty());

    if (!all_items.empty()) {
        std::cout << "Number of items retrieved from scan: " << all_items.size()
                  << std::endl;
        // Iterate each item and print.
```

```
        for (const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 &itemMap: all_items) {
            std::cout << "*****************************************************"
                      << std::endl;
            // Output each retrieved field and its value.
            for (const auto &itemEntry: itemMap)
                std::cout << itemEntry.first << ": " << itemEntry.second.GetS()
                          << std::endl;
        }
    }

    else {
        std::cout << "No items found in table: " << tableName << std::endl;
    }

    return true;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 Scan을 참조하세요.

CLI

AWS CLI

테이블을 스캔하는 방법

다음 scan 예시에서는 MusicCollection 테이블 전체를 스캔한 다음 'No One You Know' 아티스트의 곡으로 결과 범위를 좁힙니다. 각 항목에 대해 앨범 제목과 노래 제목만 반환됩니다.

```
aws dynamodb scan \
    --table-name MusicCollection \
    --filter-expression "Artist = :a" \
    --projection-expression "#ST, #AT" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json
```

expression-attribute-names.json의 콘텐츠:

```
{
    "#ST": "SongTitle",
```

```
    "#AT":"AlbumTitle"
}
```

expression-attribute-values.json의 콘텐츠:

```
{
    ":a": {"S": "No One You Know"}
}
```

출력:

```
{
    "Count": 2,
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "AlbumTitle": {
                "S": "Somewhat Famous"
            }
        },
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            },
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            }
        }
    ],
    "ScannedCount": 3,
    "ConsumedCapacity": null
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 스캔 작업을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 Scan을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
```

```
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 Scan을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDbClient를 사용하여 Amazon DynamoDB 테이블을 스캔합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To scan items from an Amazon DynamoDB table using the AWS SDK for Java V2,
```

```
 * its better practice to use the
 * Enhanced Client, See the EnhancedScanRecords example.
 */

public class DynamoDBScanItems {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to get information from
 (for example, Music3).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        scanItems(ddb, tableName);
        ddb.close();
    }

    public static void scanItems(DynamoDbClient ddb, String tableName) {
        try {
            ScanRequest scanRequest = ScanRequest.builder()
                    .tableName(tableName)
                    .build();

            ScanResponse response = ddb.scan(scanRequest);
            for (Map<String, AttributeValue> item : response.items()) {
                Set<String> keys = item.keySet();
                for (String key : keys) {
                    System.out.println("The key name is " + key + "\n");
                    System.out.println("The value is " + item.get(key).s());
```

```
                }
            }

        } catch (DynamoDbException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Scan을 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 ScanCommand를 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
```

```
  }
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Scan을 참조하세요.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values
 you want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
```

```
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 Scan을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun scanItems(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 Scan를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

    public function scan(string $tableName, array $key, string $filters)
    {
        $query = [
            'ExpressionAttributeNames' => ['#year' => 'year'],
            'ExpressionAttributeValues' => [
                ":min" => ['N' => '1990'],
                ":max" => ['N' => '1999'],
            ],
            'FilterExpression' => "#year between :min and :max",
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->scan($query);
```

```
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 Scan을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: Music 테이블에서 모든 항목을 반환합니다.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
Genre                          Country
Artist                         No One You Know
Price                          1.98
CriticRating                   8.4
SongTitle                      My Dog Spot
AlbumTitle                     Hey Now
```

예 2: Music 테이블에서 CriticRating이 9 이상인 항목을 반환합니다.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

출력:

```
Name                            Value
----                            -----
Genre                           Country
Artist                          No One You Know
Price                           1.94
CriticRating                    9
SongTitle                       Somewhere Down The Road
AlbumTitle                      Somewhat Famous
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 Scan을 참조하세요.

Tools for PowerShell V5

예 1: Music 테이블에서 모든 항목을 반환합니다.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

출력:

```
Name                            Value
----                            -----
Genre                           Country
Artist                          No One You Know
Price                           1.94
CriticRating                    9
SongTitle                       Somewhere Down The Road
AlbumTitle                      Somewhat Famous
Genre                           Country
Artist                          No One You Know
Price                           1.98
CriticRating                    8.4
SongTitle                       My Dog Spot
AlbumTitle                      Hey Now
```

예 2: Music 테이블에서 CriticRating이 9 이상인 항목을 반환합니다.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
```

```
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 Scan을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
```

```
                "Kevin Costner",
                "Kelly Preston",
                "John C. Reilly"
            ]
        }
    }
"""

def __init__(self, dyn_resource):
    """
    :param dyn_resource: A Boto3 DynamoDB resource.
    """
    self.dyn_resource = dyn_resource
    # The table variable is set during the scenario in the call to
    # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
    self.table = None


def scan_movies(self, year_range):
    """
    Scans for movies that were released in a range of years.
    Uses a projection expression to return a subset of data for each movie.

    :param year_range: The range of years to retrieve.
    :return: The list of movies released in the specified years.
    """
    movies = []
    scan_kwargs = {
        "FilterExpression": Key("year").between(
            year_range["first"], year_range["second"]
        ),
        "ProjectionExpression": "#yr, title, info.rating",
        "ExpressionAttributeNames": {"#yr": "year"},
    }
    try:
        done = False
        start_key = None
        while not done:
            if start_key:
                scan_kwargs["ExclusiveStartKey"] = start_key
            response = self.table.scan(**scan_kwargs)
            movies.extend(response.get("Items", []))
            start_key = response.get("LastEvaluatedKey", None)
            done = start_key is None
```

```
        except ClientError as err:
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 Scan를 참조하세요.

## Ruby

### SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Scans for movies that were released in a range of years.
  # Uses a projection expression to return a subset of data for each movie.
  #
  # @param year_range [Hash] The range of years to retrieve.
  # @return [Array] The list of movies released in the specified years.
  def scan_items(year_range)
    movies = []
    scan_hash = {
      filter_expression: '#yr between :start_yr and :end_yr',
```

```
      projection_expression: '#yr, title, info.rating',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: {
        ':start_yr' => year_range[:start], ':end_yr' => year_range[:end]
      }
    }
    done = false
    start_key = nil
    until done
      scan_hash[:exclusive_start_key] = start_key unless start_key.nil?
      response = @table.scan(scan_hash)
      movies.concat(response.items) unless response.items.empty?
      start_key = response.last_evaluated_key
      done = start_key.nil?
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't scan for movies. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    movies
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 Scan을 참조하세요.

## Rust

### SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
pub async fn list_items(client: &Client, table: &str, page_size: Option<i32>) ->
 Result<(), Error> {
    let page_size = page_size.unwrap_or(10);
    let items: Result<Vec<_>, _> = client
        .scan()
        .table_name(table)
```

```
        .limit(page_size)
        .into_paginator()
        .items()
        .send()
        .collect()
        .await;

    println!("Items in table (up to {page_size}):");
    for item in items? {
        println!("   {:?}", item);
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 Scan을 참조하세요.

## SAP ABAP

### SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        " Scan movies for rating greater than or equal to the rating specified
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_rating }| ) ) ).
        DATA(lt_filter_conditions) = VALUE /aws1/
cl_dyncondition=>tt_filterconditionmap(
          ( VALUE /aws1/cl_dyncondition=>ts_filterconditionmap_maprow(
          key = 'rating'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |GE|
          ) ) ) ).
        oo_scan_result = lo_dyn->scan( iv_tablename = iv_table_name
```

```
          it_scanfilter = lt_filter_conditions ).
        DATA(lt_items) = oo_scan_result->get_items( ).
        LOOP AT lt_items INTO DATA(lo_item).
          " You can loop over to get individual attributes.
          DATA(lo_title) = lo_item[ key = 'title' ]-value.
          DATA(lo_year) = lo_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_scan_result->get_count( ).
        MESSAGE 'Found ' && lv_count && ' items' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 Scan을 참조하세요.

## Swift

### SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB


    /// Return an array of `Movie` objects released in the specified range of
    /// years.
    ///
    /// - Parameters:
    ///   - firstYear: The first year of movies to return.
    ///   - lastYear: The last year of movies to return.
    ///   - startKey: A starting point to resume processing; always use `nil`.
    ///
    /// - Returns: An array of `Movie` objects describing the matching movies.
    ///
    /// > Note: The `startKey` parameter is used by this function when
    ///   recursively calling itself, and should always be `nil` when calling
    ///   directly.
```

```
    ///
    func getMovies(firstYear: Int, lastYear: Int,
                   startKey: [Swift.String: DynamoDBClientTypes.AttributeValue]?
= nil)
        async throws -> [Movie]
    {
        do {
            var movieList: [Movie] = []

            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = ScanInput(
                consistentRead: true,
                exclusiveStartKey: startKey,
                expressionAttributeNames: [
                    "#y": "year" // `year` is a reserved word, so use `#y`
instead.
                ],
                expressionAttributeValues: [
                    ":y1": .n(String(firstYear)),
                    ":y2": .n(String(lastYear))
                ],
                filterExpression: "#y BETWEEN :y1 AND :y2",
                tableName: self.tableName
            )

            let pages = client.scanPaginated(input: input)

            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Build an array of `Movie` objects for the returned items.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
```

```
        } catch {
            print("ERROR: getMovies with scan:", dump(error))
            throw error
        }
    }
```

- API에 대한 세부 정보는 AWS Swift용 SDK API 참조의 Scan을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **`UpdateItem`** 사용

다음 코드 예시는 `UpdateItem`의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 기본 사항 알아보기
- 항목의 TTL을 조건부로 업데이트
- 표현식 연산자 수 계산
- MRSC 글로벌 테이블 만들기 및 관리
- 목록 작업 수행
- 맵 작업 수행
- 집합 작업 수행
- 업데이트 표현식 순서 이해
- 항목의 TTL 업데이트
- 원자성 카운터 작업 사용
- 조건부 작업 사용
- 표현식 속성 이름 사용

.NET

SDK for .NET(v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },
```

```
                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 UpdateItem을 참조하세요.

## Bash

### Bash 스크립트와 함께 AWS CLI사용

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
```

```
    echo " -e update expression  -- An expression that defines one or more
attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
 }

 while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
 fi

 if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
 fi
 if [[ -z "$update_expression" ]]; then
    errecho "ERROR: You must provide an update expression with the -e parameter."
    usage
    return 1
 fi

 if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
```

```
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}
```

이 예제에 사용된 유틸리티 함수

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}
```

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
```

```
}
```

- API 세부 정보는 AWS CLI 명령 참조의 UpdateItem을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Update an Amazon DynamoDB table item.
/*!
  \sa updateItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &attributeKey,
                                  const Aws::String &attributeValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
```

```
    // *** Define UpdateItem request arguments.
    // Define TableName argument.
    Aws::DynamoDB::Model::UpdateItemRequest request;
    request.SetTableName(tableName);

    // Define KeyName argument.
    Aws::DynamoDB::Model::AttributeValue attribValue;
    attribValue.SetS(partitionValue);
    request.AddKey(partitionKey, attribValue);

    // Construct the SET update expression argument.
    Aws::String update_expression("SET #a = :valueA");
    request.SetUpdateExpression(update_expression);

    // Construct attribute name argument.
    Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
    expressionAttributeNames["#a"] = attributeKey;
    request.SetExpressionAttributeNames(expressionAttributeNames);

    // Construct attribute value argument.
    Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
    attributeUpdatedValue.SetS(attributeValue);
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 expressionAttributeValues;
    expressionAttributeValues[":valueA"] = attributeUpdatedValue;
    request.SetExpressionAttributeValues(expressionAttributeValues);

    // Update the item.
    const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
 dynamoClient.UpdateItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item was updated" << std::endl;
    } else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

테이블이 활성화될 때까지 대기하는 코드입니다.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 UpdateItem을 참조하세요.

CLI

AWS CLI

예 1: 테이블의 항목을 업데이트하는 방법

다음 update-item 예제에서는 MusicCollection 테이블의 항목을 업데이트합니다. 새 속성(Year)을 추가하고 AlbumTitle 속성을 수정합니다. 업데이트 후에 표시되는 항목 속성이 모두 응답에 반환됩니다.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

key.json의 콘텐츠:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

expression-attribute-names.json의 콘텐츠:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

expression-attribute-values.json의 콘텐츠:

```
{
    ":y":{"N": "2015"},
```

```
    ":t":{"S": "Louder Than Ever"}
}
```

출력:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
            "N": "2015"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 쓰기를 참조하세요.

예 2: 항목을 조건부로 업데이트하는 방법

다음 예시에서는 기존 항목에 Year 속성이 없는 경우에만 MusicCollection 테이블의 항목을 업데이트합니다.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

key.json의 콘텐츠:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

expression-attribute-names.json의 콘텐츠:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

expression-attribute-values.json의 콘텐츠:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

항목에 이미 Year 속성이 있는 경우 DynamoDB는 다음 출력을 반환합니다.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 항목 쓰기를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 UpdateItem을 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
```

```
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 UpdateItem을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

DynamoDbClient를 사용하여 테이블의 항목 업데이트

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
```

```
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
example, Awards).
                    updateVal - The value used to update an item (for example,
14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String name,
            String updateVal) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
```

```
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예제에서는 문서 클라이언트를 사용하여 DynamoDB의 항목 작업을 단순화합니다. API에 대한 세부 정보는 UpdateCommand를 참조하세요.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```

```
export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
```

```
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 UpdateItem를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);
```

```
    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
                ]
            ],
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 UpdateItem을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 파티션 키 SongTitle과 정렬 키 Artist가 있는 DynamoDB 항목에서 장르 속성을 'Rap'으로 설정합니다.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
```

```
    })
}
Update-DDBItem @updateDdbItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Rap
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 UpdateItem을 참조하세요.

Tools for PowerShell V5

예 1: 파티션 키 SongTitle과 정렬 키 Artist가 있는 DynamoDB 항목에서 장르 속성을 'Rap'으로 설정합니다.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

출력:

```
Name                           Value
----                           -----
Genre                          Rap
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 UpdateItem을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

업데이트 표현식을 사용하여 항목을 업데이트합니다.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
 plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

산술 연산을 포함하는 업데이트 표현식을 사용하여 항목을 업데이트합니다.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
```

```
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
movie.
        :return: The updated rating.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

특정 조건을 충족하는 경우에만 항목을 업데이트합니다.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
```

```
        Removes an actor from a movie, but only when the number of actors is
greater
        than a specified threshold. If the movie does not list more than the
threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"ConditionalCheckFailedException":
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["Attributes"]
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Updates rating and plot data for a movie in the table.
  #
  # @param movie [Hash] The title, year, plot, rating of the movie.
  def update_item(movie)
    response = @table.update_item(
      key: { 'year' => movie[:year], 'title' => movie[:title] },
      update_expression: 'set info.rating=:r',
      expression_attribute_values: { ':r' => movie[:rating] },
      return_values: 'UPDATED_NEW'
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
 #{@table.name}\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.attributes
  end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 UpdateItem을 참조하세요.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        oo_output = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = it_item_key
          it_attributeupdates = it_attribute_updates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWSSDK for SAP ABAP API의 UpdateItem을 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSDynamoDB
```

```
    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
[:]

            if rating != nil {
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
```

```
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }
```

- API에 대한 세부 정보는 AWS Swift용 SDK API 참조의 UpdateItem을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **UpdateTable** 사용

다음 코드 예시는 UpdateTable의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- MREC를 보여주는 글로벌 테이블 만들기 및 관리
- MRSC 글로벌 테이블 만들기 및 관리
- 글로벌 보조 인덱스 관리

- 테이블의 원 처리량 설정 업데이트
- 글로벌 테이블 및 다중 리전 복제 최종 일관성(MREC) 작업
- 테이블 암호화 작업

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Update a DynamoDB table.
/*!
  \sa updateTable()
  \param tableName: Name for the DynamoDB table.
  \param readCapacity: Provisioned read capacity.
  \param writeCapacity: Provisioned write capacity.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::updateTable(const Aws::String &tableName,
                                   long long readCapacity, long long
 writeCapacity,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Updating " << tableName << " with new provisioned throughput
 values"
              << std::endl;
    std::cout << "Read capacity : " << readCapacity << std::endl;
    std::cout << "Write capacity: " << writeCapacity << std::endl;

    Aws::DynamoDB::Model::UpdateTableRequest request;
    Aws::DynamoDB::Model::ProvisionedThroughput provisionedThroughput;

 provisionedThroughput.WithReadCapacityUnits(readCapacity).WithWriteCapacityUnits(
           writeCapacity);
```

```
 request.WithProvisionedThroughput(provisionedThroughput).WithTableName(tableName);

    const Aws::DynamoDB::Model::UpdateTableOutcome &outcome =
 dynamoClient.UpdateTable(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated the table." << std::endl;
    } else {
        const Aws::DynamoDB::DynamoDBError &error = outcome.GetError();
        if (error.GetErrorType() == Aws::DynamoDB::DynamoDBErrors::VALIDATION &&
            error.GetMessage().find("The provisioned throughput for the table
 will not change") != std::string::npos) {
            std::cout << "The provisioned throughput for the table will not
 change." << std::endl;
        } else {
            std::cerr << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    return waitTableActive(tableName, dynamoClient);
}
```

테이블이 활성화될 때까지 대기하는 코드입니다.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);
```

```
    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 UpdateTable을 참조하세요.

## CLI

### AWS CLI

#### 예 1: 테이블의 결제 모드를 수정하는 방법

다음 `update-table` 예시에서는 `MusicCollection` 테이블에 프로비저닝된 읽기 및 쓰기 용량을 늘립니다.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --billing-mode PROVISIONED \
    --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10
```

출력:


```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T13:18:18.921000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
```

```
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        }
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 Updating a Table을 참조하세요.

예 2: 글로벌 보조 인덱스를 생성하는 방법

다음 예시에서는 `MusicCollection` 테이블에 글로벌 보조 인덱스를 추가합니다.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
    --global-secondary-index-updates file://gsi-updates.json
```

`gsi-updates.json`의 콘텐츠:

```
[
    {
        "Create": {
            "IndexName": "AlbumTitle-index",
            "KeySchema": [
                {
                    "AttributeName": "AlbumTitle",
                    "KeyType": "HASH"
                }
            ],
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 10,
                "WriteCapacityUnits": 10
            },
            "Projection": {
                "ProjectionType": "ALL"
            }
        }
    }
]
```

출력:

```
{
```

```
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
```

```
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "Backfilling": false,
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ]
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 Updating a Table을 참조하세요.

예 3: 테이블에서 DynamoDB Streams를 활성화하는 방법

다음 명령은 `MusicCollection` 테이블에서 DynamoDB Streams를 활성화합니다.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_IMAGE
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
```

```
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
```

```
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
```

```
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112"
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 Updating a Table을 참조하세요.

예 4: 서버 측 암호화를 활성화하는 방법

다음 예시에서는 MusicCollection 테이블에서 서버 측 암호화를 활성화합니다.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --sse-specification Enabled=true,SSEType=KMS
```

출력:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
```

```
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
```

```
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112",
        "SSEDescription": {
            "Status": "UPDATING"
        }
```

```
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 Updating a Table을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 UpdateTable을 참조하세요.

PowerShell

Tools for PowerShell V4

예 1: 주어진 테이블의 프로비저닝된 처리량을 업데이트합니다.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 UpdateTable을 참조하세요.

Tools for PowerShell V5

예 1: 주어진 테이블의 프로비저닝된 처리량을 업데이트합니다.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 UpdateTable을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK 또는 CLI와 함께 **UpdateTimeToLive** 사용

다음 코드 예시는 UpdateTimeToLive의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- Streams 및 Time-to-Live 작업

CLI

AWS CLI

테이블에서 Time to Live 설정을 업데이트하려면

다음 update-time-to-live 예제에서는 지정된 테이블에서 Time to Live를 활성화합니다.

```
aws dynamodb update-time-to-live \
    --table-name MusicCollection \
    --time-to-live-specification Enabled=true,AttributeName=ttl
```

출력:

```
{
    "TimeToLiveSpecification": {
        "Enabled": true,
        "AttributeName": "ttl"
    }
}
```

자세한 내용은 Amazon DynamoDB 개발자 안내서의 Time to Live를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 UpdateTimeToLive를 참조하세요.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용하여 기존 DynamoDB 테이블에서 TTL을 활성화합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.TimeToLiveSpecification;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveResponse;

import java.util.logging.Level;
import java.util.logging.Logger;

    public UpdateTimeToLiveResponse enableTTL(final String tableName, final
 String attributeName, final Region region) {
        final TimeToLiveSpecification ttlSpec = TimeToLiveSpecification.builder()
            .attributeName(attributeName)
            .enabled(true)
            .build();
```

```
        final UpdateTimeToLiveRequest request = UpdateTimeToLiveRequest.builder()
            .tableName(tableName)
            .timeToLiveSpecification(ttlSpec)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.updateTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 기존 DynamoDB 테이블에서 TTL을 비활성화합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.TimeToLiveSpecification;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveResponse;

import java.util.logging.Level;
import java.util.logging.Logger;

    public UpdateTimeToLiveResponse disableTTL(
        final String tableName, final String attributeName, final Region region)
 {
        final TimeToLiveSpecification ttlSpec = TimeToLiveSpecification.builder()
            .attributeName(attributeName)
            .enabled(false)
            .build();

        final UpdateTimeToLiveRequest request = UpdateTimeToLiveRequest.builder()
            .tableName(tableName)
```

```
            .timeToLiveSpecification(ttlSpec)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.updateTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateTimeToLive를 참조하세요.

JavaScript

SDK for JavaScript (v3)

기존 DynamoDB 테이블에서 TTL을 활성화합니다.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const enableTTL = async (tableName, ttlAttribute, region = 'us-east-1') =>
 {

    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };
```

```
    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// Example usage (commented out for testing)
// enableTTL('ExampleTable', 'exampleTtlAttribute');
```

기존 DynamoDB 테이블에서 TTL을 비활성화합니다.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const disableTTL = async (tableName, ttlAttribute, region = 'us-east-1')
 => {

    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: false,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
```

```
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL disabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to disable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error disabling TTL: ${e}`);
        throw e;
    }
};

// Example usage (commented out for testing)
// disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateTimeToLive를 참조하세요.

Python

SDK for Python(Boto3)

기존 DynamoDB 테이블에서 TTL을 활성화합니다.

```
import boto3


def enable_ttl(table_name, ttl_attribute_name):
    """
    Enables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table
    :param ttl_attribute_name: The name of the TTL attribute being provided to
 the table.
    """
    try:
        dynamodb = boto3.client("dynamodb")

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
```

```
            TableName=table_name,
            TimeToLiveSpecification={"Enabled": True, "AttributeName":
 ttl_attribute_name},
        )

        # In the returned response, check for a successful status code.
        if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
            print("TTL has been enabled successfully.")
        else:
            print(
                f"Failed to enable TTL, status code {response['ResponseMetadata']
['HTTPStatusCode']}"
            )
        return response
    except Exception as ex:
        print("Couldn't enable TTL in table %s. Here's why: %s" % (table_name,
 ex))
        raise


# your values
enable_ttl("your-table-name", "expireAt")
```

기존 DynamoDB 테이블에서 TTL을 비활성화합니다.

```
import boto3


def disable_ttl(table_name, ttl_attribute_name):
    """
    Disables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table being modified
    :param ttl_attribute_name: The name of the TTL attribute being provided to
 the table.
    """
    try:
        dynamodb = boto3.client("dynamodb")

        # Enable TTL on an existing DynamoDB table
```

```
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={"Enabled": False, "AttributeName":
 ttl_attribute_name},
        )

        # In the returned response, check for a successful status code.
        if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
            print("TTL has been disabled successfully.")
        else:
            print(
                f"Failed to disable TTL, status code
 {response['ResponseMetadata']['HTTPStatusCode']}"
            )
    except Exception as ex:
        print("Couldn't disable TTL in table %s. Here's why: %s" % (table_name,
 ex))
        raise


# your values
disable_ttl("your-table-name", "expireAt")
```

- API 세부 정보는 AWS SDK for Python(Boto3) API 참조의 UpdateTimeToLive를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

# AWS SDK를 사용한 DynamoDB 관련 시나리오

다음 코드 예제에서는 AWS SDK로 DynamoDB에서 일반적인 시나리오를 구현하는 방법을 보여줍니다. 이러한 시나리오에서는 DynamoDB 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여줍니다. 각 시나리오에는 전체 소스 코드에 대한 링크가 포함되어 있습니다. 여기에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

시나리오는 컨텍스트에 맞는 서비스 작업을 이해하는 데 도움이 되도록 중급 수준의 경험을 대상으로 합니다.

예시

- AWS SDK를 사용하여 DAX로 DynamoDB 읽기 가속화
- AWS Command Line Interface v2를 사용한 고급 DynamoDB 글로벌 보조 인덱스 시나리오 작업
- DynamoDB 테이블에 데이터를 제출하기 위한 애플리케이션 구축
- AWS SDK를 사용하여 DynamoDB에서 단일 속성과 여러 값 비교
- AWS SDK를 사용하여 TTL로 DynamoDB 항목을 조건부로 업데이트
- AWS SDK를 사용하여 로컬 DynamoDB 인스턴스에 연결
- AWS SDK를 사용하여 DynamoDB에서 표현식 연산자 수 계산
- COVID-19 데이터를 추적하는 API Gateway REST API 생성
- Step Functions를 사용하여 메신저 애플리케이션 생성
- 사용자가 레이블을 사용하여 사진을 관리할 수 있는 사진 자산 관리 애플리케이션 만들기
- AWS SDK를 사용하여 글로벌 보조 인덱스가 포함된 DynamoDB 테이블 만들기
- AWS SDK를 사용하여 웜 처리량 설정으로 DynamoDB 테이블 만들기
- DynamoDB 데이터를 추적하는 웹 애플리케이션 만들기
- API Gateway를 사용하여 WebSocket 채팅 애플리케이션 생성
- AWS SDK를 사용하여 TTL이 포함된 DynamoDB 항목 생성
- AWS SDK를 사용하여 다중 리전 강력한 일관성을 갖춘 DynamoDB 글로벌 테이블을 만들고 관리하는 방법
- AWS SDK를 사용하여 MREC를 보여주는 DynamoDB 글로벌 테이블 만들기 및 관리
- AWS SDK와 함께 PartiQL DELETE 문을 사용하여 DynamoDB 데이터 삭제
- AWS SDK를 사용하여 Amazon Rekognition을 통해 이미지에서 PPE 감지
- AWS SDK와 함께 PartiQL INSERT 문을 사용하여 DynamoDB 데이터 삽입
- 브라우저에서 Lambda 함수 간접 호출
- AWS Command Line Interface v2를 사용하여 DynamoDB 글로벌 보조 인덱스 관리
- AWS Command Line Interface v2를 사용하여 DynamoDB 리소스 기반 정책 관리
- AWS SDK를 사용하여 Amazon DynamoDB의 성능 모니터링
- AWS SDK로 고급 DynamoDB 쿼리 작업 수행
- AWS SDK를 사용하여 DynamoDB에서 목록 작업 수행
- AWS SDK를 사용하여 DynamoDB에서 맵 작업 수행
- AWS SDK를 사용하여 DynamoDB에서 집합 작업 수행
- PartiQL 문 배치 및 AWS SDK를 사용하여 DynamoDB 테이블 쿼리
- PartiQL 및 AWS SDK를 사용하여 DynamoDB 테이블 쿼리

- AWS SDK로 글로벌 보조 인덱스를 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 begins_with 조건을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 정렬 키에 날짜 범위를 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 복잡한 필터 표현식을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 동적 필터 표현식을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 필터 표현식 및 제한을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 중첩 속성을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 페이지 매김을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK로 강력하게 일관된 읽기를 사용하여 DynamoDB 테이블 쿼리
- AWS SDK와 함께 PartiQL SELECT 문을 사용하여 DynamoDB 데이터 쿼리
- AWS SDK를 사용하여 TTL 항목에 대한 DynamoDB 테이블 쿼리
- AWS SDK로 날짜 및 시간 패턴을 사용하여 DynamoDB 테이블 쿼리
- AWS SDK를 사용하여 EXIF 및 기타 이미지 정보 저장
- AWS Command Line Interface v2를 사용하여 DynamoDB에 대한 속성 기반 액세스 제어 설정
- AWS SDK를 사용하여 DynamoDB에서 업데이트 표현식 순서 이해
- AWS SDK를 사용하여 웜 처리량으로 DynamoDB 테이블 설정 업데이트
- AWS SDK를 사용하여 TTL이 포함된 DynamoDB 항목 업데이트
- AWS SDK에서 PartiQL UPDATE 문을 사용하여 DynamoDB 데이터 업데이트
- API Gateway를 사용하여 Lambda 함수 간접 호출
- Step Functions를 사용하여 Lambda 함수 간접 호출
- AWS SDK를 사용하여 DynamoDB용 문서 모델 사용
- AWS SDK를 사용하여 DynamoDB용 상위 수준 객체 지속성 모델 사용
- AWS SDK를 사용하여 DynamoDB에서 원자성 카운터 작업 사용
- AWS SDK를 사용하여 DynamoDB에서 조건부 작업 사용
- AWS SDK를 사용하여 DynamoDB에서 표현식 속성 이름 사용
- 예약된 이벤트를 사용하여 Lambda 함수 간접 호출
- AWS Command Line Interface v2를 사용한 DynamoDB 로컬 보조 인덱스 작업
- AWS Command Line Interface v2를 사용한 DynamoDB Streams 및 Time-to-Live 작업
- AWS Command Line Interface v2를 사용한 DynamoDB 글로벌 테이블 및 최종 일관성(MREC)을 갖춘 다중 리전 복제 작업
- AWS Command Line Interface v2를 사용한 DynamoDB 리소스 태그 지정 작업

- AWS Command Line Interface v2를 사용한 DynamoDB 테이블 암호화 작업

## AWS SDK를 사용하여 DAX로 DynamoDB 읽기 가속화

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- DAX 클라이언트와 SDK 클라이언트를 모두 사용하여 데이터를 생성하고 테이블에 씁니다.
- 두 클라이언트를 모두 사용하여 테이블을 가져오고 쿼리하고 스캔하여 성능을 비교합니다.

자세한 내용은 DynamoDB Accelerator 클라이언트로 개발을 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

DAX 또는 Boto3 클라이언트를 사용하여 테이블을 생성합니다.

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
        "KeySchema": [
```

```
            {"AttributeName": "partition_key", "KeyType": "HASH"},
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "BillingMode": "PAY_PER_REQUEST",
    }
    table = dyn_resource.create_table(**params)
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table


if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

테이블에 테스트 데이터를 씁니다.

```
import boto3


def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate
 the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
```

```
            table.put_item(
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")


if __name__ == "__main__":
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

DAX 클라이언트와 Boto3 클라이언트를 사용하여 지정된 반복 횟수 만큼 항목을 가져오고 클라이언트마다 소요된 시간을 보고합니다.

```
import argparse
import sys
import time
import amazondax
import boto3


def get_item_test(key_count, iterations, dyn_resource=None):
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each
 iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
```

```
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(
                    Key={"partition_key": partition_key, "sort_key": sort_key}
                )
                print(".", end="")
                sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(
            f"Getting each item from the table {test_iterations} times, "
            f"using the DAX client."
        )
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax
            )
    else:
        print(
```

```
            f"Getting each item from the table {test_iterations} times, "
            f"using the Boto3 client."
        )
        test_start, test_end = get_item_test(test_key_count, test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/ test_iterations}."
    )
```

DAX 클라이언트와 Boto3 클라이언트를 사용하여 지정된 반복 횟수 만큼 테이블을 쿼리하고 클라이언트마다 소요된 시간을 보고합니다.

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key


def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
    Queries the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this
 value.
    :param sort_keys: The range of sort key values for the query. The query
 returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
```

```
    ).between(*sort_keys)

    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_partition_key = 5
    test_sort_keys = (2, 9)
    test_iterations = 100
    if args.endpoint_url:
        print(f"Querying the table {test_iterations} times, using the DAX
 client.")
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations,
 dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3
 client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )
```

```
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

DAX 클라이언트와 Boto3 클라이언트를 사용하여 지정된 반복 횟수 만큼 테이블을 스캔하고 클라이언트마다 소요된 시간을 보고합니다.

```
import argparse
import time
import sys
import amazondax
import boto3


def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
```

```
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX
 client.")
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3
 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

테이블을 삭제합니다.

```
import boto3


def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()
```

```
    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
    delete_dax_table()
    print("Table deleted!")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하세요.
  - CreateTable
  - DeleteTable
  - GetItem
  - PutItem
  - Query
  - Scan

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS Command Line Interface v2를 사용한 고급 DynamoDB 글로벌 보조 인덱스 시나리오 작업

다음 코드 예제는 고급 글로벌 보조 인덱스 구성으로 작업하는 방법을 보여줍니다.

- 여러 GSI가 있는 테이블을 생성합니다.
- 온디맨드 용량과 GSI를 사용하여 테이블을 생성합니다.
- 여러 GSI가 있는 테이블에 항목을 넣습니다.
- 다양한 조건을 사용하여 여러 GSI를 쿼리합니다.

Bash

Bash 스크립트와 함께 AWS CLI사용

여러 GSI가 있는 테이블을 생성합니다.

```
# Create a table with multiple GSIs
aws dynamodb create-table \
    --table-name MusicLibrary \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S \
        AttributeName=Genre,AttributeType=S \
        AttributeName=Year,AttributeType=N \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumIndex\",
                \"KeySchema\": [{\"AttributeName\":\"AlbumTitle\",\"KeyType\":
\"HASH\"}],
                \"Projection\": {\"ProjectionType\":\"ALL\"}
            },
            {
                \"IndexName\": \"GenreYearIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"Genre\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"Year\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {\"ProjectionType\":\"INCLUDE\",
\"NonKeyAttributes\":[\"Artist\",\"SongTitle\"]}
            }
        ]"
```

온디맨드 용량과 GSI를 사용하여 테이블을 생성합니다.

```
# Create a table with on-demand capacity and GSI
aws dynamodb create-table \
    --table-name MusicOnDemand \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=Genre,AttributeType=S \
    --key-schema \
```

```
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GenreIndex\",
                \"KeySchema\": [{\"AttributeName\":\"Genre\",\"KeyType\":\"HASH
\"}],
                \"Projection\": {\"ProjectionType\":\"ALL\"}
            }
        ]"
```

여러 GSI가 있는 테이블에 항목을 넣습니다.

```
# Add items to MusicLibrary table
aws dynamodb put-item \
    --table-name MusicLibrary \
    --item '{
        "Artist": {"S": "The Beatles"},
        "SongTitle": {"S": "Hey Jude"},
        "AlbumTitle": {"S": "Past Masters"},
        "Genre": {"S": "Rock"},
        "Year": {"N": "1968"}
    }'

aws dynamodb put-item \
    --table-name MusicLibrary \
    --item '{
        "Artist": {"S": "Miles Davis"},
        "SongTitle": {"S": "So What"},
        "AlbumTitle": {"S": "Kind of Blue"},
        "Genre": {"S": "Jazz"},
        "Year": {"N": "1959"}
    }'
```

여러 GSI가 있는 테이블에서 항목을 쿼리합니다.

```
# Query the AlbumIndex GSI
echo "Querying AlbumIndex GSI:"
aws dynamodb query \
```

```
    --table-name MusicLibrary \
    --index-name AlbumIndex \
    --key-condition-expression "AlbumTitle = :album" \
    --expression-attribute-values '{":album":{"S":"Kind of Blue"}}'

# Query the GenreYearIndex GSI with a range condition
echo "Querying GenreYearIndex GSI with range condition:"
aws dynamodb query \
    --table-name MusicLibrary \
    --index-name GenreYearIndex \
    --key-condition-expression "Genre = :genre AND #yr > :year" \
    --expression-attribute-names '{"#yr": "Year"}' \
    --expression-attribute-values '{":genre":{"S":"Rock"},":year":{"N":"1965"}}'
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - CreateTable
  - PutItem
  - Query

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## DynamoDB 테이블에 데이터를 제출하기 위한 애플리케이션 구축

다음 코드 예제에서는 Amazon DynamoDB 테이블에 데이터를 제출하고 사용자가 테이블을 업데이트할 때 알려주는 애플리케이션을 구축하는 방법을 보여줍니다.

Java

SDK for Java 2.x

Amazon DynamoDB Java API를 사용하여 데이터를 제출하고 Amazon Simple Notification Service Java API를 사용하여 문자 메시지를 전송하는 동적 웹 애플리케이션을 생성하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB

- Amazon SNS

JavaScript

SDK for JavaScript (v3)

이 예제에서는 사용자가 Amazon DynamoDB 테이블에 데이터를 제출하고 Amazon Simple Notification Service(Amazon SNS)를 사용하여 관리자에게 문자 메시지를 전송하는 앱을 구축하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예시는 AWS SDK for JavaScript v3 개발자 안내서에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon SNS

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 DynamoDB에서 단일 속성과 여러 값 비교

다음 코드 예제에서는 DynamoDB에서 단일 속성과 여러 값을 비교하는 방법을 보여줍니다.

- IN 연산자를 사용하여 단일 속성과 여러 값을 비교합니다.
- IN 연산자를 여러 OR 조건과 비교합니다.
- IN을 사용할 때 성능 및 표현식 복잡성 측면의 이점을 이해합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용하여 DynamoDB에서 단일 속성과 여러 값을 비교합니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;

    /**
     * Queries a table using the IN operator to compare multiple values with a
 single attribute.
     *
     * <p>This method demonstrates how to use the IN operator in a filter
 expression
     * to match an attribute against multiple values.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The query response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse compareMultipleValues(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);
```

```
        // Add values for IN operator
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the IN clause
        StringBuilder inClause = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                inClause.append(", ");
            }
            inClause.append(":val").append(i);
        }

        // Define the query parameters
        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression("#attrName IN (" + inClause.toString() + ")")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Queries a table using multiple OR conditions to compare multiple values
with a single attribute.
     *
     * <p>This method demonstrates the alternative approach to using the IN
operator,
     * by using multiple OR conditions.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The query response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
```

```
    public static QueryResponse compareWithOrConditions(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);

        // Add values for OR conditions
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the OR conditions
        StringBuilder orConditions = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                orConditions.append(" OR ");
            }
            orConditions.append("#attrName = :val").append(i);
        }

        // Define the query parameters
        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression(orConditions.toString())
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }
```

```
   /**
    * Compares the performance of using the IN operator versus multiple OR
conditions.
    *
    * <p>This method demonstrates the performance difference between using the
IN operator
    * and using multiple OR conditions.
    *
    * @param dynamoDbClient The DynamoDB client
    * @param tableName The name of the DynamoDB table
    * @param partitionKeyName The name of the partition key attribute
    * @param partitionKeyValue The value of the partition key to query
    * @param attributeName The name of the attribute to compare
    * @param valuesList List of values to compare against
    * @return Map containing the performance comparison results
    */
   public static Map<String, Object> comparePerformance(
       DynamoDbClient dynamoDbClient,
       String tableName,
       String partitionKeyName,
       AttributeValue partitionKeyValue,
       String attributeName,
       List<AttributeValue> valuesList) {

       Map<String, Object> results = new HashMap<>();

       try {
           // Measure performance of IN operator
           long inStartTime = System.nanoTime();
           QueryResponse inResponse = compareMultipleValues(
               dynamoDbClient, tableName, partitionKeyName, partitionKeyValue,
attributeName, valuesList);
           long inEndTime = System.nanoTime();
           long inDuration = inEndTime - inStartTime;

           // Measure performance of OR conditions
           long orStartTime = System.nanoTime();
           QueryResponse orResponse = compareWithOrConditions(
               dynamoDbClient, tableName, partitionKeyName, partitionKeyValue,
attributeName, valuesList);
           long orEndTime = System.nanoTime();
           long orDuration = orEndTime - orStartTime;

           // Record results
```

```
            results.put("inOperatorDuration", inDuration);
            results.put("orConditionsDuration", orDuration);
            results.put("inOperatorItems", inResponse.count());
            results.put("orConditionsItems", orResponse.count());
            results.put("inOperatorExpression", "IN operator with " +
valuesList.size() + " values");
            results.put("orConditionsExpression", valuesList.size() + " OR
conditions");
            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Scans a table using the IN operator with a large number of values.
     *
     * <p>This method demonstrates how to use the IN operator with a large number
of values,
     * which can help stay within the 300 operator limit.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The scan response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static ScanResponse scanWithLargeInClause(
        DynamoDbClient dynamoDbClient, String tableName, String attributeName,
List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

        // Add values for IN operator
```

```
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the IN clause
        StringBuilder inClause = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                inClause.append(", ");
            }
            inClause.append(":val").append(i);
        }

        // Define the scan parameters
        ScanRequest request = ScanRequest.builder()
            .tableName(tableName)
            .filterExpression("#attrName IN (" + inClause.toString() + ")")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the scan operation
        return dynamoDbClient.scan(request);
    }

    /**
     * Generates a list of sample values for testing.
     *
     * <p>Helper method to generate a list of sample values for testing.
     *
     * @param valueType The type of values to generate (string, number, or
boolean)
     * @param count The number of values to generate
     * @return List of generated attribute values
     */
    public static List<AttributeValue> generateSampleValues(String valueType, int
count) {
        List<AttributeValue> values = new ArrayList<>();

        for (int i = 0; i < count; i++) {
            AttributeValue value;

            switch (valueType.toLowerCase(Locale.ROOT)) {
                case "string":
```

```
                    value = AttributeValue.builder().s("Value" + i).build();
                    break;
                case "number":
                    value =
AttributeValue.builder().n(String.valueOf(i)).build();
                    break;
                case "boolean":
                    value = AttributeValue.builder().bool(i % 2 == 0).build();
                    break;
                default:
                    throw new IllegalArgumentException("Unsupported value type: "
+ valueType);
            }

            values.add(value);
        }

        return values;
    }
```

AWS SDK for Java 2.x를 사용하여 여러 값을 비교하는 사용 예시입니다.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        System.out.println("Demonstrating how to compare multiple values with a
single attribute in DynamoDB");

        try {
            // Example 1: Using the IN operator
            System.out.println("\nExample 1: Using the IN operator");
            List<AttributeValue> categories = List.of(
                AttributeValue.builder().s("Electronics").build(),
                AttributeValue.builder().s("Computers").build(),
                AttributeValue.builder().s("Accessories").build());

            QueryResponse inResponse = compareMultipleValues(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);
```

```
            System.out.println("Found " + inResponse.count() + " items using IN
operator");
            System.out.println("Items: " + inResponse.items());

            // Example 2: Using multiple OR conditions
            System.out.println("\nExample 2: Using multiple OR conditions");
            QueryResponse orResponse = compareWithOrConditions(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            System.out.println("Found " + orResponse.count() + " items using OR
conditions");
            System.out.println("Items: " + orResponse.items());

            // Example 3: Performance comparison
            System.out.println("\nExample 3: Performance comparison");
            Map<String, Object> perfComparison = comparePerformance(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            if ((boolean) perfComparison.get("success")) {
                System.out.println("IN operator duration: " +
perfComparison.get("inOperatorDuration") + " ns");
                System.out.println("OR conditions duration: " +
perfComparison.get("orConditionsDuration") + " ns");
                System.out.println("IN operator found " +
perfComparison.get("inOperatorItems") + " items");
                System.out.println("OR conditions found " +
perfComparison.get("orConditionsItems") + " items");
                System.out.println("Expression complexity comparison:");
                System.out.println("  IN operator: " +
perfComparison.get("inOperatorExpression"));
                System.out.println("  OR conditions: " +
perfComparison.get("orConditionsExpression"));
            } else {
```

```
                System.out.println("Performance comparison failed: " +
perfComparison.get("error"));
            }

            // Example 4: Using IN with a large number of values
            System.out.println("\nExample 4: Using IN with a large number of
values");
            List<AttributeValue> productIds = generateSampleValues("string", 20);

            ScanResponse largeInResponse = scanWithLargeInClause(dynamoDbClient,
tableName, "ProductId", productIds);

            System.out.println(
                "Found " + largeInResponse.count() + " items using IN with " +
productIds.size() + " values");

            // Explain the benefits of using IN
            System.out.println("\nKey points about using the IN operator in
DynamoDB:");
            System.out.println("1. The IN operator allows comparing a single
attribute against multiple values");
            System.out.println("2. IN is more concise than using multiple OR
conditions");
            System.out.println("3. IN counts as only 1 operator regardless of the
number of values");
            System.out.println("4. Multiple OR conditions count as 1 operator per
condition plus 1 per OR");
            System.out.println("5. Using IN helps stay within the 300 operator
limit for complex expressions");
            System.out.println("6. IN can be used in filter expressions and
condition expressions");
            System.out.println("7. The IN operator supports up to 100 comparison
values");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - Query

- Scan

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용하여 단일 속성과 여러 값을 비교합니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  ScanCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Query or scan a DynamoDB table to find items where an attribute matches any
 value from a list.
 *
 * This function demonstrates the use of the IN operator to compare a single
 attribute
 * against multiple possible values, which is more efficient than using multiple
 OR conditions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against
 the values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key
 attribute for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the
 matching items
 */
async function compareMultipleValues(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
```

```
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the filter expression using the IN operator
  const filterExpression = `${attributeName} IN (${valuesList.map((_, index) =>
 `:val${index}`).join(', ')})`;

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous
 query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
```

```
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
```

```
      Count: allItems.length
    };
  }
}

/**
 * Alternative implementation using multiple OR conditions instead of the IN
 operator.
 *
 * This function is provided for comparison to show why using the IN operator is
 preferable.
 * With many values, this approach becomes verbose and less efficient.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against
 the values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key
 attribute for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the
 matching items
 */
async function compareWithOrConditions(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // If no values provided, return empty result
  if (!valuesList || valuesList.length === 0) {
    return {
      Items: [],
      Count: 0
    };
  }
```

```
  // Create the filter expression using multiple OR conditions
  const filterConditions = valuesList.map((_, index) => `${attributeName} = :val
${index}`);
  const filterExpression = filterConditions.join(' OR ');

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous
 query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
```

```
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
```

```
}
```

AWS SDK for JavaScript를 사용하여 여러 값을 비교하는 사용 예시입니다.

```
/**
 * Example of how to use the compareMultipleValues function.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const attributeName = "Category";
  const valuesList = ["Electronics", "Computers", "Accessories"];

  console.log(`Searching for products in any of these categories:
 ${valuesList.join(', ')}`);

  try {
    // Using the IN operator (recommended approach)
    console.log("\nApproach 1: Using the IN operator");
    const response = await compareMultipleValues(
      config,
      tableName,
      attributeName,
      valuesList
    );

    console.log(`Found ${response.Count} products in the specified categories`);

    // Using multiple OR conditions (alternative approach)
    console.log("\nApproach 2: Using multiple OR conditions");
    const response2 = await compareWithOrConditions(
      config,
      tableName,
      attributeName,
      valuesList
    );

    console.log(`Found ${response2.Count} products in the specified categories`);

    // Example with a query operation
```

```
    console.log("\nQuerying a specific manufacturer's products in multiple
 categories");
    const partitionKeyName = "Manufacturer";
    const partitionKeyValue = "Acme";

    const response3 = await compareMultipleValues(
      config,
      tableName,
      attributeName,
      valuesList,
      partitionKeyName,
      partitionKeyValue
    );

    console.log(`Found ${response3.Count} Acme products in the specified
 categories`);

    // Explain the benefits of using the IN operator
    console.log("\nBenefits of using the IN operator:");
    console.log("1. More concise expression compared to multiple OR conditions");
    console.log("2. Better readability and maintainability");
    console.log("3. Potentially better performance with large value lists");
    console.log("4. Simpler code that's less prone to errors");
    console.log("5. Easier to modify when adding or removing values");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - Query
  - Scan

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용하여 단일 속성과 여러 값을 비교합니다.

```
import boto3
```

```
from boto3.dynamodb.conditions import Attr, Key
from typing import Any, Dict, List, Optional


def compare_multiple_values(
    table_name: str,
    attribute_name: str,
    values_list: List[Any],
    partition_key_name: Optional[str] = None,
    partition_key_value: Optional[str] = None,
) -> Dict[str, Any]:
    """
    Query or scan a DynamoDB table to find items where an attribute matches any
 value from a list.

    This function demonstrates the use of the IN operator to compare a single
 attribute
    against multiple possible values, which is more efficient than using multiple
 OR conditions.

    Args:
        table_name (str): The name of the DynamoDB table.
        attribute_name (str): The name of the attribute to compare against the
 values list.
        values_list (List[Any]): List of values to compare the attribute against.
        partition_key_name (Optional[str]): The name of the partition key
 attribute for query operations.
        partition_key_value (Optional[str]): The value of the partition key to
 query.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the matching items.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create the filter expression using the is_in method
    filter_expression = Attr(attribute_name).is_in(values_list)

    # If partition key is provided, perform a query operation
    if partition_key_name and partition_key_value:
        key_condition = Key(partition_key_name).eq(partition_key_value)
        response = table.query(
```

```
            KeyConditionExpression=key_condition,
 FilterExpression=filter_expression
        )
    else:
        # Otherwise, perform a scan operation
        response = table.scan(FilterExpression=filter_expression)

    # Handle pagination if there are more results
    items = response.get("Items", [])
    while "LastEvaluatedKey" in response:
        if partition_key_name and partition_key_value:
            response = table.query(
                KeyConditionExpression=key_condition,
                FilterExpression=filter_expression,
                ExclusiveStartKey=response["LastEvaluatedKey"],
            )
        else:
            response = table.scan(
                FilterExpression=filter_expression,
 ExclusiveStartKey=response["LastEvaluatedKey"]
            )
        items.extend(response.get("Items", []))

    # Return the complete result
    return {"Items": items, "Count": len(items)}


def compare_with_or_conditions(
    table_name: str,
    attribute_name: str,
    values_list: List[Any],
    partition_key_name: Optional[str] = None,
    partition_key_value: Optional[str] = None,
) -> Dict[str, Any]:
    """
    Alternative implementation using multiple OR conditions instead of the IN
 operator.

    This function is provided for comparison to show why using the IN operator is
 preferable.
    With many values, this approach becomes verbose and less efficient.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        attribute_name (str): The name of the attribute to compare against the
values list.
        values_list (List[Any]): List of values to compare the attribute against.
        partition_key_name (Optional[str]): The name of the partition key
attribute for query operations.
        partition_key_value (Optional[str]): The value of the partition key to
query.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the matching items.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create a filter expression with multiple OR conditions
    filter_expression = None
    for value in values_list:
        condition = Attr(attribute_name).eq(value)
        if filter_expression is None:
            filter_expression = condition
        else:
            filter_expression = filter_expression | condition

    # If partition key is provided, perform a query operation
    if partition_key_name and partition_key_value and filter_expression:
        key_condition = Key(partition_key_name).eq(partition_key_value)
        response = table.query(
            KeyConditionExpression=key_condition,
FilterExpression=filter_expression
        )
    elif filter_expression:
        # Otherwise, perform a scan operation
        response = table.scan(FilterExpression=filter_expression)
    else:
        # Return empty response if no values provided
        return {"Items": [], "Count": 0}

    # Handle pagination if there are more results
    items = response.get("Items", [])
    while "LastEvaluatedKey" in response:
        if partition_key_name and partition_key_value:
            response = table.query(
                KeyConditionExpression=key_condition,
```

```
                FilterExpression=filter_expression,
                ExclusiveStartKey=response["LastEvaluatedKey"],
            )
        else:
            response = table.scan(
                FilterExpression=filter_expression,
 ExclusiveStartKey=response["LastEvaluatedKey"]
            )
        items.extend(response.get("Items", []))

    # Return the complete result
    return {"Items": items, "Count": len(items)}
```

AWS SDK for Python (Boto3)를 사용하여 여러 값을 비교하는 사용 예시입니다.

```
def example_usage():
    """Example of how to use the compare_multiple_values function."""
    # Example parameters
    table_name = "Products"
    attribute_name = "Category"
    values_list = ["Electronics", "Computers", "Accessories"]

    print(f"Searching for products in any of these categories: {values_list}")

    # Using the IN operator (recommended approach)
    print("\nApproach 1: Using the IN operator")
    response = compare_multiple_values(
        table_name=table_name, attribute_name=attribute_name,
 values_list=values_list
    )

    print(f"Found {response['Count']} products in the specified categories")

    # Using multiple OR conditions (alternative approach)
    print("\nApproach 2: Using multiple OR conditions")
    response2 = compare_with_or_conditions(
        table_name=table_name, attribute_name=attribute_name,
 values_list=values_list
    )
```

```
    print(f"Found {response2['Count']} products in the specified categories")

    # Example with a query operation
    print("\nQuerying a specific manufacturer's products in multiple categories")
    partition_key_name = "Manufacturer"
    partition_key_value = "Acme"

    response3 = compare_multiple_values(
        table_name=table_name,
        attribute_name=attribute_name,
        values_list=values_list,
        partition_key_name=partition_key_name,
        partition_key_value=partition_key_value,
    )

    print(f"Found {response3['Count']} Acme products in the specified
categories")

    # Explain the benefits of using the IN operator
    print("\nBenefits of using the IN operator:")
    print("1. More concise expression compared to multiple OR conditions")
    print("2. Better readability and maintainability")
    print("3. Potentially better performance with large value lists")
    print("4. Simpler code that's less prone to errors")
    print("5. Easier to modify when adding or removing values")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하세요.
  - Query
  - Scan

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 TTL로 DynamoDB 항목을 조건부로 업데이트

다음 코드 예제는 항목의 TTL을 조건부로 업데이트하는 방법을 보여줍니다.

Java

SDK for Java 2.x

조건을 사용하여 테이블의 기존 DynamoDB 항목에서 TTL을 업데이트합니다.

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Map;
import java.util.Optional;

/**
 * Updates an item in a DynamoDB table with TTL attributes using a conditional
 expression.
 * This class demonstrates how to conditionally update TTL expiration timestamps.
 */
public class UpdateTTLConditional {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
```

```
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String UPDATED_AT_ATTR = "updatedAt";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String UPDATE_EXPRESSION = "SET " + UPDATED_AT_ATTR +
"=:c, " + EXPIRE_AT_ATTR + "=:e";
    private static final String CONDITION_EXPRESSION = "attribute_exists(" +
PRIMARY_KEY_ATTR + ")";
    private static final String SUCCESS_MESSAGE = "%s UpdateItem operation with
TTL successful.";
    private static final String CONDITION_FAILED_MESSAGE = "Condition check
failed. Item does not exist.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs an UpdateTTLConditional with a default DynamoDB client.
     */
    public UpdateTTLConditional() {
        this.dynamoDbClient = null;
    }

    /**
     * Constructs an UpdateTTLConditional with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public UpdateTTLConditional(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Main method to demonstrate conditionally updating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new UpdateTTLConditional().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and conditionally update an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = Map.of(
            PRIMARY_KEY_ATTR, AttributeValue.builder().s(primaryKey).build(),
            SORT_KEY_ATTR, AttributeValue.builder().s(sortKey).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = Map.of(
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build(),
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());
```

```
        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .conditionExpression(CONDITION_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final UpdateItemResponse response = ddb.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return 0;
        } catch (ConditionalCheckFailedException e) {
            System.err.println(CONDITION_FAILED_MESSAGE);
            throw e;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

조건을 사용하여 테이블의 기존 DynamoDB 항목에서 TTL을 업데이트합니다.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItemConditional = async (tableName, partitionKey, sortKey,
 region = 'us-east-1', newAttribute = 'default-value') => {
    const client = new DynamoDBClient({
```

```
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-
sort-key-value');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

Python

SDK for Python(Boto3)

조건을 사용하여 테이블의 기존 DynamoDB 항목에서 TTL을 업데이트합니다.

```
from datetime import datetime, timedelta

import boto3
from botocore.exceptions import ClientError


def update_dynamodb_item_ttl(table_name, region, primary_key, sort_key,
 ttl_attribute):
    """
    Updates an existing record in a DynamoDB table with a new or updated TTL
 attribute.

    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :param ttl_attribute: name of the TTL attribute in the target DynamoDB table
    :return:
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Generate updated TTL in epoch second format
        updated_expiration_time = int((datetime.now() +
 timedelta(days=90)).timestamp())

        # Define the update expression for adding/updating a new attribute
        update_expression = "SET newAttribute = :val1"

        # Define the condition expression for checking if 'expireAt' is not
 expired
        condition_expression = "expireAt > :val2"

        # Define the expression attribute values
        expression_attribute_values = {":val1": ttl_attribute, ":val2":
 updated_expiration_time}
```

```
        response = table.update_item(
            Key={"primaryKey": primary_key, "sortKey": sort_key},
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeValues=expression_attribute_values,
        )

        print("Item updated successfully.")
        return response["ResponseMetadata"]["HTTPStatusCode"]  # Ideally a 200 OK
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            print("Condition check failed: Item's 'expireAt' is expired.")
        else:
            print(f"Error updating item: {e}")
    except Exception as e:
        print(f"Error updating item: {e}")


# replace with your values
update_dynamodb_item_ttl(
    "your-table-name",
    "us-east-1",
    "your-partition-key-value",
    "your-sort-key-value",
    "your-ttl-attribute-value",
)
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 로컬 DynamoDB 인스턴스에 연결

다음 코드 예제는 DynamoDB 및 AWS SDK의 로컬 개발 배포에 연결하기 위해 엔드포인트 URL을 재정의하는 방법을 보여줍니다.

자세한 내용은 DynamoDB Local을 참조하세요.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// Lists your tables from a local DynamoDB instance by setting the SDK Config's
/// endpoint_url and test_credentials.
#[tokio::main]
async fn main() {
    tracing_subscriber::fmt::init();

    let config = aws_config::defaults(aws_config::BehaviorVersion::latest())
        .test_credentials()
        // DynamoDB run locally uses port 8000 by default.
        .endpoint_url("http://localhost:8000")
        .load()
        .await;
    let dynamodb_local_config =
 aws_sdk_dynamodb::config::Builder::from(&config).build();

    let client = aws_sdk_dynamodb::Client::from_conf(dynamodb_local_config);

    let list_resp = client.list_tables().send().await;
    match list_resp {
        Ok(resp) => {
            println!("Found {} tables", resp.table_names().len());
            for name in resp.table_names() {
                println!("  {}", name);
            }
        }
        Err(err) => eprintln!("Failed to list local dynamodb tables: {err:?}"),
    }
}
```

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

# AWS SDK를 사용하여 DynamoDB에서 표현식 연산자 수 계산

다음 코드 예제에서는 DynamoDB에서 표현식 연산자 수를 계산하는 방법을 보여줍니다.

- DynamoDB의 연산자 제한(300개)을 이해합니다.
- 복잡한 표현식에서 연산자 수를 계산합니다.
- 제한 범위를 벗어나지 않도록 표현식을 최적화합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용한 표현식 연산자 수 계산을 보여줍니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

    /**
     * Creates a complex filter expression with a specified number of conditions.
     *
     * <p>This method demonstrates how to generate a complex expression with
     * a specific number of operators to test the 300 operator limit.
     *
     * @param conditionsCount Number of conditions to include
     * @param useAnd Whether to use AND (true) or OR (false) between conditions
     * @return Map containing the filter expression, attribute values, and
 operator count
     */
    public static Map<String, Object> createComplexFilterExpression(int
 conditionsCount, boolean useAnd) {
        // Initialize the expression parts and attribute values
```

```
        StringBuilder filterExpression = new StringBuilder();
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

        // Generate the specified number of conditions
        for (int i = 0; i < conditionsCount; i++) {
            // Add the operator between conditions (except for the first one)
            if (i > 0) {
                filterExpression.append(useAnd ? " AND " : " OR ");
            }

            // Alternate between different comparison operators for variety
            String valueKey = ":val" + i;

            switch (i % 5) {
                case 0:
                    filterExpression.append("attribute").append(i).append(" =
").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey, AttributeValue.builder().s("value" +
i).build());
                    break;
                case 1:
                    filterExpression.append("attribute").append(i).append(" >
").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey,
AttributeValue.builder().n(String.valueOf(i)).build());
                    break;
                case 2:
                    filterExpression.append("attribute").append(i).append(" <
").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey,
                        AttributeValue.builder().n(String.valueOf(i *
10)).build());
                    break;
                case 3:
                    filterExpression
                        .append("contains(attribute")
                        .append(i)
                        .append(", ")
                        .append(valueKey)
                        .append(")");
                    expressionAttributeValues.put(
```

```
                        valueKey, AttributeValue.builder().s("substring" +
i).build());
                    break;
                case 4:
                    filterExpression
                        .append("attribute_exists(attribute")
                        .append(i)
                        .append(")");
                    break;
                default:
                    // This case will never be reached, but added to satisfy
checkstyle
                    break;
            }
        }

        // Calculate the operator count
        // Each condition has 1 operator (=, >, <, contains, attribute_exists)
        // Each AND or OR between conditions is 1 operator
        int operatorCount = conditionsCount + (conditionsCount > 0 ?
conditionsCount - 1 : 0);

        // Create the result map
        Map<String, Object> result = new HashMap<>();
        result.put("filterExpression", filterExpression.toString());
        result.put("expressionAttributeValues", expressionAttributeValues);
        result.put("operatorCount", operatorCount);

        return result;
    }

    /**
     * Creates a complex update expression with a specified number of operations.
     *
     * <p>This method demonstrates how to generate a complex update expression
with
     * a specific number of operators to test the 300 operator limit.
     *
     * @param operationsCount Number of operations to include
     * @return Map containing the update expression, attribute values, and
operator count
     */
    public static Map<String, Object> createComplexUpdateExpression(int
operationsCount) {
```

```
        // Initialize the expression parts and attribute values
        StringBuilder updateExpression = new StringBuilder("SET ");
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

        // Generate the specified number of SET operations
        for (int i = 0; i < operationsCount; i++) {
            // Add comma between operations (except for the first one)
            if (i > 0) {
                updateExpression.append(", ");
            }

            // Alternate between different types of SET operations
            String valueKey = ":val" + i;

            switch (i % 3) {
                case 0:
                    // Simple assignment (1 operator: =)
                    updateExpression.append("attribute").append(i).append(" =
").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey, AttributeValue.builder().s("value" +
i).build());
                    break;
                case 1:
                    // Addition (2 operators: = and +)
                    updateExpression
                        .append("attribute")
                        .append(i)
                        .append(" = attribute")
                        .append(i)
                        .append(" + ")
                        .append(valueKey);
                    expressionAttributeValues.put(
                        valueKey,
AttributeValue.builder().n(String.valueOf(i)).build());
                    break;
                case 2:
                    // Conditional assignment with if_not_exists (2 operators: =
and if_not_exists)
                    updateExpression
                        .append("attribute")
                        .append(i)
                        .append(" = if_not_exists(attribute")
                        .append(i)
```

```
                        .append(", ")
                        .append(valueKey)
                        .append(")");
                    expressionAttributeValues.put(
                        valueKey,
                        AttributeValue.builder().n(String.valueOf(i *
10)).build());
                    break;
                default:
                    // This case will never be reached, but added to satisfy
checkstyle
                    break;
            }
        }

        // Calculate the operator count
        // Each operation has 1-2 operators as noted above
        int operatorCount = 0;
        for (int i = 0; i < operationsCount; i++) {
            operatorCount += (i % 3 == 0) ? 1 : 2;
        }

        // Create the result map
        Map<String, Object> result = new HashMap<>();
        result.put("updateExpression", updateExpression.toString());
        result.put("expressionAttributeValues", expressionAttributeValues);
        result.put("operatorCount", operatorCount);

        return result;
    }

    /**
     * Test the operator limit by attempting an operation with a complex
expression.
     *
     * <p>This method demonstrates what happens when an expression approaches or
     * exceeds the 300 operator limit.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param operatorCount Target number of operators to include
     * @return Map containing the result of the operation attempt
     */
```

```
   public static Map<String, Object> testOperatorLimit(
       DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, int operatorCount) {

       // Create a complex update expression with the specified operator count
       Map<String, Object> expressionData =
           createComplexUpdateExpression((int) Math.ceil(operatorCount /
1.5)); // Adjust to get close to target count

       String updateExpression = (String)
expressionData.get("updateExpression");
       @SuppressWarnings("unchecked")
       Map<String, AttributeValue> expressionAttributeValues =
           (Map<String, AttributeValue>)
expressionData.get("expressionAttributeValues");
       int actualCount = (int) expressionData.get("operatorCount");

       System.out.println("Generated update expression with approximately " +
actualCount + " operators");

       // Define the update parameters
       UpdateItemRequest request = UpdateItemRequest.builder()
           .tableName(tableName)
           .key(key)
           .updateExpression(updateExpression)
           .expressionAttributeValues(expressionAttributeValues)
           .returnValues("UPDATED_NEW")
           .build();

       try {
           // Attempt the update operation
           UpdateItemResponse response = dynamoDbClient.updateItem(request);

           Map<String, Object> result = new HashMap<>();
           result.put("success", true);
           result.put("message", "Operation succeeded with " + actualCount + "
operators");
           result.put("data", response);
           return result;

       } catch (DynamoDbException e) {
           // Check if the error is due to exceeding the operator limit
           if (e.getMessage().contains("too many operators")) {
               Map<String, Object> result = new HashMap<>();
```

```
                result.put("success", false);
                result.put("message", "Operation failed: " + e.getMessage());
                result.put("operatorCount", actualCount);
                return result;
            }

            // Return other errors
            Map<String, Object> result = new HashMap<>();
            result.put("success", false);
            result.put("message", "Operation failed: " + e.getMessage());
            result.put("error", e);
            return result;
        }
    }

    /**
     * Break down a complex expression into multiple simpler operations.
     *
     * <p>This method demonstrates how to handle expressions that would exceed
     * the 300 operator limit by breaking them into multiple operations.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param totalOperations Total number of operations to perform
     * @return Map containing the results of the operations
     */
    public static Map<String, Object> breakDownComplexExpression(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, int totalOperations) {

        // Calculate how many operations we can safely include in each batch
        // Using 150 as a conservative limit (well below 300)
        final int operationsPerBatch = 100;
        final int batchCount = (int) Math.ceil((double) totalOperations /
operationsPerBatch);

        System.out.println("Breaking down " + totalOperations + " operations into
" + batchCount + " batches");

        Map<String, Object> results = new HashMap<>();
        results.put("totalBatches", batchCount);

        Map<Integer, Map<String, Object>> batchResults = new HashMap<>();
```

```
        // Process each batch
        for (int batch = 0; batch < batchCount; batch++) {
            // Calculate the operations for this batch
            int batchStart = batch * operationsPerBatch;
            int batchEnd = Math.min(batchStart + operationsPerBatch,
totalOperations);
            int batchSize = batchEnd - batchStart;

            System.out.println(
                "Processing batch " + (batch + 1) + "/" + batchCount + " with " +
batchSize + " operations");

            // Create an update expression for this batch
            Map<String, Object> expressionData =
createComplexUpdateExpression(batchSize);

            String updateExpression = (String)
expressionData.get("updateExpression");
            @SuppressWarnings("unchecked")
            Map<String, AttributeValue> expressionAttributeValues =
                (Map<String, AttributeValue>)
expressionData.get("expressionAttributeValues");
            int operatorCount = (int) expressionData.get("operatorCount");

            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression(updateExpression)
                .expressionAttributeValues(expressionAttributeValues)
                .returnValues("UPDATED_NEW")
                .build();

            try {
                // Perform the update operation for this batch
                UpdateItemResponse response = dynamoDbClient.updateItem(request);

                Map<String, Object> batchResult = new HashMap<>();
                batchResult.put("batch", batch + 1);
                batchResult.put("success", true);
                batchResult.put("operatorCount", operatorCount);
                batchResult.put("attributes", response.attributes());
```

```
                batchResults.put(batch, batchResult);

            } catch (DynamoDbException e) {
                Map<String, Object> batchResult = new HashMap<>();
                batchResult.put("batch", batch + 1);
                batchResult.put("success", false);
                batchResult.put("operatorCount", operatorCount);
                batchResult.put("error", e.getMessage());

                batchResults.put(batch, batchResult);

                // Continue with next batch instead of breaking
                continue;
            }
        }

        results.put("results", batchResults);
        return results;
    }

    /**
     * Count operators in a DynamoDB expression based on the rules in the
documentation.
     *
     * <p>This method demonstrates how operators are counted according to the
     * DynamoDB documentation.
     *
     * @param expression The DynamoDB expression to analyze
     * @return Map containing the breakdown of operator counts
     */
    public static Map<String, Integer> countOperatorsInExpression(String
expression) {
        // Initialize counters for different operator types
        Map<String, Integer> counts = new HashMap<>();
        counts.put("comparisonOperators", 0);
        counts.put("logicalOperators", 0);
        counts.put("functions", 0);
        counts.put("arithmeticOperators", 0);
        counts.put("specialOperators", 0);
        counts.put("total", 0);

        // Count comparison operators (=, <>, <, <=, >, >=)
        // This is a simplified approach and may not catch all cases
        int comparisonCount = 0;
```

```
        Pattern comparisonPattern = Pattern.compile("(=|<>|<=|>=|<|>)");
        Matcher comparisonMatcher = comparisonPattern.matcher(expression);
        while (comparisonMatcher.find()) {
            comparisonCount++;
        }
        counts.put("comparisonOperators", comparisonCount);

        // Count logical operators (AND, OR, NOT)
        int andCount = countOccurrences(expression, "\\bAND\\b");
        int orCount = countOccurrences(expression, "\\bOR\\b");
        int notCount = countOccurrences(expression, "\\bNOT\\b");
        counts.put("logicalOperators", andCount + orCount + notCount);

        // Count functions (attribute_exists, attribute_not_exists,
 attribute_type, begins_with, contains, size)
        int functionCount = countOccurrences(
            expression,
            "\\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\\(");
        counts.put("functions", functionCount);

        // Count arithmetic operators (+ and -)
        // This is a simplified approach and may not catch all cases
        int arithmeticCount = 0;
        Pattern arithmeticPattern = Pattern.compile("[a-zA-Z0-9_)\\]]\\s*[\\+\
\-]\\s*[a-zA-Z0-9_:(]");
        Matcher arithmeticMatcher = arithmeticPattern.matcher(expression);
        while (arithmeticMatcher.find()) {
            arithmeticCount++;
        }
        counts.put("arithmeticOperators", arithmeticCount);

        // Count special operators (BETWEEN, IN)
        int betweenCount = countOccurrences(expression, "\\bBETWEEN\\b");
        int inCount = countOccurrences(expression, "\\bIN\\b");
        counts.put("specialOperators", betweenCount + inCount);

        // Add extra operators for BETWEEN (each BETWEEN includes an AND)
        int currentLogicalOps = counts.getOrDefault("logicalOperators", 0);
        counts.put("logicalOperators", currentLogicalOps + betweenCount);

        // Calculate total
        int total = counts.getOrDefault("comparisonOperators", 0)
            + counts.getOrDefault("logicalOperators", 0)
```

```
            + counts.getOrDefault("functions", 0)
            + counts.getOrDefault("arithmeticOperators", 0)
            + counts.getOrDefault("specialOperators", 0);
        counts.put("total", total);

        return counts;
    }

    /**
     * Helper method to count occurrences of a pattern in a string.
     *
     * @param text The text to search in
     * @param regex The regular expression pattern to search for
     * @return The number of occurrences
     */
    private static int countOccurrences(String text, String regex) {
        final Pattern pattern = Pattern.compile(regex);
        final Matcher matcher = pattern.matcher(text);
        int count = 0;
        while (matcher.find()) {
            count++;
        }
        return count;
    }
```

AWS SDK for Java 2.x를 사용한 표현식 연산자 수 계산의 사용 예시입니다.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating DynamoDB expression operator counting
and the 300 operator limit");

        try {
            // Example 1: Analyze a simple expression
            System.out.println("\nExample 1: Analyzing a simple expression");
            String simpleExpression = "Price = :price AND Rating > :rating AND
Category IN (:cat1, :cat2, :cat3)";
```

```
            Map<String, Integer> simpleCount =
countOperatorsInExpression(simpleExpression);

            System.out.println("Expression: " + simpleExpression);
            System.out.println("Operator count breakdown:");
            System.out.println("- Comparison operators: " +
simpleCount.get("comparisonOperators"));
            System.out.println("- Logical operators: " +
simpleCount.get("logicalOperators"));
            System.out.println("- Functions: " + simpleCount.get("functions"));
            System.out.println("- Arithmetic operators: " +
simpleCount.get("arithmeticOperators"));
            System.out.println("- Special operators: " +
simpleCount.get("specialOperators"));
            System.out.println("- Total operators: " + simpleCount.get("total"));

            // Example 2: Analyze a complex expression
            System.out.println("\nExample 2: Analyzing a complex expression");
            String complexExpression = "(attribute_exists(Category) AND Size
BETWEEN :min AND :max) OR "
                + "(Price > :price AND contains(Description, :keyword) AND "
                + "(Rating >= :minRating OR Reviews > :minReviews))";
            Map<String, Integer> complexCount =
countOperatorsInExpression(complexExpression);

            System.out.println("Expression: " + complexExpression);
            System.out.println("Operator count breakdown:");
            System.out.println("- Comparison operators: " +
complexCount.get("comparisonOperators"));
            System.out.println("- Logical operators: " +
complexCount.get("logicalOperators"));
            System.out.println("- Functions: " + complexCount.get("functions"));
            System.out.println("- Arithmetic operators: " +
complexCount.get("arithmeticOperators"));
            System.out.println("- Special operators: " +
complexCount.get("specialOperators"));
            System.out.println("- Total operators: " +
complexCount.get("total"));

            // Example 3: Test approaching the operator limit
            System.out.println("\nExample 3: Testing an expression approaching
the operator limit");
            Map<String, Object> approachingLimit =
testOperatorLimit(dynamoDbClient, tableName, key, 290);
```

```
            System.out.println(approachingLimit.get("message"));

            // Example 4: Test exceeding the operator limit
            System.out.println("\nExample 4: Testing an expression exceeding the
operator limit");
            Map<String, Object> exceedingLimit =
testOperatorLimit(dynamoDbClient, tableName, key, 310);
            System.out.println(exceedingLimit.get("message"));

            // Example 5: Breaking down a complex expression
            System.out.println("\nExample 5: Breaking down a complex expression
into multiple operations");
            Map<String, Object> breakdownResult =
breakDownComplexExpression(dynamoDbClient, tableName, key, 500);
            @SuppressWarnings("unchecked")
            Map<Integer, Map<String, Object>> results =
                (Map<Integer, Map<String, Object>>)
breakdownResult.get("results");
            System.out.println(
                "Processed " + results.size() + " of " +
breakdownResult.get("totalBatches") + " batches");

            // Explain the operator counting rules
            System.out.println("\nKey points about DynamoDB expression operator
counting:");
            System.out.println("1. The maximum number of operators in any
expression is 300");
            System.out.println("2. Each comparison operator (=, <>, <, <=, >, >=)
counts as 1 operator");
            System.out.println("3. Each logical operator (AND, OR, NOT) counts as
1 operator");
            System.out.println("4. Each function call (attribute_exists,
contains, etc.) counts as 1 operator");
            System.out.println("5. Each arithmetic operator (+ or -) counts as 1
operator");
            System.out.println("6. BETWEEN counts as 2 operators (BETWEEN itself
and the AND within it)");
            System.out.println("7. IN counts as 1 operator regardless of the
number of values");
            System.out.println("8. Parentheses for grouping and attribute paths
don't count as operators");
            System.out.println("9. When you exceed the limit, the error always
reports '301 operators'");
```

```
            System.out.println("10. For complex operations, break them into
 multiple smaller operations");

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용한 표현식 연산자 수 계산을 보여줍니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Create a complex filter expression with a specified number of conditions.
 *
 * This function demonstrates how to generate a complex expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} conditionsCount - Number of conditions to include
 * @param {boolean} useAnd - Whether to use AND (true) or OR (false) between
 conditions
 * @returns {Object} - Object containing the filter expression and attribute
 values
 */
function createComplexFilterExpression(conditionsCount, useAnd = true) {
  // Initialize the expression parts and attribute values
  const conditions = [];
  const expressionAttributeValues = {};

  // Generate the specified number of conditions
```

```
  for (let i = 0; i < conditionsCount; i++) {
    // Alternate between different comparison operators for variety
    let condition;
    const valueKey = `:val${i}`;

    switch (i % 5) {
      case 0:
        condition = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        condition = `attribute${i} > ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        condition = `attribute${i} < ${valueKey}`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
      case 3:
        condition = `contains(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = `substring${i}`;
        break;
      case 4:
        condition = `attribute_exists(attribute${i})`;
        break;
    }

    conditions.push(condition);
  }

  // Join the conditions with AND or OR
  const operator = useAnd ? " AND " : " OR ";
  const filterExpression = conditions.join(operator);

  // Calculate the operator count
  // Each condition has 1 operator (=, >, <, contains, attribute_exists)
  // Each AND or OR between conditions is 1 operator
  const operatorCount = conditionsCount + (conditionsCount > 0 ? conditionsCount
- 1 : 0);

  return {
    filterExpression,
    expressionAttributeValues,
    operatorCount
```

```
  };
}

/**
 * Create a complex update expression with a specified number of operations.
 *
 * This function demonstrates how to generate a complex update expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} operationsCount - Number of operations to include
 * @returns {Object} - Object containing the update expression and attribute
 values
 */
function createComplexUpdateExpression(operationsCount) {
  // Initialize the expression parts and attribute values
  const setOperations = [];
  const expressionAttributeValues = {};

  // Generate the specified number of SET operations
  for (let i = 0; i < operationsCount; i++) {
    // Alternate between different types of SET operations
    let operation;
    const valueKey = `:val${i}`;

    switch (i % 3) {
      case 0:
        // Simple assignment (1 operator: =)
        operation = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        // Addition (2 operators: = and +)
        operation = `attribute${i} = attribute${i} + ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        // Conditional assignment with if_not_exists (2 operators: = and
 if_not_exists)
        operation = `attribute${i} = if_not_exists(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
    }

    setOperations.push(operation);
```

```
  }

  // Create the update expression
  const updateExpression = `SET ${setOperations.join(", ")}`;

  // Calculate the operator count
  // Each operation has 1-2 operators as noted above
  let operatorCount = 0;
  for (let i = 0; i < operationsCount; i++) {
    operatorCount += (i % 3 === 0) ? 1 : 2;
  }

  return {
    updateExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Test the operator limit by attempting an operation with a complex expression.
 *
 * This function demonstrates what happens when an expression approaches or
 * exceeds the 300 operator limit.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} operatorCount - Target number of operators to include
 * @returns {Promise<Object>} - Result of the operation attempt
 */
async function testOperatorLimit(
  config,
  tableName,
  key,
  operatorCount
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create a complex update expression with the specified operator count
  const { updateExpression, expressionAttributeValues, operatorCount:
 actualCount } =
```

```
    createComplexUpdateExpression(Math.ceil(operatorCount / 1.5)); // Adjust to
 get close to target count

  console.log(`Generated update expression with approximately ${actualCount}
 operators`);

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Attempt the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      message: `Operation succeeded with ${actualCount} operators`,
      data: response
    };
  } catch (error) {
    // Check if the error is due to exceeding the operator limit
    if (error.name === "ValidationException" &&
        error.message.includes("too many operators")) {
      return {
        success: false,
        message: `Operation failed: ${error.message}`,
        operatorCount: actualCount
      };
    }

    // Return other errors
    return {
      success: false,
      message: `Operation failed: ${error.message}`,
      error
    };
  }
}

/**
```

```
 * Break down a complex expression into multiple simpler operations.
 *
 * This function demonstrates how to handle expressions that would exceed
 * the 300 operator limit by breaking them into multiple operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} totalOperations - Total number of operations to perform
 * @returns {Promise<Object>} - Result of the operations
 */
async function breakDownComplexExpression(
  config,
  tableName,
  key,
  totalOperations
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Calculate how many operations we can safely include in each batch
  // Using 150 as a conservative limit (well below 300)
  const operationsPerBatch = 100;
  const batchCount = Math.ceil(totalOperations / operationsPerBatch);

  console.log(`Breaking down ${totalOperations} operations into ${batchCount}
 batches`);

  const results = [];

  // Process each batch
  for (let batch = 0; batch < batchCount; batch++) {
    // Calculate the operations for this batch
    const batchStart = batch * operationsPerBatch;
    const batchEnd = Math.min(batchStart + operationsPerBatch, totalOperations);
    const batchSize = batchEnd - batchStart;

    console.log(`Processing batch ${batch + 1}/${batchCount} with ${batchSize}
 operations`);

    // Create an update expression for this batch
    const { updateExpression, expressionAttributeValues, operatorCount } =
      createComplexUpdateExpression(batchSize);
```

```
    // Define the update parameters
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: updateExpression,
      ExpressionAttributeValues: expressionAttributeValues,
      ReturnValues: "UPDATED_NEW"
    };

    try {
      // Perform the update operation for this batch
      const response = await docClient.send(new UpdateCommand(params));

      results.push({
        batch: batch + 1,
        success: true,
        operatorCount,
        attributes: response.Attributes
      });
    } catch (error) {
      results.push({
        batch: batch + 1,
        success: false,
        operatorCount,
        error: error.message
      });

      // Stop processing if an error occurs
      break;
    }
  }

  return {
    totalBatches: batchCount,
    results
  };
}

/**
 * Count operators in a DynamoDB expression based on the rules in the
 documentation.
 *
 * This function demonstrates how operators are counted according to the
```

```
 * DynamoDB documentation.
 *
 * @param {string} expression - The DynamoDB expression to analyze
 * @returns {Object} - Breakdown of operator counts
 */
function countOperatorsInExpression(expression) {
  // Initialize counters for different operator types
  const counts = {
    comparisonOperators: 0,
    logicalOperators: 0,
    functions: 0,
    arithmeticOperators: 0,
    specialOperators: 0,
    total: 0
  };

  // Count comparison operators (=, <>, <, <=, >, >=)
  const comparisonRegex = /[^<>]=[^=]|<>|<=|>=|[^<]>[^=]|[^>]<[^=]/g;
  const comparisonMatches = expression.match(comparisonRegex) || [];
  counts.comparisonOperators = comparisonMatches.length;

  // Count logical operators (AND, OR, NOT)
  const andMatches = expression.match(/\bAND\b/g) || [];
  const orMatches = expression.match(/\bOR\b/g) || [];
  const notMatches = expression.match(/\bNOT\b/g) || [];
  counts.logicalOperators = andMatches.length + orMatches.length +
 notMatches.length;

  // Count functions (attribute_exists, attribute_not_exists, attribute_type,
 begins_with, contains, size)
  const functionRegex = /\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\(/g;
  const functionMatches = expression.match(functionRegex) || [];
  counts.functions = functionMatches.length;

  // Count arithmetic operators (+ and -)
  const arithmeticMatches = expression.match(/[a-zA-Z0-9_)\]]\s*[\+\-]\s*[a-zA-
Z0-9_(:]/g) || [];
  counts.arithmeticOperators = arithmeticMatches.length;

  // Count special operators (BETWEEN, IN)
  const betweenMatches = expression.match(/\bBETWEEN\b/g) || [];
  const inMatches = expression.match(/\bIN\b/g) || [];
  counts.specialOperators = betweenMatches.length + inMatches.length;
```

```
  // Add extra operators for BETWEEN (each BETWEEN includes an AND)
  counts.logicalOperators += betweenMatches.length;

  // Calculate total
  counts.total = counts.comparisonOperators +
                 counts.logicalOperators +
                 counts.functions +
                 counts.arithmeticOperators +
                 counts.specialOperators;

  return counts;
}
```

AWS SDK for JavaScript를 사용한 표현식 연산자 수 계산의 사용 예시입니다.

```
/**
 * Example of how to work with expression operator counting.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating DynamoDB expression operator counting and the 300
 operator limit");

  try {
    // Example 1: Analyze a simple expression
    console.log("\nExample 1: Analyzing a simple expression");
    const simpleExpression = "Price = :price AND Rating > :rating AND Category IN
 (:cat1, :cat2, :cat3)";
    const simpleCount = countOperatorsInExpression(simpleExpression);

    console.log(`Expression: ${simpleExpression}`);
    console.log("Operator count breakdown:");
    console.log(`- Comparison operators: ${simpleCount.comparisonOperators}`);
    console.log(`- Logical operators: ${simpleCount.logicalOperators}`);
    console.log(`- Functions: ${simpleCount.functions}`);
    console.log(`- Arithmetic operators: ${simpleCount.arithmeticOperators}`);
    console.log(`- Special operators: ${simpleCount.specialOperators}`);
```

```
    console.log(`- Total operators: ${simpleCount.total}`);

    // Example 2: Analyze a complex expression
    console.log("\nExample 2: Analyzing a complex expression");
    const complexExpression =
      "(attribute_exists(Category) AND Size BETWEEN :min AND :max) OR " +
      "(Price > :price AND contains(Description, :keyword) AND " +
      "(Rating >= :minRating OR Reviews > :minReviews))";
    const complexCount = countOperatorsInExpression(complexExpression);

    console.log(`Expression: ${complexExpression}`);
    console.log("Operator count breakdown:");
    console.log(`- Comparison operators: ${complexCount.comparisonOperators}`);
    console.log(`- Logical operators: ${complexCount.logicalOperators}`);
    console.log(`- Functions: ${complexCount.functions}`);
    console.log(`- Arithmetic operators: ${complexCount.arithmeticOperators}`);
    console.log(`- Special operators: ${complexCount.specialOperators}`);
    console.log(`- Total operators: ${complexCount.total}`);

    // Example 3: Test approaching the operator limit
    console.log("\nExample 3: Testing an expression approaching the operator
limit");
    const approachingLimit = await testOperatorLimit(config, tableName, key,
290);
    console.log(approachingLimit.message);

    // Example 4: Test exceeding the operator limit
    console.log("\nExample 4: Testing an expression exceeding the operator
limit");
    const exceedingLimit = await testOperatorLimit(config, tableName, key, 310);
    console.log(exceedingLimit.message);

    // Example 5: Breaking down a complex expression
    console.log("\nExample 5: Breaking down a complex expression into multiple
operations");
    const breakdownResult = await breakDownComplexExpression(config, tableName,
key, 500);
    console.log(`Processed ${breakdownResult.results.length} of
${breakdownResult.totalBatches} batches`);

    // Explain the operator counting rules
    console.log("\nKey points about DynamoDB expression operator counting:");
    console.log("1. The maximum number of operators in any expression is 300");
```

```
    console.log("2. Each comparison operator (=, <>, <, <=, >, >=) counts as 1
 operator");
    console.log("3. Each logical operator (AND, OR, NOT) counts as 1 operator");
    console.log("4. Each function call (attribute_exists, contains, etc.) counts
 as 1 operator");
    console.log("5. Each arithmetic operator (+ or -) counts as 1 operator");
    console.log("6. BETWEEN counts as 2 operators (BETWEEN itself and the AND
 within it)");
    console.log("7. IN counts as 1 operator regardless of the number of values");
    console.log("8. Parentheses for grouping and attribute paths don't count as
 operators");
    console.log("9. When you exceed the limit, the error always reports '301
 operators'");
    console.log("10. For complex operations, break them into multiple smaller
 operations");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용한 표현식 연산자 수 계산을 보여줍니다.

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, List, Optional, Tuple



def create_complex_filter_expression(
    attribute_name: str, values: List[Any], use_or: bool = True
) -> Tuple[str, Dict[str, Any], Dict[str, str], int]:
    """
    Create a complex filter expression with multiple conditions.

    This function demonstrates how to build a complex filter expression
```

```
    and count the number of operators used.

    Args:
        attribute_name (str): The name of the attribute to filter on.
        values (List[Any]): List of values to compare against.
        use_or (bool, optional): Whether to use OR between conditions. Defaults
to True.

    Returns:
        Tuple[str, Dict[str, Any], Dict[str, str], int]: A tuple containing:
            - The filter expression string
            - Expression attribute values
            - Expression attribute names
            - The number of operators used
    """
    if not values:
        return "", {}, {}, 0

    # Initialize expression components
    filter_expression = ""
    expression_attribute_values = {}
    expression_attribute_names = {"#attr": attribute_name}
    operator_count = 0

    # Build the filter expression
    for i, value in enumerate(values):
        value_placeholder = f":val{i}"
        expression_attribute_values[value_placeholder] = value

        if i > 0:
            # Add OR or AND operator between conditions
            filter_expression += " OR " if use_or else " AND "
            operator_count += 1  # Count the OR/AND operator

        # Add the condition
        filter_expression += f"#attr = {value_placeholder}"
        operator_count += 1  # Count the = operator

    return (
        filter_expression,
        expression_attribute_values,
        expression_attribute_names,
        operator_count,
    )
```

```
def create_nested_filter_expression(
    depth: int, conditions_per_level: int
) -> Tuple[str, Dict[str, Any], Dict[str, str], int]:
    """
    Create a deeply nested filter expression with multiple conditions.

    This function demonstrates how to build a complex nested filter expression
    and count the number of operators used.

    Args:
        depth (int): The depth of nesting.
        conditions_per_level (int): Number of conditions at each level.

    Returns:
        Tuple[str, Dict[str, Any], Dict[str, str], int]: A tuple containing:
            - The filter expression string
            - Expression attribute values
            - Expression attribute names
            - The number of operators used
    """
    if depth <= 0 or conditions_per_level <= 0:
        return "", {}, {}, 0

    # Initialize expression components
    expression_attribute_values = {}
    expression_attribute_names = {}
    operator_count = 0

    def build_nested_expression(current_depth: int, prefix: str) -> str:
        nonlocal operator_count

        if current_depth <= 0:
            return ""

        # Build conditions at this level
        conditions = []
        for i in range(conditions_per_level):
            attr_name = f"attr{prefix}_{i}"
            attr_placeholder = f"#attr{prefix}_{i}"
            val_placeholder = f":val{prefix}_{i}"

            expression_attribute_names[attr_placeholder] = attr_name
```

```
            expression_attribute_values[val_placeholder] = i

            conditions.append(f"{attr_placeholder} = {val_placeholder}")
            operator_count += 1  # Count the = operator

        # Join conditions with AND
        level_expression = " AND ".join(conditions)
        operator_count += max(0, len(conditions) - 1)  # Count the AND operators

        # If not at the deepest level, add nested expressions
        if current_depth > 1:
            nested_expr = build_nested_expression(current_depth - 1,
 f"{prefix}_{current_depth}")
            if nested_expr:
                level_expression = f"({level_expression}) OR ({nested_expr})"
                operator_count += 1  # Count the OR operator

        return level_expression

    # Build the expression starting from the top level
    filter_expression = build_nested_expression(depth, "1")

    return (
        filter_expression,
        expression_attribute_values,
        expression_attribute_names,
        operator_count,
    )


def count_operators_in_update_expression(update_expression: str) -> int:
    """
    Count the number of operators in an update expression.

    This function demonstrates how to count operators in an update expression
    based on DynamoDB's rules.

    Args:
        update_expression (str): The update expression to analyze.

    Returns:
        int: The number of operators in the expression.
    """
    operator_count = 0
```

```
    # Count SET operations
    if "SET" in update_expression:
        set_section = (
            update_expression.split("SET")[1].split("REMOVE")[0].split("ADD")
[0].split("DELETE")[0]
        )

        # Count assignment operators (=)
        operator_count += set_section.count("=")

        # Count arithmetic operators (+, -)
        operator_count += set_section.count("+")
        operator_count += set_section.count("-")

        # Count list_append function calls (each counts as 1 operator)
        operator_count += set_section.lower().count("list_append")

        # Count if_not_exists function calls (each counts as 1 operator)
        operator_count += set_section.lower().count("if_not_exists")

    # Count REMOVE operations (no additional operators)

    # Count ADD operations (each ADD counts as 1 operator)
    if "ADD" in update_expression:
        add_section = (
            update_expression.split("ADD")[1].split("DELETE")[0].split("SET")
[0].split("REMOVE")[0]
        )
        operator_count += add_section.count(",") + 1

    # Count DELETE operations (each DELETE counts as 1 operator)
    if "DELETE" in update_expression:
        delete_section = (
            update_expression.split("DELETE")[1].split("SET")[0].split("ADD")
[0].split("REMOVE")[0]
        )
        operator_count += delete_section.count(",") + 1

    return operator_count


def count_operators_in_condition_expression(condition_expression: str) -> int:
    """
```

```
    Count the number of operators in a condition expression.

    This function demonstrates how to count operators in a condition expression
    based on DynamoDB's rules.

    Args:
        condition_expression (str): The condition expression to analyze.

    Returns:
        int: The number of operators in the expression.
    """
    operator_count = 0

    # Count comparison operators
    comparison_operators = ["=", "<>", "<", "<=", ">", ">="]
    for op in comparison_operators:
        operator_count += condition_expression.count(op)

    # Count logical operators
    operator_count += condition_expression.upper().count(" AND ")
    operator_count += condition_expression.upper().count(" OR ")
    operator_count += condition_expression.upper().count("NOT ")

    # Count BETWEEN operator (counts as 2: BETWEEN + AND)
    between_count = condition_expression.upper().count(" BETWEEN ")
    operator_count += between_count * 2

    # Count IN operator (counts as 1 regardless of number of values)
    operator_count += condition_expression.upper().count(" IN ")

    # Count functions (each counts as 1 operator)
    functions = [
        "attribute_exists",
        "attribute_not_exists",
        "attribute_type",
        "begins_with",
        "contains",
        "size",
    ]
    for func in functions:
        operator_count += condition_expression.lower().count(func)

    return operator_count
```

```
# Note: This function is for demonstration purposes only and should be called
 from example_usage()
# It's not meant to be used directly as a test function
def _test_expression_limit(
    table_name: str, key: Dict[str, Any], operator_count: int, attribute_name:
 str = "TestAttribute"
) -> Tuple[bool, Optional[str]]:
    """
    Test if an expression with a specific number of operators exceeds the limit.

    This function demonstrates how to test the 300 operator limit by creating
    an expression with a specified number of operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        operator_count (int): The number of operators to include in the
 expression.
        attribute_name (str, optional): The name of the attribute to update.
 Defaults to "TestAttribute".

    Returns:
        Tuple[bool, Optional[str]]: A tuple containing:
            - A boolean indicating if the operation succeeded
            - The error message if it failed, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create an update expression with the specified number of operators
    update_expression = f"SET #{attribute_name} = :val0"
    expression_attribute_names = {f"#{attribute_name}": attribute_name}
    expression_attribute_values = {":val0": 0}

    # Add additional SET operations to reach the desired operator count
    # Each assignment adds 1 operator
    for i in range(1, operator_count):
        attr_name = f"{attribute_name}{i}"
        attr_placeholder = f"#attr{i}"
        val_placeholder = f":val{i}"

        update_expression += f", {attr_placeholder} = {val_placeholder}"
```

```
        expression_attribute_names[attr_placeholder] = attr_name
        expression_attribute_values[val_placeholder] = i

    try:
        # Attempt the update operation
        table.update_item(
            Key=key,
            UpdateExpression=update_expression,
            ExpressionAttributeNames=expression_attribute_names,
            ExpressionAttributeValues=expression_attribute_values,
        )
        return True, None
    except ClientError as e:
        error_message = e.response["Error"]["Message"]

        if "expression contains too many operators" in error_message.lower():
            return False, error_message
        else:
            # Other error occurred
            raise
```

AWS SDK for Python (Boto3)를 사용한 표현식 연산자 수 계산의 사용 예시입니다.

```
def example_usage():
    """Example of how to use the expression operator counting functions."""

    print("Example 1: Creating a complex filter expression with multiple
 conditions")
    attribute_name = "Status"
    values = ["Active", "Pending", "Processing", "Shipped", "Delivered"]

    filter_expr, expr_attr_vals, expr_attr_names, op_count =
 create_complex_filter_expression(
        attribute_name=attribute_name, values=values, use_or=True
    )

    print(f"Filter Expression: {filter_expr}")
    print(f"Expression Attribute Values: {expr_attr_vals}")
    print(f"Expression Attribute Names: {expr_attr_names}")
    print(f"Operator Count: {op_count}")
```

```
   print("\nExample 2: Creating a nested filter expression")
   nested_expr, nested_vals, nested_names, nested_count =
create_nested_filter_expression(
       depth=3, conditions_per_level=2
   )

   print(f"Nested Filter Expression: {nested_expr}")
   print(f"Operator Count: {nested_count}")

   print("\nExample 3: Counting operators in an update expression")
   update_expression = "SET #name = :name, #age = :age + :increment,
#address.#city = :city, #status = if_not_exists(#status, :default_status) REMOVE
#old_field ADD #counter :value DELETE #set_attr :set_val"
   update_op_count = count_operators_in_update_expression(update_expression)

   print(f"Update Expression: {update_expression}")
   print(f"Operator Count: {update_op_count}")

   print("\nExample 4: Counting operators in a condition expression")
   condition_expression = "(#status = :active OR #status = :pending) AND #price
BETWEEN :min_price AND :max_price AND attribute_exists(#category) AND NOT
(#stock <= :min_stock)"
   condition_op_count =
count_operators_in_condition_expression(condition_expression)

   print(f"Condition Expression: {condition_expression}")
   print(f"Operator Count: {condition_op_count}")

   print("\nExample 5: Testing the 300 operator limit")

   # This is just for demonstration - in a real application, you would use your
actual table
   # Note: This function is renamed to _test_expression_limit to avoid pytest
trying to run it
   print("In a real application, you would test with _test_expression_limit
function")
   print("Expression with 250 operators would be under the limit")
   print("Expression with 350 operators would exceed the 300 operator limit")

   print("\nOperator Counting Rules in DynamoDB:")
   print("1. Comparison Operators (=, <>, <, <=, >, >=): 1 operator each")
   print("2. Logical Operators (AND, OR, NOT): 1 operator each")
   print("3. BETWEEN: 2 operators (BETWEEN + AND)")
```

```
    print("4. IN: 1 operator (regardless of number of values)")
    print("5. Functions (attribute_exists, begins_with, etc.): 1 operator each")
    print("6. Arithmetic Operators (+, -): 1 operator each")
    print("7. SET assignments (=): 1 operator each")
    print("8. ADD and DELETE operations: 1 operator each")

    print("\nStrategies for Working Within the 300 Operator Limit:")
    print("1. Break operations into multiple requests")
    print("2. Use DynamoDB Transactions for complex operations")
    print("3. Optimize data model to reduce query complexity")
    print("4. Use application-side filtering for less critical filters")
    print("5. Consider using IN operator instead of multiple OR conditions")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## COVID-19 데이터를 추적하는 API Gateway REST API 생성

다음 코드 예제에서는 가상 데이터를 사용하여 미국의 일별 COVID-19 발생 현황을 추적하는 시스템을 시뮬레이션하는 REST API를 생성하는 방법을 보여줍니다.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)와 함께 AWS Chalice를 사용하여 Amazon API Gateway, AWS Lambda 및 Amazon DynamoDB를 사용한 서버리스 REST API를 생성하는 방법을 보여줍니다. REST API로 가상 데이터를 사용하여 미국의 일별 COVID-19 발생 현황을 추적하는 시스템을 시뮬레이션합니다. 다음 작업을 수행하는 방법에 대해 알아보세요.

- AWS Chalice를 사용하여 API Gateway를 통해 들어오는 REST 요청을 처리하도록 호출되는 Lambda 함수의 경로를 정의합니다.
- Lambda 함수로 데이터를 검색하고 DynamoDB 테이블에 저장하여 REST 요청을 처리합니다.
- AWS CloudFormation 템플릿에 테이블 구조 및 보안 역할 리소스를 정의합니다.
- AWS Chalice 및 CloudFormation을 사용하여 필요한 모든 리소스를 패키징하고 배포합니다.

- CloudFormation을 사용하여 생성된 모든 리소스를 정리합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- AWS CloudFormation
- DynamoDB
- Lambda

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## Step Functions를 사용하여 메신저 애플리케이션 생성

다음 코드 예제에서는 데이터베이스 테이블에서 메시지 레코드를 검색하는 AWS Step Functions 메신저 애플리케이션을 생성하는 방법을 보여줍니다.

Python

SDK for Python(Boto3)

AWS Step Functions를 AWS SDK for Python (Boto3)와 함께 사용하여 Amazon DynamoDB 테이블에서 메시지 레코드를 검색하고 Amazon Simple Queue Service(Amazon SQS)를 통해 전송하는 메신저 애플리케이션을 생성하는 방법을 보여줍니다. 상태 머신은 AWS Lambda 함수와 통합되어 데이터베이스에서 전송되지 않은 메시지를 스캔합니다.

- Amazon DynamoDB 테이블에서 메시지 레코드를 검색하고 업데이트하는 상태 머신을 생성합니다.
- 상태 머신 정의를 업데이트하여 메시지를 Amazon Simple Queue Service(Amazon SQS)에도 전송합니다.
- 상태 머신의 실행을 시작하고 중지합니다.
- 서비스 통합을 사용하여 상태 머신에서 Lambda, DynamoDB 및 Amazon SQS에 연결합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB

- Lambda
- Amazon SQS
- Step Functions

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## 사용자가 레이블을 사용하여 사진을 관리할 수 있는 사진 자산 관리 애플리케이션 만들기

다음 코드 예제는 사용자가 레이블을 사용하여 사진을 관리할 수 있는 서버리스 애플리케이션을 생성하는 방법을 보여 줍니다.

.NET

SDK for .NET

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK for C++

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK for Java 2.x

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK for Kotlin

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK for PHP

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK for Rust

Amazon Rekognition을 사용하여 이미지에서 레이블을 감지하고 나중에 검색할 수 있도록 저장하는 사진 자산 관리 애플리케이션을 개발하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제의 출처에 대한 자세한 내용은 AWS 커뮤니티의 게시물을 참조하세요.

이 예시에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

# AWS SDK를 사용하여 글로벌 보조 인덱스가 포함된 DynamoDB 테이블 만들기

다음 코드 예제는 글로벌 보조 인덱스로 테이블을 만드는 방법을 보여줍니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용하여 글로벌 보조 인덱스가 있는 DynamoDB 테이블을 만듭니다.

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.ProjectionType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    public void createTable() {
        try {
            // Attribute definitions
```

```
            final List<AttributeDefinition> attributeDefinitions = new
ArrayList<>();
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(ISSUE_ID_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(TITLE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(CREATE_DATE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(DUE_DATE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());

            // Key schema for table
            final List<KeySchemaElement> tableKeySchema = new ArrayList<>();
            tableKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.HASH)
                .build()); // Partition key
            tableKeySchema.add(KeySchemaElement.builder()
                .attributeName(TITLE_ATTR)
                .keyType(KeyType.RANGE)
                .build()); // Sort key

            // Initial provisioned throughput settings for the indexes
            final ProvisionedThroughput ptIndex = ProvisionedThroughput.builder()
                .readCapacityUnits(1L)
                .writeCapacityUnits(1L)
                .build();

            // CreateDateIndex
            final List<KeySchemaElement> createDateKeySchema = new ArrayList<>();
            createDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(CREATE_DATE_ATTR)
                .keyType(KeyType.HASH)
                .build());
            createDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
```

```
                .keyType(KeyType.RANGE)
                .build());

            final Projection createDateProjection = Projection.builder()
                .projectionType(ProjectionType.INCLUDE)
                .nonKeyAttributes(DESCRIPTION_ATTR, STATUS_ATTR)
                .build();

            final GlobalSecondaryIndex createDateIndex =
GlobalSecondaryIndex.builder()
                .indexName(CREATE_DATE_INDEX)
                .keySchema(createDateKeySchema)
                .projection(createDateProjection)
                .provisionedThroughput(ptIndex)
                .build();

            // TitleIndex
            final List<KeySchemaElement> titleKeySchema = new ArrayList<>();
            titleKeySchema.add(KeySchemaElement.builder()
                .attributeName(TITLE_ATTR)
                .keyType(KeyType.HASH)
                .build());
            titleKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.RANGE)
                .build());

            final Projection titleProjection =

Projection.builder().projectionType(ProjectionType.KEYS_ONLY).build();

            final GlobalSecondaryIndex titleIndex =
GlobalSecondaryIndex.builder()
                .indexName(TITLE_INDEX)
                .keySchema(titleKeySchema)
                .projection(titleProjection)
                .provisionedThroughput(ptIndex)
                .build();

            // DueDateIndex
            final List<KeySchemaElement> dueDateKeySchema = new ArrayList<>();
            dueDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(DUE_DATE_ATTR)
                .keyType(KeyType.HASH)
```

```
                .build());

            final Projection dueDateProjection =
                Projection.builder().projectionType(ProjectionType.ALL).build();

            final GlobalSecondaryIndex dueDateIndex =
GlobalSecondaryIndex.builder()
                .indexName(DUE_DATE_INDEX)
                .keySchema(dueDateKeySchema)
                .projection(dueDateProjection)
                .provisionedThroughput(ptIndex)
                .build();

            final CreateTableRequest createTableRequest =
CreateTableRequest.builder()
                .tableName(TABLE_NAME)
                .keySchema(tableKeySchema)
                .attributeDefinitions(attributeDefinitions)
                .globalSecondaryIndexes(createDateIndex, titleIndex,
dueDateIndex)
                .provisionedThroughput(ProvisionedThroughput.builder()
                    .readCapacityUnits(1L)
                    .writeCapacityUnits(1L)
                    .build())
                .build();

            System.out.println("Creating table " + TABLE_NAME + "...");
            dynamoDbClient.createTable(createTableRequest);

            // Wait for table to become active
            System.out.println("Waiting for " + TABLE_NAME + " to become
ACTIVE...");
            final DynamoDbWaiter waiter = dynamoDbClient.waiter();
            final DescribeTableRequest describeTableRequest =
                DescribeTableRequest.builder().tableName(TABLE_NAME).build();

            final WaiterResponse<DescribeTableResponse> waiterResponse =
                waiter.waitUntilTableExists(describeTableRequest);
            waiterResponse.matched().response().ifPresent(response ->
System.out.println("Table is now ready for use"));

        } catch (DynamoDbException e) {
            System.err.println("Error creating table: " + e.getMessage());
            e.printStackTrace();
```

```
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateTable을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 웜 처리량 설정으로 DynamoDB 테이블 만들기

다음 코드 예제에서는 웜 처리량이 활성화된 테이블을 만드는 방법을 보여줍니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용하여 웜 처리량 설정이 있는 DynamoDB 테이블을 만듭니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

    public static WarmThroughput buildWarmThroughput(final Long
 readUnitsPerSecond, final Long writeUnitsPerSecond) {
        return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
    }

    /**
     * Builds a ProvisionedThroughput object with the specified read and write
 capacity units.
     *
```

```
     * @param readCapacityUnits The read capacity units
     * @param writeCapacityUnits The write capacity units
     * @return A configured ProvisionedThroughput object
     */
    public static ProvisionedThroughput buildProvisionedThroughput(
        final Long readCapacityUnits, final Long writeCapacityUnits) {
        return ProvisionedThroughput.builder()
            .readCapacityUnits(readCapacityUnits)
            .writeCapacityUnits(writeCapacityUnits)
            .build();
    }

    /**
     * Builds an AttributeDefinition with the specified name and type.
     *
     * @param attributeName The attribute name
     * @param scalarAttributeType The attribute type
     * @return A configured AttributeDefinition
     */
    private static AttributeDefinition buildAttributeDefinition(
        final String attributeName, final ScalarAttributeType
scalarAttributeType) {
        return AttributeDefinition.builder()
            .attributeName(attributeName)
            .attributeType(scalarAttributeType)
            .build();
    }

    /**
     * Builds a KeySchemaElement with the specified name and key type.
     *
     * @param attributeName The attribute name
     * @param keyType The key type (HASH or RANGE)
     * @return A configured KeySchemaElement
     */
    private static KeySchemaElement buildKeySchemaElement(final String
attributeName, final KeyType keyType) {
        return KeySchemaElement.builder()
            .attributeName(attributeName)
            .keyType(keyType)
            .build();
    }

    /**
```

```
    * Creates a DynamoDB table with the specified configuration including warm
throughput settings.
    *
    * @param ddb The DynamoDB client
    * @param tableName The name of the table to create
    * @param partitionKey The partition key attribute name
    * @param sortKey The sort key attribute name
    * @param miscellaneousKeyAttribute Additional key attribute name for GSI
    * @param nonKeyAttribute Non-key attribute to include in GSI projection
    * @param tableReadCapacityUnits Read capacity units for the table
    * @param tableWriteCapacityUnits Write capacity units for the table
    * @param tableWarmReadUnitsPerSecond Warm read units per second for the
table
    * @param tableWarmWriteUnitsPerSecond Warm write units per second for the
table
    * @param globalSecondaryIndexName The name of the GSI to create
    * @param globalSecondaryIndexReadCapacityUnits Read capacity units for the
GSI
    * @param globalSecondaryIndexWriteCapacityUnits Write capacity units for the
GSI
    * @param globalSecondaryIndexWarmReadUnitsPerSecond Warm read units per
second for the GSI
    * @param globalSecondaryIndexWarmWriteUnitsPerSecond Warm write units per
second for the GSI
    */
   public static void createDynamoDBTable(
       final DynamoDbClient ddb,
       final String tableName,
       final String partitionKey,
       final String sortKey,
       final String miscellaneousKeyAttribute,
       final String nonKeyAttribute,
       final Long tableReadCapacityUnits,
       final Long tableWriteCapacityUnits,
       final Long tableWarmReadUnitsPerSecond,
       final Long tableWarmWriteUnitsPerSecond,
       final String globalSecondaryIndexName,
       final Long globalSecondaryIndexReadCapacityUnits,
       final Long globalSecondaryIndexWriteCapacityUnits,
       final Long globalSecondaryIndexWarmReadUnitsPerSecond,
       final Long globalSecondaryIndexWarmWriteUnitsPerSecond) {

       // Define the table attributes
```

```
        final AttributeDefinition partitionKeyAttribute =
buildAttributeDefinition(partitionKey, ScalarAttributeType.S);
        final AttributeDefinition sortKeyAttribute =
buildAttributeDefinition(sortKey, ScalarAttributeType.S);
        final AttributeDefinition miscellaneousKeyAttributeDefinition =
            buildAttributeDefinition(miscellaneousKeyAttribute,
ScalarAttributeType.N);
        final AttributeDefinition[] attributeDefinitions = {
            partitionKeyAttribute, sortKeyAttribute,
miscellaneousKeyAttributeDefinition
        };

        // Define the table key schema
        final KeySchemaElement partitionKeyElement =
buildKeySchemaElement(partitionKey, KeyType.HASH);
        final KeySchemaElement sortKeyElement = buildKeySchemaElement(sortKey,
KeyType.RANGE);
        final KeySchemaElement[] keySchema = {partitionKeyElement,
sortKeyElement};

        // Define the provisioned throughput for the table
        final ProvisionedThroughput provisionedThroughput =
            buildProvisionedThroughput(tableReadCapacityUnits,
tableWriteCapacityUnits);

        // Define the Global Secondary Index (GSI)
        final KeySchemaElement globalSecondaryIndexPartitionKeyElement =
buildKeySchemaElement(sortKey, KeyType.HASH);
        final KeySchemaElement globalSecondaryIndexSortKeyElement =
            buildKeySchemaElement(miscellaneousKeyAttribute, KeyType.RANGE);
        final KeySchemaElement[] gsiKeySchema = {
            globalSecondaryIndexPartitionKeyElement,
globalSecondaryIndexSortKeyElement
        };

        final Projection gsiProjection = Projection.builder()
            .projectionType(PROJECTION_TYPE_INCLUDE)
            .nonKeyAttributes(nonKeyAttribute)
            .build();

        final ProvisionedThroughput gsiProvisionedThroughput =
            buildProvisionedThroughput(globalSecondaryIndexReadCapacityUnits,
globalSecondaryIndexWriteCapacityUnits);
```

```
        // Define the warm throughput for the Global Secondary Index (GSI)
        final WarmThroughput gsiWarmThroughput = buildWarmThroughput(
            globalSecondaryIndexWarmReadUnitsPerSecond,
globalSecondaryIndexWarmWriteUnitsPerSecond);

        final GlobalSecondaryIndex globalSecondaryIndex =
GlobalSecondaryIndex.builder()
            .indexName(globalSecondaryIndexName)
            .keySchema(gsiKeySchema)
            .projection(gsiProjection)
            .provisionedThroughput(gsiProvisionedThroughput)
            .warmThroughput(gsiWarmThroughput)
            .build();

        // Define the warm throughput for the table
        final WarmThroughput tableWarmThroughput =
            buildWarmThroughput(tableWarmReadUnitsPerSecond,
tableWarmWriteUnitsPerSecond);

        final CreateTableRequest request = CreateTableRequest.builder()
            .tableName(tableName)
            .attributeDefinitions(attributeDefinitions)
            .keySchema(keySchema)
            .provisionedThroughput(provisionedThroughput)
            .globalSecondaryIndexes(globalSecondaryIndex)
            .warmThroughput(tableWarmThroughput)
            .build();

        final CreateTableResponse response = ddb.createTable(request);
        System.out.println(response);
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateTable을 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용하여 웜 처리량 설정이 있는 DynamoDB 테이블을 만듭니다.

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";
```

```
export async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
          ReadCapacityUnits: tableProvisionedReadUnits,
          WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
          ReadUnitsPerSecond: tableWarmReads,
          WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
          {
            IndexName: indexName,
            KeySchema: [
                { AttributeName: sortKey, KeyType: "HASH" },
                { AttributeName: miscKeyAttr, KeyType: "RANGE" },
```

```
            ],
            Projection: {
                ProjectionType: "INCLUDE",
                NonKeyAttributes: [nonKeyAttr],
            },
            ProvisionedThroughput: {
                ReadCapacityUnits: indexProvisionedReadUnits,
                WriteCapacityUnits: indexProvisionedWriteUnits,
            },
            WarmThroughput: {
                ReadUnitsPerSecond: indexWarmReads,
                WriteUnitsPerSecond: indexWarmWrites,
            },
          },
      ],
    });
    const response = await ddbClient.send(command);
    console.log(response);
    return response;
  } catch (error) {
    console.error(`Error creating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
createDynamoDBTableWithWarmThroughput(
  'example-table',
  'pk',
  'sk',
  'gsiKey',
  'data',
  10, 10, 5, 5,
  'example-index',
  5, 5, 2, 2
);
*/
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 CreateTable을 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용하여 웜 처리량 설정이 있는 DynamoDB 테이블을 만듭니다.

```
from boto3 import client
from botocore.exceptions import ClientError


def create_dynamodb_table_warm_throughput(
    table_name,
    partition_key,
    sort_key,
    misc_key_attr,
    non_key_attr,
    table_provisioned_read_units,
    table_provisioned_write_units,
    table_warm_reads,
    table_warm_writes,
    gsi_name,
    gsi_provisioned_read_units,
    gsi_provisioned_write_units,
    gsi_warm_reads,
    gsi_warm_writes,
    region_name="us-east-1",
):
    """
    Creates a DynamoDB table with a warm throughput setting configured.

    :param table_name: The name of the table to be created.
    :param partition_key: The partition key for the table being created.
    :param sort_key: The sort key for the table being created.
    :param misc_key_attr: A miscellaneous key attribute for the table being
 created.
    :param non_key_attr: A non-key attribute for the table being created.
    :param table_provisioned_read_units: The newly created table's provisioned
 read capacity units.
    :param table_provisioned_write_units: The newly created table's provisioned
 write capacity units.
    :param table_warm_reads: The read units per second setting for the table's
 warm throughput.
```

```
   :param table_warm_writes: The write units per second setting for the table's
warm throughput.
   :param gsi_name: The name of the Global Secondary Index (GSI) to be created
on the table.
   :param gsi_provisioned_read_units: The configured Global Secondary Index
(GSI) provisioned read capacity units.
   :param gsi_provisioned_write_units: The configured Global Secondary Index
(GSI) provisioned write capacity units.
   :param gsi_warm_reads: The read units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
   :param gsi_warm_writes: The write units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
   :param region_name: The AWS Region name to target. defaults to us-east-1
   """
   try:
       ddb = client("dynamodb", region_name=region_name)

       # Define the table attributes
       attribute_definitions = [
           {"AttributeName": partition_key, "AttributeType": "S"},
           {"AttributeName": sort_key, "AttributeType": "S"},
           {"AttributeName": misc_key_attr, "AttributeType": "N"},
       ]

       # Define the table key schema
       key_schema = [
           {"AttributeName": partition_key, "KeyType": "HASH"},
           {"AttributeName": sort_key, "KeyType": "RANGE"},
       ]

       # Define the provisioned throughput for the table
       provisioned_throughput = {
           "ReadCapacityUnits": table_provisioned_read_units,
           "WriteCapacityUnits": table_provisioned_write_units,
       }

       # Define the global secondary index
       gsi_key_schema = [
           {"AttributeName": sort_key, "KeyType": "HASH"},
           {"AttributeName": misc_key_attr, "KeyType": "RANGE"},
       ]
       gsi_projection = {"ProjectionType": "INCLUDE", "NonKeyAttributes":
[non_key_attr]}
       gsi_provisioned_throughput = {
```

```
            "ReadCapacityUnits": gsi_provisioned_read_units,
            "WriteCapacityUnits": gsi_provisioned_write_units,
        }
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_warm_reads,
            "WriteUnitsPerSecond": gsi_warm_writes,
        }
        global_secondary_indexes = [
            {
                "IndexName": gsi_name,
                "KeySchema": gsi_key_schema,
                "Projection": gsi_projection,
                "ProvisionedThroughput": gsi_provisioned_throughput,
                "WarmThroughput": gsi_warm_throughput,
            }
        ]

        # Define the warm throughput for the table
        warm_throughput = {
            "ReadUnitsPerSecond": table_warm_reads,
            "WriteUnitsPerSecond": table_warm_writes,
        }

        # Create the DynamoDB client and create the table
        response = ddb.create_table(
            TableName=table_name,
            AttributeDefinitions=attribute_definitions,
            KeySchema=key_schema,
            ProvisionedThroughput=provisioned_throughput,
            GlobalSecondaryIndexes=global_secondary_indexes,
            WarmThroughput=warm_throughput,
        )

        print(response)
        return response
    except ClientError as e:
        print(f"Error creating table: {e}")
        raise e
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 CreateTable를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## DynamoDB 데이터를 추적하는 웹 애플리케이션 만들기

다음 코드 예제는 Amazon DynamoDB 테이블에서 작업 항목을 추적하고 Amazon Simple Email Service(Amazon SES)를 사용하여 보고서를 보내는 웹 애플리케이션을 만드는 방법을 보여 줍니다.

.NET

SDK for .NET

Amazon DynamoDB .NET API를 사용하여 DynamoDB 작업 데이터를 추적하는 동적 웹 애플리케이션을 만드는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon SES

Java

SDK for Java 2.x

Amazon DynamoDB API를 사용하여 DynamoDB 작업 데이터를 추적하는 동적 웹 애플리케이션을 만드는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon SES

Kotlin

SDK for Kotlin

Amazon DynamoDB API를 사용하여 DynamoDB 작업 데이터를 추적하는 동적 웹 애플리케이션을 만드는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon SES

Python

SDK for Python (Boto3)

AWS SDK for Python (Boto3)을 사용하여 Amazon DynamoDB에서 작업 항목을 추적하고 Amazon Simple Email Service(Amazon SES)를 통해 보고서를 이메일로 보내는 REST 서비스를 만드는 방법을 보여 줍니다. 이 예제는 Flask 웹 프레임워크를 사용하여 HTTP 라우팅을 처리하고 React 웹 페이지와 통합하여 완전한 기능을 갖춘 웹 애플리케이션을 제공합니다.

- AWS 서비스와 통합되는 Flask REST 서비스를 구축합니다.
- DynamoDB 테이블에 저장된 작업 항목을 읽고, 쓰고, 업데이트합니다.
- Amazon SES를 사용하여 작업 항목에 대한 이메일 보고서를 보냅니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub의 AWS 코드 예제 리포지토리에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon SES

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## API Gateway를 사용하여 WebSocket 채팅 애플리케이션 생성

다음 코드 예제에서는 Amazon API Gateway 기반의 WebSocket API에서 제공되는 채팅 애플리케이션을 생성하는 방법을 보여줍니다.

Python

SDK for Python(Boto3)

Amazon API Gateway V2와 함께 AWS SDK for Python (Boto3)를 사용하여 AWS Lambda 및 Amazon DynamoDB와 통합되는 WebSocket API를 생성하는 방법을 보여줍니다.

- API Gateway에서 제공되는 WebSocket API를 생성합니다.
- DynamoDB에 연결을 저장하고 다른 채팅 참가자에게 메시지를 게시하는 Lambda 핸들러를 정의합니다.
- WebSocket 채팅 애플리케이션에 연결하고 WebSocket 패키지를 사용하여 메시지를 전송합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 TTL이 포함된 DynamoDB 항목 생성

다음 코드 예제는 TTL로 항목을 만드는 방법을 보여줍니다.

Java

SDK for Java 2.x

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
```

```
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

/**
 * Creates an item in a DynamoDB table with TTL attributes.
 * This class demonstrates how to add TTL expiration timestamps to DynamoDB
 items.
 */
public class CreateTTL {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String CREATION_DATE_ATTR = "creationDate";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String SUCCESS_MESSAGE = "%s PutItem operation with TTL
 successful.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
 DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs a CreateTTL instance with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
```

```
    public CreateTTL(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Constructs a CreateTTL with a default DynamoDB client.
     */
    public CreateTTL() {
        this.dynamoDbClient = null;
    }

    /**
     * Main method to demonstrate creating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new CreateTTL().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and create an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
```

```
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final CreateTTL createTTL = new CreateTTL(ddb);
            createTTL.createItemWithTTL(tableName, primaryKey, sortKey);
            return 0;
        } catch (Exception e) {
            throw e;
        }
    }

    /**
     * Creates an item in the specified table with TTL attributes.
     *
     * @param tableName The name of the table
     * @param primaryKeyValue The value for the primary key
     * @param sortKeyValue The value for the sort key
     * @return The response from the PutItem operation
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     */
    public PutItemResponse createItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
sortKeyValue) {
        // Get current time in epoch second format
        final long createDate = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = createDate + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        final Map<String, AttributeValue> itemMap = new HashMap<>();
        itemMap.put(
            PRIMARY_KEY_ATTR,
AttributeValue.builder().s(primaryKeyValue).build());
        itemMap.put(SORT_KEY_ATTR,
AttributeValue.builder().s(sortKeyValue).build());
        itemMap.put(
            CREATION_DATE_ATTR,
            AttributeValue.builder().n(String.valueOf(createDate)).build());
```

```
        itemMap.put(
            EXPIRE_AT_ATTR,
            AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final PutItemRequest request =
            PutItemRequest.builder().tableName(tableName).item(itemMap).build();

        try {
            final PutItemResponse response = dynamoDbClient.putItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 PutItem을 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 *
 1000) / 1000);
```

```
    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, err);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 PutItem을 참조하세요.

## Python

### SDK for Python(Boto3)

```
from datetime import datetime, timedelta

import boto3
```

```
def create_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Creates a DynamoDB item with an attached expiry attribute.

    :param table_name: Table name for the boto3 resource to target when creating
 an item
    :param region: string representing the AWS region. Example: `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expiration time (90 days from now) in epoch second format
        expiration_time = int((datetime.now() + timedelta(days=90)).timestamp())

        item = {
            "primaryKey": primary_key,
            "sortKey": sort_key,
            "creationDate": current_time,
            "expireAt": expiration_time,
        }
        response = table.put_item(Item=item)

        print("Item created successfully.")
        return response
    except Exception as e:
        print(f"Error creating item: {e}")
        raise e


# Use your own values
create_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 PutItem를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 다중 리전 강력한 일관성을 갖춘 DynamoDB 글로벌 테이블을 만들고 관리하는 방법

다음 코드 예제에서는 다중 리전 강력한 일관성(MRSC)을 갖춘 DynamoDB 글로벌 테이블을 만들고 관리하는 방법을 보여줍니다.

- 다중 리전 강력한 일관성이 있는 테이블을 만듭니다.
- MRSC 구성 및 복제본 상태를 확인합니다.
- 즉각적인 읽기를 통해 리전 간 강력한 일관성을 테스트합니다.
- MRSC 보장을 사용하여 조건부 쓰기를 수행합니다.
- MRSC 글로벌 테이블 리소스를 정리합니다.

Bash

Bash 스크립트와 함께 AWS CLI사용

다중 리전 강력한 일관성이 있는 테이블을 만듭니다.

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2
```

```
# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

MRSC 구성 및 복제본 상태를 확인합니다.

```
# Verify the global table configuration and MRSC setting
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

리전 간 즉각적인 읽기를 통해 강력한 일관성을 테스트합니다.

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
# No wait time needed - MRSC provides immediate consistency
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

MRSC 보장을 사용하여 조건부 쓰기를 수행합니다.

```
# Perform a conditional update from a different region
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

MRSC 글로벌 테이블 리소스를 정리합니다.

```
# Remove replica tables (must be done before deleting the primary table)
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용하여 MRSC 변환이 준비된 리전 테이블을 만듭니다.

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
                       .attributeType(ScalarAttributeType.S)
                       .build())
               .keySchema(
                   KeySchemaElement.builder()
                       .attributeName("Artist")
                       .keyType(KeyType.HASH)
                       .build(),
                   KeySchemaElement.builder()
                       .attributeName("SongTitle")
                       .keyType(KeyType.RANGE)
                       .build())
               .billingMode(BillingMode.PAY_PER_REQUEST)
               .build();
```

```
            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 리전 테이블을 복제본 및 감시자를 갖춘 MRSC로 변환합니다.

```
    public static UpdateTableResponse convertToMRSCWithWitness(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
```

```
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 글로벌 테이블 구성을 설명합니다.

```
   public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Describing MRSC global table: " + tableName);

           DescribeTableRequest request =
               DescribeTableRequest.builder().tableName(tableName).build();

           DescribeTableResponse response =
dynamoDbClient.describeTable(request);

           LOGGER.info("Table status: " + response.table().tableStatus());
           LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

           if (response.table().replicas() != null
               && !response.table().replicas().isEmpty()) {
               LOGGER.info("Number of replicas: " +
response.table().replicas().size());
               response.table()
                   .replicas()
                   .forEach(replica -> LOGGER.info(
                       "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
           }

           if (response.table().globalTableWitnesses() != null
               && !response.table().globalTableWitnesses().isEmpty()) {
               LOGGER.info("Number of witnesses: "
                   + response.table().globalTableWitnesses().size());
```

```
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 강력한 일관성을 확인하기 위한 테스트 항목을 추가합니다.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String album,
        final String year) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to MRSC global table: " + tableName);

            Map<String, AttributeValue> item = new HashMap<>();
            item.put("Artist", AttributeValue.builder().s(artist).build());
            item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            if (album != null && !album.trim().isEmpty()) {
                item.put("Album", AttributeValue.builder().s(album).build());
            }
            if (year != null && !year.trim().isEmpty()) {
                item.put("Year", AttributeValue.builder().n(year).build());
            }

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully with strong consistency");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 복제본에서 일관된 읽기로 항목을 읽습니다.

```
   public static GetItemResponse getItemWithConsistentRead(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (artist == null || artist.trim().isEmpty()) {
           throw new IllegalArgumentException("Artist cannot be null or empty");
       }
       if (songTitle == null || songTitle.trim().isEmpty()) {
           throw new IllegalArgumentException("Song title cannot be null or
empty");
       }

       try {
           LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);

           Map<String, AttributeValue> key = new HashMap<>();
           key.put("Artist", AttributeValue.builder().s(artist).build());
           key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

           GetItemRequest getItemRequest = GetItemRequest.builder()
               .tableName(tableName)
               .key(key)
               .consistentRead(true)
               .build();

           GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

           if (response.hasItem()) {
               LOGGER.info("Item found with strong consistency - no wait time
needed");
           } else {
               LOGGER.info("Item not found");
           }
```

```
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 보장이 있는 조건부 업데이트를 수행합니다.

```
    public static UpdateItemResponse performConditionalUpdate(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
            throw new IllegalArgumentException("Rating cannot be null or empty");
        }

        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);
```

```
            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");

            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");

            return response;

        } catch (ConditionalCheckFailedException e) {
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 복제본과 감시자가 활성화될 때까지 기다립니다.

```
   public static void waitForMRSCReplicasActive(
       final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
       throws InterruptedException {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (maxWaitTimeSeconds <= 0) {
           throw new IllegalArgumentException("Max wait time must be positive");
       }

       try {
           LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);

           final long startTime = System.currentTimeMillis();
           final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
           int backoffSeconds = 5; // Start with 5 second intervals
           final int maxBackoffSeconds = 30; // Cap at 30 seconds

           while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
               DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

               boolean allActive = true;
               StringBuilder statusReport = new StringBuilder();

               if (response.table().multiRegionConsistency() == null
                   || !MultiRegionConsistency.STRONG
                       .toString()

.equals(response.table().multiRegionConsistency().toString())) {
                   allActive = false;
                   statusReport
                       .append("MultiRegionConsistency: ")
                       .append(response.table().multiRegionConsistency())
                       .append(" ");
```

```
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }

                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }

                // Check witness status
                if (response.table().globalTableWitnesses() != null) {
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
```

```
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }

                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }

            throw DynamoDbException.builder()
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 복제본 및 감시자를 정리합니다.

```
    public static UpdateTableResponse cleanupMRSCReplicas(
        final DynamoDbClient dynamoDbClient,
```

```
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);

            // Remove replica using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .delete(DeleteReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Remove witness
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();
```

```
            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용하여 MRSC 워크플로 데모를 완료합니다.

```
    public static void demonstrateCompleteMRSCWorkflow(
        final DynamoDbClient primaryClient,
        final DynamoDbClient replicaClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion)
        throws InterruptedException {

        if (primaryClient == null) {
            throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
        }
        if (replicaClient == null) {
            throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
```

```
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

            // Step 1: Create an empty single-Region table
            LOGGER.info("Step 1: Creating empty single-Region table");
            createRegionalTable(primaryClient, tableName);

            // Use the existing GlobalTableOperations method for basic table
waiting
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 2: Convert to MRSC with replica and witness
            LOGGER.info("Step 2: Converting to MRSC with replica and witness");
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);

            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");

            // Add test item to primary region
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
```

```
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");

            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
            LOGGER.info("Waiting for replica cleanup to complete...");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }
```

```
            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

# AWS SDK를 사용하여 MREC를 보여주는 DynamoDB 글로벌 테이블 만들기 및 관리

다음 코드 예제에서는 다중 리전 간 복제본을 사용하여 DynamoDB 글로벌 테이블을 만들고 관리하는 방법을 보여줍니다.

- 글로벌 보조 인덱스와 DynamoDB Streams가 있는 테이블을 만듭니다.
- 다른 리전에 복제본을 추가하여 글로벌 테이블을 만듭니다.
- 글로벌 테이블에서 복제본을 제거합니다.
- 테스트 항목을 추가하여 리전 간 복제를 확인합니다.
- 글로벌 테이블 구성 및 복제본 상태를 설명합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용하여 글로벌 보조 인덱스와 DynamoDB Streams가 있는 테이블을 만듭니다.

```
   public static CreateTableResponse createTableWithGSI(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
```

```
                .tableName(tableName)
                .attributeDefinitions(
                    AttributeDefinition.builder()
                        .attributeName("Artist")
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                    .indexName(indexName)
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(

Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
```

```
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 테이블이 활성화될 때까지 기다립니다.

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 글로벌 테이블을 생성하거나 확장하기 위한 복제본을 추가합니다.

```
   public static UpdateTableResponse addReplica(
       final DynamoDbClient dynamoDbClient,
       final String tableName,
       final Region replicaRegion,
       final String indexName,
       final Long readCapacity) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (replicaRegion == null) {
           throw new IllegalArgumentException("Replica region cannot be null");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }
       if (readCapacity == null || readCapacity <= 0) {
           throw new IllegalArgumentException("Read capacity must be a positive
number");
       }

       try {
           LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

           // Create a ReplicationGroupUpdate for adding a replica
           ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
               .create(builder -> builder.regionName(replicaRegion.id())
                   .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                       .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
                           .readCapacityUnits(readCapacity)
                           .build())
                       .build())
                   .build())
               .build();
```

```
            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 글로벌 테이블에서 복제본을 제거합니다.

```
    public static UpdateTableResponse removeReplica(
        final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }

        try {
            LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);

            // Create a ReplicationGroupUpdate for removing a replica
```

```
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .delete(builder ->
builder.regionName(replicaRegion.id()).build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 테스트 항목을 추가함으로써 복제를 확인합니다.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to table: " + tableName);

            Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
            item.put(
                "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(artist)
                    .build());
            item.put(
                "SongTitle",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(songTitle)
                    .build());

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용하여 글로벌 테이블 구성 및 복제본을 설명합니다.

```
    public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {
```

```
        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x를 사용한 글로벌 테이블 작업의 전체 예제입니다.

```
   public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {

       String tableName = "Music";
       String indexName = "SongTitleIndex";
       Long readCapacity = 15L;

       // Create DynamoDB client for the source region
       try (DynamoDbClient dynamoDbClient =
           DynamoDbClient.builder().region(sourceRegion).build()) {

           try {
               // Step 1: Create the initial table with GSI and streams
               LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
               createTableWithGSI(dynamoDbClient, tableName, indexName);

               // Step 2: Wait for table to become active
               LOGGER.info("Step 2: Waiting for table to become active");
               waitForTableActive(dynamoDbClient, tableName);

               // Step 3: Add replica in destination region
               LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
               addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

               // Step 4: Wait a moment for replica creation to start
               Thread.sleep(5000);

               // Step 5: Describe table to view replica information
               LOGGER.info("Step 5: Describing table to view replicas");
               describeTable(dynamoDbClient, tableName);

               // Step 6: Add a test item to verify replication
               LOGGER.info("Step 6: Adding test item to verify replication");
               putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");

               LOGGER.info("Global table setup completed successfully!");
               LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());
```

```
                // Step 7: Remove replica and clean up table
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
                throw e;
            }
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - CreateTable
  - DescribeTable
  - PutItem
  - UpdateTable

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK와 함께 PartiQL DELETE 문을 사용하여 DynamoDB 데이터 삭제

다음 코드 예제에서는 PartiQL DELETE 문을 사용하여 데이터를 삭제하는 방법을 보여줍니다.

JavaScript

JavaScript용 SDK(v3)

AWS SDK for JavaScript와 함께 PartiQL DELETE 문을 사용하여 DynamoDB 테이블에서 항목을 삭제합니다.

```
/**
 * This example demonstrates how to delete items from a DynamoDB table using
 PartiQL.
 * It shows different ways to delete documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Delete a single item by its partition key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};
```

```
/**
 * Delete an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item with a condition to ensure the delete only happens if a
 condition is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
```

```
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${conditionAttribute} = ?`,
    Parameters: [partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item with condition:", err);
    throw err;
  }
};

/**
 * Batch delete multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param keys - Array of objects containing key information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchDeleteItems = async (
  tableName: string,
  keys: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    sortKeyName?: string;
    sortKeyValue?: string | number;
```

```
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each delete
  const statements = keys.map((key) => {
    if (key.sortKeyName && key.sortKeyValue !== undefined) {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?
 AND ${key.sortKeyName} = ?`,
        Parameters: [key.partitionKeyValue, key.sortKeyValue],
      };
    } else {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?
`,
        Parameters: [key.partitionKeyValue],
      };
    }
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch deleted successfully");
    return data;
  } catch (err) {
    console.error("Error batch deleting items:", err);
    throw err;
  }
};

/**
 * Delete multiple items that match a filter condition.
 * Note: This performs a scan operation which can be expensive on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
```

```
 */
export const deleteItemsByFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items deleted by filter successfully");
    return data;
  } catch (err) {
    console.error("Error deleting items by filter:", err);
    throw err;
  }
};

/**
 * Example usage showing how to delete items with different index types
 */
export const deleteExamples = async () => {
  // Delete an item by partition key (simple primary key)
  await deleteItemByPartitionKey("UsersTable", "userId", "user123");

  // Delete an item by composite key (partition key + sort key)
  await deleteItemByCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789"
  );

  // Delete with a condition
  await deleteItemWithCondition(
    "UsersTable",
    "userId",
```

```
    "user789",
    "userStatus",
    "inactive"
  );

  // Batch delete multiple items
  await batchDeleteItems("UsersTable", [
    { partitionKeyName: "userId", partitionKeyValue: "user234" },
    { partitionKeyName: "userId", partitionKeyValue: "user345" },
  ]);

  // Batch delete items with composite keys
  await batchDeleteItems("OrdersTable", [
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order567",
      sortKeyName: "productId",
      sortKeyValue: "prod123",
    },
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order678",
      sortKeyName: "productId",
      sortKeyValue: "prod456",
    },
  ]);

  // Delete items by filter (use with caution)
  await deleteItemsByFilter("UsersTable", "userStatus", "deleted");
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - BatchExecuteStatement
  - ExecuteStatement

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 Amazon Rekognition을 통해 이미지에서 PPE 감지

다음 코드 예제에서는 Amazon Rekognition을 사용하여 이미지에서 개인 보호 장비(PPE)를 감지하는 앱을 구축하는 방법을 보여줍니다.

Java

SDK for Java 2.x

개인 보호 장비로 이미지를 감지하는 AWS Lambda 함수를 만드는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK와 함께 PartiQL INSERT 문을 사용하여 DynamoDB 데이터 삽입

다음 코드 예제에서는 PartiQL INSERT 문을 사용하여 데이터를 삽입하는 방법을 보여줍니다.

JavaScript

JavaScript용 SDK(v3)

AWS SDK for JavaScript와 함께 PartiQL INSERT 문을 사용하여 DynamoDB 테이블에 항목을 삽입합니다.

```
/**
 * This example demonstrates how to insert items into a DynamoDB table using
 PartiQL.
 * It shows different ways to insert documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
```

```
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Insert a single item into a DynamoDB table using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItem = async (tableName: string, item: Record<string, any>) =>
 {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Convert the item to a string representation for PartiQL
  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item:", err);
    throw err;
  }
};

/**
 * Insert multiple items into a DynamoDB table using PartiQL batch operation.
 * This is more efficient than inserting items one by one.
 *
 * @param tableName - The name of the DynamoDB table
 * @param items - Array of items to insert
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchInsertItems = async (tableName: string, items: Record<string,
 any>[]) => {
```

```
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each item
  const statements = items.map((item) => {
    const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
    return {
      Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items inserted successfully");
    return data;
  } catch (err) {
    console.error("Error batch inserting items:", err);
    throw err;
  }
};

/**
 * Insert an item with a condition to prevent overwriting existing items.
 * This is useful for ensuring you don't accidentally overwrite data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @param partitionKeyName - The name of the partition key attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItemWithCondition = async (
  tableName: string,
  item: Record<string, any>,
  partitionKeyName: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
  const partitionKeyValue = JSON.stringify(item[partitionKeyName]);
```

```
  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString} WHERE
 attribute_not_exists(${partitionKeyName})`,
    Parameters: [{ S: partitionKeyValue }],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item with condition:", err);
    throw err;
  }
};

/**
 * Example usage showing how to insert items with different index types
 */
export const insertExamples = async () => {
  // Example table with a simple primary key (just partition key)
  const simpleKeyItem = {
    userId: "user123",
    name: "John Doe",
    email: "john@example.com",
  };
  await insertItem("UsersTable", simpleKeyItem);

  // Example table with composite key (partition key + sort key)
  const compositeKeyItem = {
    orderId: "order456",
    productId: "prod789",
    quantity: 2,
    price: 29.99,
  };
  await insertItem("OrdersTable", compositeKeyItem);

  // Example with Global Secondary Index (GSI)
  // The GSI might be on the email attribute
  const gsiItem = {
    userId: "user789",
    email: "jane@example.com",
    name: "Jane Smith",
```

```
    userType: "premium",  // This could be part of a GSI
  };
  await insertItem("UsersTable", gsiItem);

  // Example with Local Secondary Index (LSI)
  // LSI uses the same partition key but different sort key
  const lsiItem = {
    orderId: "order567",  // Partition key
    productId: "prod123", // Sort key for the table
    orderDate: "2023-11-15", // Potential sort key for an LSI
    quantity: 1,
    price: 19.99,
  };
  await insertItem("OrdersTable", lsiItem);

  // Batch insert example with multiple items
  const batchItems = [
    {
      userId: "user234",
      name: "Alice Johnson",
      email: "alice@example.com",
    },
    {
      userId: "user345",
      name: "Bob Williams",
      email: "bob@example.com",
    },
  ];
  await batchInsertItems("UsersTable", batchItems);
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - BatchExecuteStatement
  - ExecuteStatement

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## 브라우저에서 Lambda 함수 간접 호출

다음 코드 예제에서는 브라우저에서 AWS Lambda 함수를 간접 호출하는 방법을 보여줍니다.

JavaScript

JavaScript용 SDK(v2)

AWS Lambda 함수를 사용하여 사용자 선택 사항으로 Amazon DynamoDB 테이블을 업데이트하는 브라우저 기반 애플리케이션을 생성할 수 있습니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Lambda

SDK for JavaScript (v3)

AWS Lambda 함수를 사용하여 사용자 선택 사항으로 Amazon DynamoDB 테이블을 업데이트하는 브라우저 기반 애플리케이션을 생성할 수 있습니다. 이 앱은 AWS SDK for JavaScript v3를 사용합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Lambda

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS Command Line Interface v2를 사용하여 DynamoDB 글로벌 보조 인덱스 관리

다음 코드 예제에서는 글로벌 보조 인덱스의 전체 수명 주기를 관리하는 방법을 보여줍니다.

- 글로벌 보조 인덱스가 있는 테이블을 생성합니다.
- 기존 테이블에 새 GSI를 추가합니다.
- GSI 원 처리량을 업데이트(증가)합니다.
- GSI를 사용하여 데이터를 쿼리합니다.
- GSI를 삭제합니다.

Bash

Bash 스크립트와 함께 AWS CLI사용

글로벌 보조 인덱스가 있는 테이블을 생성합니다.

```
# Create a table with a GSI
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "IndexName=AlbumIndex,\
        KeySchema=[{AttributeName=AlbumTitle,KeyType=HASH}],\
        Projection={ProjectionType=ALL}"
```

기존 테이블에 새 (온디맨드) GSI를 추가합니다.

```
# Add a new GSI to an existing table
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Genre,AttributeType=S \
    --global-secondary-index-updates \
        "[{\"Create\":{\"IndexName\":\"GenreIndex\",\
        \"KeySchema\":[{\"AttributeName\":\"Genre\",\"KeyType\":\"HASH\"}],\
        \"Projection\":{\"ProjectionType\":\"ALL\"}}}]"
```

GSI 웜 처리량을 업데이트(증가)합니다.

```
# Increase the warm throughput of a GSI (default values are 12k reads, 4k writes)
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
```

```
        "[{\"Update\":{\"IndexName\":\"AlbumIndex\",\
        \"WarmThroughput\":{\"ReadUnitsPerSecond\":15000,\"WriteUnitsPerSecond
\":6000}}}]"
```

GSI를 사용하여 데이터를 쿼리합니다.

```
# Query the AlbumIndex GSI
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumIndex \
    --key-condition-expression "AlbumTitle = :album" \
    --expression-attribute-values '{":album":{"S":"Let It Be"}}'

# Query the GenreIndex GSI
aws dynamodb query \
    --table-name MusicCollection \
    --index-name GenreIndex \
    --key-condition-expression "Genre = :genre" \
    --expression-attribute-values '{":genre":{"S":"Jazz"}}'
```

GSI를 삭제합니다.

```
# Delete a GSI from a table
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Delete\":{\"IndexName\":\"GenreIndex\"}}]"
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - CreateTable
  - DeleteTable
  - Query
  - UpdateTable

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

# AWS Command Line Interface v2를 사용하여 DynamoDB 리소스 기반 정책 관리

다음 코드 예제에서는 DynamoDB 테이블에 대한 리소스 기반 정책의 전체 수명 주기를 관리하는 방법을 보여줍니다.

- 리소스 정책을 포함한 테이블을 생성합니다.
- 리소스 정책을 가져옵니다.
- 리소스 정책을 업데이트합니다.
- 리소스 정책을 삭제합니다.

Bash

Bash 스크립트와 함께 AWS CLI사용

리소스 정책을 포함한 테이블을 생성합니다.

```
# Step 1: Create a DynamoDB table
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST

# Step 2: Create a resource-based policy document
cat > policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/DynamoDBReadOnly"
      },
      "Action": [
        "dynamodb:GetItem",
```

```
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
    }
  ]
}
EOF

# Step 3: Attach the resource-based policy to the table
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
 \
    --policy file://policy.json
```

리소스 정책을 가져옵니다.

```
# Get the resource-based policy attached to a table
aws dynamodb get-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

리소스 정책을 업데이트합니다.

```
# Step 1: Create an updated policy document
cat > updated-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:role/DynamoDBReadOnly",
          "arn:aws:iam::123456789012:role/DynamoDBAnalytics"
        ]
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
```

```
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
    }
  ]
}
EOF

# Step 2: Update the resource-based policy on the table
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
 \
    --policy file://updated-policy.json
```

리소스 정책을 삭제합니다.

```
# Delete the resource-based policy from a table
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - CreateTable
  - DeleteResourcePolicy
  - GetResourcePolicy
  - PutResourcePolicy

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 Amazon DynamoDB의 성능 모니터링

다음 코드 예제는 성능 모니터링을 위해 애플리케이션의 DynamoDB 사용을 구성하는 방법을 보여줍니다.

Java

SDK for Java 2.x

이 예제는 DynamoDB의 성능을 모니터링하도록 Java 애플리케이션을 구성하는 방법을 보여줍니다. 애플리케이션은 성능을 모니터링할 수 있는 CloudWatch로 지표 데이터를 전송합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예시에서 사용되는 서비스

- CloudWatch
- DynamoDB

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK로 고급 DynamoDB 쿼리 작업 수행

다음 코드 예제에서는 DynamoDB에서 고급 쿼리 작업을 수행하는 방법을 보여줍니다.

- 다양한 필터링 및 조건 기법을 사용하여 테이블을 쿼리합니다.
- 대규모 결과 세트에 페이지 매김을 구현합니다.
- 대체 액세스 패턴에 글로벌 보조 인덱스를 사용합니다.
- 애플리케이션 요구 사항에 따라 일관성 제어를 적용합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x로 강력하게 일관된 읽기를 사용하여 쿼리합니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
```

```
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithConsistentReads(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final boolean useConsistentRead) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .consistentRead(useConsistentRead)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query successful. Found {0} items",
 response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with consistent reads", e);
            throw e;
```

```
        }
    }
```

AWS SDK for Java 2.x로 글로벌 보조 인덱스를 사용하여 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryTable(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();
```

```
        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on base table successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw new DynamoDbQueryException("Table not found: " + tableName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying base table: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on base
table", e);
        }
    }

    /**
     * Queries a DynamoDB Global Secondary Index (GSI) by partition key.
     *
     * @param tableName         The name of the DynamoDB table
     * @param indexName         The name of the GSI
     * @param partitionKeyName  The name of the GSI partition key attribute
     * @param partitionKeyValue The value of the GSI partition key to query
     * @return The query response from DynamoDB
     * @throws ResourceNotFoundException if the table or index doesn't exist
     * @throws DynamoDbException if the query fails
     */
    public QueryResponse queryGlobalSecondaryIndex(
        final String tableName, final String indexName, final String
partitionKeyName, final String partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Index name", indexName);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_IK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
```

```
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_IK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .indexName(indexName)
            .keyConditionExpression(GSI_KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on GSI successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(
                "Error: The Amazon DynamoDB table \"%s\" or index \"%s\" can't be
found.\n", tableName, indexName);
            throw new DynamoDbQueryException("Table or index not found: " +
tableName + "/" + indexName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying GSI: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on GSI",
e);
        }
    }
```

AWS SDK for Java 2.x로 페이지 매김을 사용하여 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.ArrayList;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    public List<Map<String, AttributeValue>> queryWithPagination(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue, final int pageSize) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validatePositiveInteger("Page size", pageSize);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(pageSize);

        // List to store all items from all pages
        final List<Map<String, AttributeValue>> allItems = new ArrayList<>();

        // Map to store the last evaluated key for pagination
        Map<String, AttributeValue> lastEvaluatedKey = null;
        int pageNumber = 1;

        try {
            do {
                // If we have a last evaluated key, use it for the next page
                if (lastEvaluatedKey != null) {
                    queryRequestBuilder.exclusiveStartKey(lastEvaluatedKey);
```

```
                }

                // Execute the query
                final QueryResponse response =
dynamoDbClient.query(queryRequestBuilder.build());

                // Process the current page of results
                final List<Map<String, AttributeValue>> pageItems =
response.items();
                allItems.addAll(pageItems);

                // Get the last evaluated key for the next page
                lastEvaluatedKey = response.lastEvaluatedKey();
                if (lastEvaluatedKey != null && lastEvaluatedKey.isEmpty()) {
                    lastEvaluatedKey = null;
                }

                System.out.println("Page " + pageNumber + ": Retrieved " +
pageItems.size() + " items (Running total: "
                    + allItems.size() + ")");

                pageNumber++;

            } while (lastEvaluatedKey != null);

            System.out.println("Query with pagination complete. Retrieved a total
of " + allItems.size()
                + " items across " + (pageNumber - 1) + " pages");

            return allItems;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with pagination: " +
e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x로 복잡한 필터를 사용하여 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithComplexFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String statusAttrName,
        final String activeStatus,
        final String pendingStatus,
        final String priceAttrName,
        final double minPrice,
        final double maxPrice,
        final String categoryAttrName) {

        // Validate parameters
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Status attribute name",
 statusAttrName);
        CodeSampleUtils.validateStringParameter("Active status", activeStatus);
        CodeSampleUtils.validateStringParameter("Pending status", pendingStatus);
        CodeSampleUtils.validateStringParameter("Price attribute name",
 priceAttrName);
        CodeSampleUtils.validateStringParameter("Category attribute name",
 categoryAttrName);
        CodeSampleUtils.validateNumericRange("Minimum price", minPrice, 0.0,
 Double.MAX_VALUE);
        CodeSampleUtils.validateNumericRange("Maximum price", maxPrice, minPrice,
 Double.MAX_VALUE);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
```

```
        expressionAttributeNames.put("#pk", partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_STATUS,
statusAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PRICE,
priceAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_CATEGORY,
categoryAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":pkValue", AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_ACTIVE,
            AttributeValue.builder().s(activeStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PENDING,
            AttributeValue.builder().s(pendingStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MIN_PRICE,
            AttributeValue.builder().n(String.valueOf(minPrice)).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MAX_PRICE,
            AttributeValue.builder().n(String.valueOf(maxPrice)).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        return dynamoDbClient.query(queryRequest);
    }
```

AWS SDK for Java 2.x로 동적으로 구성된 필터 표현식을 사용하여 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public static QueryResponse queryWithDynamicFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final Map<String, Object> filterCriteria,
        final Region region,
        final DynamoDbClient dynamoDbClient) {

        validateParameters(tableName, partitionKeyName, partitionKeyValue,
 filterCriteria);

        DynamoDbClient ddbClient = dynamoDbClient;
        boolean shouldClose = false;

        try {
            if (ddbClient == null) {
                ddbClient = createClient(region);
                shouldClose = true;
            }

            final QueryWithDynamicFilter queryHelper = new
 QueryWithDynamicFilter(ddbClient);
            return queryHelper.queryWithDynamicFilter(tableName,
 partitionKeyName, partitionKeyValue, filterCriteria);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Failed to execute dynamic filter query: " +
 e.getMessage());
            throw e;
        } catch (Exception e) {
            System.err.println("Unexpected error during query: " +
 e.getMessage());
            throw e;
```

```
        } finally {
            if (shouldClose && ddbClient != null) {
                ddbClient.close();
            }
        }
    }

    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<filterAttrName> <filterAttrValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    filterAttrName - The name of the attribute to filter on.
                    filterAttrValue - The value to filter by.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 5) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final String filterAttrName = args[3];
        final String filterAttrValue = args[4];
        final Region region = args.length > 5 ? Region.of(args[5]) :
Region.US_EAST_1;

        System.out.println("Querying items with dynamic filter: " +
filterAttrName + " = " + filterAttrValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
```

```
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriterion(filterAttrName, filterAttrValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> System.out.println(item));

            // Demonstrate multiple filter criteria
            System.out.println("\nNow querying with multiple filter criteria:");

            Map<String, Object> multipleFilters = new HashMap<>();
            multipleFilters.put(filterAttrName, filterAttrValue);
            multipleFilters.put("status", "active");

            final QueryResponse multiFilterResponse = new
DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriteria(multipleFilters)
                .withRegion(region)
                .execute();

            System.out.println("Found " + multiFilterResponse.count() + " items
with multiple filters:");
            multiFilterResponse.items().forEach(item ->
System.out.println(item));

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

AWS SDK for Java 2.x로 필터 표현식 및 제한을 사용하여 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithFilterAndLimit(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String filterAttrName,
        final String filterAttrValue,
        final int limit) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Filter attribute name",
 filterAttrName);
        CodeSampleUtils.validateStringParameter("Filter attribute value",
 filterAttrValue);
        CodeSampleUtils.validatePositiveInteger("Limit", limit);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_FILTER,
 filterAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
```

```
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_FILTER,
            AttributeValue.builder().s(filterAttrValue).build());

        // Create the filter expression
        final String filterExpression = "#filterAttr = :filterValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(limit)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with filter and limit successful. Found
{0} items", response.count());
            LOGGER.log(
                Level.INFO, "ScannedCount: {0} (total items evaluated before
filtering)", response.scannedCount());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with filter and limit: {0}",
e.getMessage());
            throw e;
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript로 강력하게 일관된 읽기를 사용하여 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
```

```
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

AWS SDK for JavaScript로 글로벌 보조 인덱스를 사용하여 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
```

```
    throw error;
  }
}

/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

AWS SDK for JavaScript로 페이지 매김을 사용하여 쿼리합니다.

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
```

```
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };

      // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
      if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
      }

      // Execute the query
      const command = new QueryCommand(input);
      const response = await client.send(command);

      // Process the current page of results
      pageCount++;
      console.log(`Processing page ${pageCount} with ${response.Items.length}
 items`);

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
      }

      // Get the LastEvaluatedKey for the next page
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages

    console.log(`Query complete. Retrieved ${allItems.length} items in
 ${pageCount} pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}
```

```
/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to
 retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

AWS SDK for JavaScript로 복잡한 필터를 사용하여 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
```

```
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

AWS SDK for JavaScript로 동적으로 구성된 필터 표현식을 사용하여 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");
```

```
async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
 filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }
```

```
    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)로 강력하게 일관된 읽기를 사용하여 쿼리합니다.

```
import time

import boto3
from boto3.dynamodb.conditions import Key


def query_with_consistent_read(
    table_name,
    partition_key_name,
    partition_key_value,
```

```
    sort_key_name=None,
    sort_key_value=None,
    consistent_read=True,
):
    """
    Query a DynamoDB table with the option for strongly consistent reads.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str, optional): The name of the sort key attribute.
        sort_key_value (str, optional): The value of the sort key to query.
        consistent_read (bool, optional): Whether to use strongly consistent
 reads. Defaults to True.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    if sort_key_name and sort_key_value:
        key_condition = key_condition & Key(sort_key_name).eq(sort_key_value)

    # Perform the query with the consistent read option
    response = table.query(KeyConditionExpression=key_condition,
 ConsistentRead=consistent_read)

    return response
```

AWS SDK for Python (Boto3)로 글로벌 보조 인덱스를 사용하여 쿼리합니다.

```
import boto3
from boto3.dynamodb.conditions import Key
```

```
def query_table(table_name, partition_key_name, partition_key_value):
    """
    Query a DynamoDB table using its primary key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the table's primary key
    response =
 table.query(KeyConditionExpression=Key(partition_key_name).eq(partition_key_value))

    return response


def query_gsi(table_name, index_name, partition_key_name, partition_key_value):
    """
    Query a Global Secondary Index (GSI) on a DynamoDB table.

    Args:
        table_name (str): The name of the DynamoDB table.
        index_name (str): The name of the Global Secondary Index.
        partition_key_name (str): The name of the GSI's partition key attribute.
        partition_key_value (str): The value of the GSI's partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the GSI
    response = table.query(
```

```
        IndexName=index_name,
 KeyConditionExpression=Key(partition_key_name).eq(partition_key_value)
    )

    return response
```

AWS SDK for Python (Boto3)로 페이지 매김을 사용하여 쿼리합니다.

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_pagination(
    table_name, partition_key_name, partition_key_value, page_size=25,
 max_pages=None
):
    """
    Query a DynamoDB table with pagination to handle large result sets.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.
        max_pages (int, optional): The maximum number of pages to retrieve. If
 None, retrieves all pages.

    Returns:
        list: All items retrieved from the query across all pages.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_count = 0
    all_items = []
```

```
    # Paginate through the results
    while True:
        # Check if we've reached the maximum number of pages
        if max_pages is not None and page_count >= max_pages:
            break

        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
 Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Process the current page of results
        items = response.get("Items", [])
        all_items.extend(items)

        # Update pagination tracking
        page_count += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if not last_evaluated_key:
            break

    return all_items


def query_with_pagination_generator(
    table_name, partition_key_name, partition_key_value, page_size=25
):
    """
    Query a DynamoDB table with pagination using a generator to handle large
 result sets.
```

```
    This approach is memory-efficient as it yields one page at a time.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
Defaults to 25.

    Yields:
        tuple: A tuple containing (items, page_number, last_page) where:
            - items is a list of items for the current page
            - page_number is the current page number (starting from 1)
            - last_page is a boolean indicating if this is the last page
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_number = 0

    # Paginate through the results
    while True:
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Get the current page of results
        items = response.get("Items", [])
        page_number += 1
```

```
        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # Determine if this is the last page
        is_last_page = last_evaluated_key is None

        # Yield the current page of results
        yield (items, page_number, is_last_page)

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if is_last_page:
            break
```

AWS SDK for Python (Boto3)로 복잡한 필터를 사용하여 쿼리합니다.

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_complex_filter(
    table_name,
    partition_key_name,
    partition_key_value,
    min_rating=None,
    status_list=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        min_rating (float, optional): Minimum rating value for filtering.
        status_list (list, optional): List of status values to include.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
```

```
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize the filter expression and expression attribute values
    filter_expression = None
    expression_attribute_values = {}

    # Build the filter expression based on provided parameters
    if min_rating is not None:
        filter_expression = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if status_list and len(status_list) > 0:
        status_condition = None
        for i, status in enumerate(status_list):
            status_value_name = f":status{i}"
            expression_attribute_values[status_value_name] = status

            if status_condition is None:
                status_condition = Attr("status").eq(status)
            else:
                status_condition = status_condition | Attr("status").eq(status)

        if filter_expression is None:
            filter_expression = status_condition
        else:
            filter_expression = filter_expression & status_condition

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if filter_expression is None:
            filter_expression = price_condition
        else:
            filter_expression = filter_expression & price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}
```

```
    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
 expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response


def query_with_complex_filter_and_or(
    table_name,
    partition_key_name,
    partition_key_value,
    category=None,
    min_rating=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression using AND and OR
 operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        category (str, optional): Category value for filtering.
        min_rating (float, optional): Minimum rating value for filtering.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build a complex filter expression with AND and OR operators
    filter_expression = None
```

```
    expression_attribute_values = {}

    # Build the category condition
    if category:
        filter_expression = Attr("category").eq(category)
        expression_attribute_values[":category"] = category

    # Build the rating and price condition (rating >= min_rating OR price <=
max_price)
    rating_price_condition = None

    if min_rating is not None:
        rating_price_condition = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if rating_price_condition is None:
            rating_price_condition = price_condition
        else:
            rating_price_condition = rating_price_condition | price_condition

    # Combine the conditions
    if rating_price_condition:
        if filter_expression is None:
            filter_expression = rating_price_condition
        else:
            filter_expression = filter_expression & rating_price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response
```

AWS SDK for Python (Boto3)로 동적으로 구성된 필터 표현식을 사용하여 쿼리합니다.

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_dynamic_filter(
    table_name, partition_key_name, partition_key_value, filter_conditions=None
):
    """
    Query a DynamoDB table with a dynamically constructed filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_conditions (dict, optional): A dictionary of filter conditions
 where
            keys are attribute names and values are dictionaries with 'operator'
 and 'value'.
            Example: {'rating': {'operator': '>=', 'value': 4}, 'status':
 {'operator': '=', 'value': 'active'}}

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize variables for the filter expression and attribute values
    filter_expression = None
    expression_attribute_values = {":pk_val": partition_key_value}

    # Dynamically build the filter expression if filter conditions are provided
    if filter_conditions:
        for attr_name, condition in filter_conditions.items():
```

```
            operator = condition.get("operator")
            value = condition.get("value")
            attr_value_name = f":{attr_name}"
            expression_attribute_values[attr_value_name] = value

            # Create the appropriate filter expression based on the operator
            current_condition = None
            if operator == "=":
                current_condition = Attr(attr_name).eq(value)
            elif operator == "!=":
                current_condition = Attr(attr_name).ne(value)
            elif operator == ">":
                current_condition = Attr(attr_name).gt(value)
            elif operator == ">=":
                current_condition = Attr(attr_name).gte(value)
            elif operator == "<":
                current_condition = Attr(attr_name).lt(value)
            elif operator == "<=":
                current_condition = Attr(attr_name).lte(value)
            elif operator == "contains":
                current_condition = Attr(attr_name).contains(value)
            elif operator == "begins_with":
                current_condition = Attr(attr_name).begins_with(value)

            # Combine with existing filter expression using AND
            if current_condition:
                if filter_expression is None:
                    filter_expression = current_condition
                else:
                    filter_expression = filter_expression & current_condition

    # Perform the query with the dynamically built filter expression
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression

    response = table.query(**query_params)
    return response
```

AWS SDK for Python (Boto3)로 필터 표현식 및 제한을 사용하여 쿼리합니다.

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_filter_and_limit(
    table_name,
    partition_key_name,
    partition_key_value,
    filter_attribute=None,
    filter_value=None,
    limit=10,
):
    """
    Query a DynamoDB table with a filter expression and limit the number of
 results.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute (str, optional): The attribute name to filter on.
        filter_value (any, optional): The value to compare against in the filter.
        limit (int, optional): The maximum number of items to evaluate. Defaults
 to 10.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition, "Limit": limit}

    # Add the filter expression if filter attributes are provided
    if filter_attribute and filter_value is not None:
        query_params["FilterExpression"] =
 Attr(filter_attribute).gt(filter_value)
```

```
        query_params["ExpressionAttributeValues"] = {":filter_value":
filter_value}

    # Execute the query
    response = table.query(**query_params)
    return response
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 DynamoDB에서 목록 작업 수행

다음 코드 예제에서는 DynamoDB에서 목록 작업을 수행하는 방법을 보여줍니다.

- 목록 속성에 요소를 추가합니다.
- 목록 속성에서 요소를 제거합니다.
- 인덱스별로 목록의 특정 요소를 업데이트합니다.
- 목록 추가 및 목록 인덱스 함수를 사용합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용한 목록 작업을 보여줍니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.ArrayList;
import java.util.HashMap;
```

```
import java.util.List;
import java.util.Map;

    /**
     * Appends items to a list attribute.
     *
     * <p>This method demonstrates how to use the list_append function to add
     * items to the end of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param itemsToAppend The items to append to the list
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse appendToList(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        List<AttributeValue> itemsToAppend) {

        // Create a list value from the items to append
        AttributeValue listValue =
 AttributeValue.builder().l(itemsToAppend).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName =
 list_append(if_not_exists(#attrName, :emptyList), :newItems)")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(
                ":newItems",
                listValue,
                ":emptyList",
                AttributeValue.builder().l(new
 ArrayList<AttributeValue>()).build()))
            .returnValues("UPDATED_NEW")
            .build();
```

```
        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Prepends items to a list attribute.
     *
     * <p>This method demonstrates how to use the list_append function to add
     * items to the beginning of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param itemsToPrepend The items to prepend to the list
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse prependToList(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        List<AttributeValue> itemsToPrepend) {

        // Create a list value from the items to prepend
        AttributeValue listValue =
AttributeValue.builder().l(itemsToPrepend).build();

        // Define the update parameters
        // Note: To prepend, we put the new items first in the list_append
function
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName = list_append(:newItems,
if_not_exists(#attrName, :emptyList))")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(
                ":newItems",
                listValue,
                ":emptyList",
                AttributeValue.builder().l(new
ArrayList<AttributeValue>()).build()))
```

```
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Updates a specific element in a list attribute.
     *
     * <p>This method demonstrates how to update a specific element in a list
     * by its index.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param index The index of the element to update
     * @param newValue The new value for the element
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateListElement(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        int index,
        AttributeValue newValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName[" + index + "] = :newValue")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(":newValue", newValue))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }
```

```
    /**
     * Removes a specific element from a list attribute.
     *
     * <p>This method demonstrates how to remove a specific element from a list
     * by its index.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param index The index of the element to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeListElement(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        int index) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #attrName[" + index + "]")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a list attribute.
     *
     * <p>Helper method to retrieve the current value of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param listAttributeName The name of the list attribute
     * @return The list attribute value or null if not found
```

```
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static List<AttributeValue> getListAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String listAttributeName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(listAttributeName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the list attribute if it exists, otherwise null
            if (response.item() != null &&
response.item().containsKey(listAttributeName)) {
                return response.item().get(listAttributeName).l();
            }

            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get list attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용한 목록 작업의 사용 예입니다.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating list operations in DynamoDB");
```

```
        try {
            // Example 1: Append items to a list
            System.out.println("\nExample 1: Appending items to a list");
            List<AttributeValue> tagsToAppend = List.of(
                AttributeValue.builder().s("Electronics").build(),
                AttributeValue.builder().s("Gadget").build());

            UpdateItemResponse appendResponse = appendToList(dynamoDbClient,
tableName, key, "Tags", tagsToAppend);

            System.out.println("Updated list attribute: " +
appendResponse.attributes());

            // Example 2: Prepend items to a list
            System.out.println("\nExample 2: Prepending items to a list");
            List<AttributeValue> tagsToPrepend = List.of(
                AttributeValue.builder().s("Featured").build(),
                AttributeValue.builder().s("New").build());

            UpdateItemResponse prependResponse = prependToList(dynamoDbClient,
tableName, key, "Tags", tagsToPrepend);

            System.out.println("Updated list attribute: " +
prependResponse.attributes());

            // Example 3: Update a specific element in a list
            System.out.println("\nExample 3: Updating a specific element in a
list");
            UpdateItemResponse updateResponse = updateListElement(
                dynamoDbClient,
                tableName,
                key,
                "Tags",
                0,
                AttributeValue.builder().s("BestSeller").build());

            System.out.println("Updated list attribute: " +
updateResponse.attributes());

            // Example 4: Remove a specific element from a list
            System.out.println("\nExample 4: Removing a specific element from a
list");
            UpdateItemResponse removeResponse = removeListElement(dynamoDbClient,
tableName, key, "Tags", 1);
```

```
            System.out.println("Updated list attribute: " +
removeResponse.attributes());

            // Example 5: Get the current value of a list attribute
            System.out.println("\nExample 5: Getting the current value of a list
attribute");
            List<AttributeValue> currentList = getListAttribute(dynamoDbClient,
tableName, key, "Tags");

            if (currentList != null) {
                System.out.println("Current list attribute:");
                for (int i = 0; i < currentList.size(); i++) {
                    System.out.println("  [" + i + "]: " +
currentList.get(i).s());
                }
            } else {
                System.out.println("List attribute not found");
            }

            // Explain list operations
            System.out.println("\nKey points about DynamoDB list operations:");
            System.out.println("1. Lists are ordered collections of attributes");
            System.out.println("2. Use list_append to add items to a list");
            System.out.println("3. To append items, use list_append(existingList,
newItems)");
            System.out.println("4. To prepend items, use list_append(newItems,
existingList)");
            System.out.println("5. Use index notation (list[0]) to access or
update specific elements");
            System.out.println("6. Use REMOVE to delete elements from a list");
            System.out.println("7. List indices are zero-based");
            System.out.println("8. Use if_not_exists to handle the case where the
list doesn't exist yet");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용한 목록 작업을 보여줍니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Append elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the end of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to append to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function appendToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} =
 list_append(if_not_exists(${listName}, :empty_list), :values)`,
```

```
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Prepend elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the beginning of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to prepend to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function prependToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  // Note: To prepend, we put the new values first in the list_append function
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = list_append(:values,
 if_not_exists(${listName}, :empty_list))`,
```

```
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update a specific element in a list by index.
 *
 * This function demonstrates how to update a specific element in a list
 * using the index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to update
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateListElement(
  config,
  tableName,
  key,
  listName,
  index,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${index}] = :value`,
```

```
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove an element from a list by index.
 *
 * This function demonstrates how to remove a specific element from a list
 * using the REMOVE action with index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to remove
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeListElement(
  config,
  tableName,
  key,
  listName,
  index
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using REMOVE with index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `REMOVE ${listName}[${index}]`,
    ReturnValues: "UPDATED_NEW"
  };
```

```
  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Concatenate two lists.
 *
 * This function demonstrates how to concatenate two lists using the list_append
 function.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName1 - The name of the first list attribute
 * @param {string} listName2 - The name of the second list attribute
 * @param {string} resultListName - The name of the attribute to store the
 concatenated list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function concatenateLists(
  config,
  tableName,
  key,
  listName1,
  listName2,
  resultListName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${resultListName} =
 list_append(if_not_exists(${listName1}, :empty_list),
 if_not_exists(${listName2}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": []
    },
    ReturnValues: "UPDATED_NEW"
```

```
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a nested list structure.
 *
 * This function demonstrates how to create and work with nested lists.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} nestedLists - An array of arrays to create a nested list
 structure
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createNestedList(
  config,
  tableName,
  key,
  listName,
  nestedLists
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters to create a nested list
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = :nested_lists`,
    ExpressionAttributeValues: {
      ":nested_lists": nestedLists
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
```

```
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an element in a nested list.
 *
 * This function demonstrates how to update an element in a nested list
 * using multiple index notations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} outerIndex - The index in the outer list
 * @param {number} innerIndex - The index in the inner list
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedListElement(
  config,
  tableName,
  key,
  listName,
  outerIndex,
  innerIndex,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using multiple index notations
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${outerIndex}][${innerIndex}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };
```

```
  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

AWS SDK for JavaScript를 사용한 목록 작업의 사용 예입니다.

```
/**
 * Example of how to work with lists in DynamoDB.
```

```
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "UserProfiles";
  const key = { UserId: "U12345" };

  console.log("Demonstrating list operations in DynamoDB");

  try {
    // Example 1: Append elements to a list
    console.log("\nExample 1: Appending elements to a list");
    const response1 = await appendToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["laptop", "headphones", "monitor"]
    );

    console.log("Appended to list:", response1.Attributes);

    // Example 2: Prepend elements to a list
    console.log("\nExample 2: Prepending elements to a list");
    const response2 = await prependToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["keyboard", "mouse"]
    );

    console.log("Prepended to list:", response2.Attributes);

    // Get the current state of the item
    let currentItem = await getItem(config, tableName, key);
    console.log("\nCurrent state of RecentSearches:",
 currentItem?.RecentSearches);

    // Example 3: Update a specific element in a list
    console.log("\nExample 3: Updating a specific element in a list");
    const response3 = await updateListElement(
      config,
      tableName,
```

```
      key,
      "RecentSearches",
      0, // Update the first element
      "mechanical keyboard" // New value
    );

    console.log("Updated list element:", response3.Attributes);

    // Example 4: Remove an element from a list
    console.log("\nExample 4: Removing an element from a list");
    const response4 = await removeListElement(
      config,
      tableName,
      key,
      "RecentSearches",
      2 // Remove the third element
    );

    console.log("List after removing element:", response4.Attributes);

    // Example 5: Create and concatenate lists
    console.log("\nExample 5: Creating and concatenating lists");

    // First, create two separate lists
    await updateWithMultipleActions(
      config,
      tableName,
      key,
      "SET WishList = :wishlist, SavedItems = :saveditems",
      null,
      {
        ":wishlist": ["gaming laptop", "wireless earbuds"],
        ":saveditems": ["smartphone", "tablet"]
      }
    );

    // Then, concatenate them
    const response5 = await concatenateLists(
      config,
      tableName,
      key,
      "WishList",
      "SavedItems",
      "AllItems"
```

```
    );

    console.log("Concatenated lists:", response5.Attributes);

    // Example 6: Create a nested list structure
    console.log("\nExample 6: Creating a nested list structure");
    const response6 = await createNestedList(
      config,
      tableName,
      key,
      "Categories",
      [
        ["Electronics", "Computers", "Accessories"],
        ["Books", "Magazines", "E-books"],
        ["Clothing", "Shoes", "Watches"]
      ]
    );

    console.log("Created nested list:", response6.Attributes);

    // Example 7: Update an element in a nested list
    console.log("\nExample 7: Updating an element in a nested list");
    const response7 = await updateNestedListElement(
      config,
      tableName,
      key,
      "Categories",
      0, // First inner list
      1, // Second element in that list
      "Laptops" // New value
    );

    console.log("Updated nested list element:", response7.Attributes);

    // Get the final state of the item
    currentItem = await getItem(config, tableName, key);
    console.log("\nFinal state of the item:", JSON.stringify(currentItem, null,
2));

    // Explain list operations
    console.log("\nKey points about list operations in DynamoDB:");
    console.log("1. Use list_append to add elements to a list");
    console.log("2. To append elements, use list_append(existingList,
newElements)");
```

```
    console.log("3. To prepend elements, use list_append(newElements,
 existingList)");
    console.log("4. Use if_not_exists to handle cases where the list might not
 exist yet");
    console.log("5. Use index notation (list[0]) to access or update specific
 elements");
    console.log("6. Use REMOVE with index notation to remove elements from a
 list");
    console.log("7. Lists can contain elements of different types");
    console.log("8. Lists can be nested (lists of lists)");
    console.log("9. Use multiple index notations (list[0][1]) to access nested
 list elements");

  } catch (error) {
    console.error("Error:", error);
  }
}

/**
 * Helper function for the examples.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} updateExpression - The update expression
 * @param {Object} expressionAttributeNames - Expression attribute name
 placeholders
 * @param {Object} expressionAttributeValues - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
```

```
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용한 목록 작업을 보여줍니다.

```
import boto3
import json
from typing import Any, Dict, List, Optional, Union


def create_list_attribute(
    table_name: str, key: Dict[str, Any], list_name: str, list_values: List[Any]
) -> Dict[str, Any]:
    """
    Create a new list attribute or replace an existing one.
```

```
    This function demonstrates how to create a new list attribute or replace
    an existing list with new values.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        list_values (List[Any]): The values to set in the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the SET operation to create or replace the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = :list_values",
        ExpressionAttributeValues={":list_values": list_values},
        ReturnValues="UPDATED_NEW",
    )

    return response


def append_to_list(
    table_name: str, key: Dict[str, Any], list_name: str, values_to_append:
 List[Any]
) -> Dict[str, Any]:
    """
    Append values to the end of a list attribute.

    This function demonstrates how to use the list_append function to add
 elements
    to the end of a list attribute.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_append (List[Any]): The values to append to the list.
```

```
    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use list_append to add values to the end of the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = list_append({list_name}, :values)",
        ExpressionAttributeValues={":values": values_to_append},
        ReturnValues="UPDATED_NEW",
    )

    return response


def prepend_to_list(
    table_name: str, key: Dict[str, Any], list_name: str, values_to_prepend:
 List[Any]
) -> Dict[str, Any]:
    """
    Prepend values to the beginning of a list attribute.

    This function demonstrates how to use the list_append function to add
 elements
    to the beginning of a list attribute.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_prepend (List[Any]): The values to prepend to the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Use list_append with reversed order to add values to the beginning of the
 list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = list_append(:values, {list_name})",
        ExpressionAttributeValues={":values": values_to_prepend},
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_list_element(
    table_name: str, key: Dict[str, Any], list_name: str, index: int, new_value:
 Any
) -> Dict[str, Any]:
    """
    Update a specific element in a list attribute.

    This function demonstrates how to update a specific element in a list
 attribute
    using the index notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        index (int): The zero-based index of the element to update.
        new_value (Any): The new value for the element.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the index notation to update a specific element
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name}[{index}] = :value",
        ExpressionAttributeValues={":value": new_value},
```

```
        ReturnValues="UPDATED_NEW",
    )

    return response


def remove_list_element(
    table_name: str, key: Dict[str, Any], list_name: str, index: int
) -> Dict[str, Any]:
    """
    Remove a specific element from a list attribute.

    This function demonstrates how to remove a specific element from a list
 attribute
    using the REMOVE action with index notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        index (int): The zero-based index of the element to remove.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the REMOVE action with index notation to remove a specific element
    response = table.update_item(
        Key=key, UpdateExpression=f"REMOVE {list_name}[{index}]",
 ReturnValues="UPDATED_NEW"
    )

    return response


def update_nested_list_element(
    table_name: str, key: Dict[str, Any], path: str, new_value: Any
) -> Dict[str, Any]:
    """
    Update an element in a nested list structure.
```

```
    This function demonstrates how to update an element in a nested list
structure
    using expression attribute names for the path components.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path (str): The path to the nested element (e.g., "parent[0].child[1]").
        new_value (Any): The new value for the element.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Define a type for path parts
    path_part = Dict[str, Union[str, int]]
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Parse the path to extract attribute names and indices
    path_parts: List[path_part] = []
    current_part = ""
    in_bracket = False

    for char in path:
        if char == "[":
            if current_part:
                path_parts.append({"type": "attribute", "value": current_part})
                current_part = ""
            in_bracket = True
        elif char == "]":
            if current_part:
                # Fix for mypy: Use a properly typed dictionary with Union type
                path_parts.append({"type": "index", "value": int(current_part)})
                current_part = ""
            in_bracket = False
        elif char == "." and not in_bracket:
            if current_part:
                path_parts.append({"type": "attribute", "value": current_part})
                current_part = ""
        else:
            current_part += char
```

```
    if current_part:
        path_parts.append({"type": "attribute", "value": current_part})

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        if part["type"] == "attribute":
            name_placeholder = f"#attr{i}"
            expression_attribute_names[name_placeholder] = part["value"]

            if path_expression:
                path_expression += "."
            path_expression += name_placeholder
        elif part["type"] == "index":
            path_expression += f"[{part['value']}]"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": new_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def create_list_if_not_exists(
    table_name: str, key: Dict[str, Any], list_name: str, default_values:
 List[Any]
) -> Dict[str, Any]:
    """
    Create a list attribute if it doesn't exist.
```

```
    This function demonstrates how to use if_not_exists to create a list
 attribute
    with default values if it doesn't already exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        default_values (List[Any]): The default values for the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use if_not_exists to create the list if it doesn't exist
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} =
 if_not_exists({list_name}, :default)",
        ExpressionAttributeValues={":default": default_values},
        ReturnValues="UPDATED_NEW",
    )

    return response


def append_to_list_safely(
    table_name: str,
    key: Dict[str, Any],
    list_name: str,
    values_to_append: List[Any],
    default_values: Optional[List[Any]] = None,
) -> Dict[str, Any]:
    """
    Append values to a list, creating it if it doesn't exist.

    This function demonstrates how to safely append values to a list attribute,
    creating the list with default values if it doesn't exist.

    Args:
```

```
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_append (List[Any]): The values to append to the list.
        default_values (Optional[List[Any]]): The default values if the list
doesn't exist.
            If not provided, values_to_append will be used as the default.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # If default_values is not provided, use values_to_append
    if default_values is None:
        default_values = values_to_append

    # Use if_not_exists with list_append to safely append to the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} =
list_append(if_not_exists({list_name}, :default), :values)",
        ExpressionAttributeValues={
            ":default": default_values if default_values else [],
            ":values": values_to_append,
        },
        ReturnValues="UPDATED_NEW",
    )

    return response
```

AWS SDK for Python (Boto3)를 사용한 목록 작업의 사용 예입니다.

```
def example_usage():
    """Example of how to use list operations in DynamoDB."""
    # Example parameters
    table_name = "UserData"
```

```
    key = {"UserId": "user123"}

    print("Example 1: Creating a list attribute")
    try:
        response = create_list_attribute(
            table_name=table_name,
            key=key,
            list_name="Interests",
            list_values=["Reading", "Hiking", "Photography"],
        )
        print(
            f"List attribute created successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error creating list attribute: {e}")

    print("\nExample 2: Appending values to a list")
    try:
        response = append_to_list(
            table_name=table_name,
            key=key,
            list_name="Interests",
            values_to_append=["Cooking", "Gardening"],
        )
        print(
            f"Values appended to list successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error appending to list: {e}")

    print("\nExample 3: Prepending values to a list")
    try:
        response = prepend_to_list(
            table_name=table_name,
            key=key,
            list_name="Interests",
            values_to_prepend=["Travel", "Music"],
        )
        print(
            f"Values prepended to list successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
```

```
    except Exception as e:
        print(f"Error prepending to list: {e}")

    print("\nExample 4: Updating a specific list element")
    try:
        response = update_list_element(
            table_name=table_name,
            key=key,
            list_name="Interests",
            index=2,
            new_value="Mountain Hiking",
        )
        print(
            f"List element updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error updating list element: {e}")

    print("\nExample 5: Removing a list element")
    try:
        response = remove_list_element(
            table_name=table_name, key=key, list_name="Interests", index=0
        )
        print(
            f"List element removed successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error removing list element: {e}")

    print("\nExample 6: Working with nested lists")
    try:
        # First, create an item with a nested structure
        dynamodb = boto3.resource("dynamodb")
        table = dynamodb.Table(table_name)

        table.update_item(
            Key={"UserId": "user456"},
            UpdateExpression="SET #skills = :skills",
            ExpressionAttributeNames={"#skills": "Skills"},
            ExpressionAttributeValues={
                ":skills": [
```

```
                    {"Category": "Programming", "Languages": ["Python", "Java",
"JavaScript"]},
                    {"Category": "Database", "Systems": ["DynamoDB", "MongoDB",
"PostgreSQL"]},
                ]
            },
        )

        # Now update a nested element
        response = update_nested_list_element(
            table_name=table_name,
            key={"UserId": "user456"},
            path="Skills[0].Languages[1]",
            new_value="TypeScript",
        )
        print(
            f"Nested list element updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error working with nested lists: {e}")

    print("\nExample 7: Creating a list if it doesn't exist")
    try:
        response = create_list_if_not_exists(
            table_name=table_name,
            key={"UserId": "user789"},
            list_name="Preferences",
            default_values=["Default1", "Default2", "Default3"],
        )
        print(
            f"List created with default values:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error creating list with default values: {e}")

    print("\nExample 8: Safely appending to a list")
    try:
        response = append_to_list_safely(
            table_name=table_name,
            key={"UserId": "user789"},
            list_name="Notifications",
            values_to_append=["New message received"],
```

```
            default_values=[],
        )
        print(f"Safely appended to list: {json.dumps(response.get('Attributes',
{}), default=str)}")
    except Exception as e:
        print(f"Error safely appending to list: {e}")

    print("\nKey Points About Working with Lists in DynamoDB:")
    print("1. Lists are ordered collections of elements that can be of different
types")
    print("2. Use the SET operation with direct assignment to create or replace a
list")
    print("3. Use list_append() to add elements to a list without replacing the
entire list")
    print("4. To append to the end: list_append(list_name, :values)")
    print("5. To prepend to the beginning: list_append(:values, list_name)")
    print("6. Use index notation list_name[index] to access or update specific
elements")
    print("7. Use the REMOVE action with index notation to remove specific
elements")
    print("8. Lists can contain nested structures like maps and other lists")
    print("9. Use if_not_exists() to create a list with default values if it
doesn't exist")
    print("10. List indices are zero-based (the first element is at index 0)")
    print("11. Attempting to access an index beyond the list bounds will result
in an error")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 DynamoDB에서 맵 작업 수행

다음 코드 예제에서는 DynamoDB에서 맵 작업을 수행하는 방법을 보여줍니다.

- 맵 구조에서 중첩 속성을 추가하고 업데이트합니다.
- 맵에서 특정 필드를 제거합니다.
- 깊이 중첩된 맵 속성을 사용하여 작업합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용한 맵 작업을 보여줍니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Updates a map attribute that may not exist.
     *
     * <p>This method demonstrates how to safely update a map attribute
     * by using if_not_exists to handle the case where the map doesn't exist yet.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param mapKey The key within the map to update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateMapAttributeSafe(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        String mapKey,
        AttributeValue value) {

        // Create an empty map to use if the map doesn't exist
        Map<String, AttributeValue> emptyMap = new HashMap<>();
```

```
        AttributeValue emptyMapValue =
AttributeValue.builder().m(emptyMap).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #mapName = if_not_exists(#mapName, :emptyMap),
#mapName.#mapKey = :value")
            .expressionAttributeNames(Map.of(
                "#mapName", mapName,
                "#mapKey", mapKey))
            .expressionAttributeValues(Map.of(
                ":value",
                value,
                ":emptyMap",
                AttributeValue.builder().m(new HashMap<>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Adds an attribute to a nested map.
     *
     * <p>This method demonstrates how to update a nested attribute without
     * overwriting the entire map.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param path The path to the nested attribute as a list
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToNestedMap(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        List<String> path,
        AttributeValue value) {
```

```
        // Create expression attribute names for each part of the path
        Map<String, String> expressionAttributeNames = new HashMap<>();
        for (int i = 0; i < path.size(); i++) {
            expressionAttributeNames.put("#attr" + i, path.get(i));
        }

        // Build the attribute path using the expression attribute names
        StringBuilder attributePathExpression = new StringBuilder();
        for (int i = 0; i < path.size(); i++) {
            if (i > 0) {
                attributePathExpression.append(".");
            }
            attributePathExpression.append("#attr").append(i);
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET " + attributePathExpression.toString() + "
= :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes an attribute from a map.
     *
     * <p>This method demonstrates how to remove a specific attribute from a map.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param mapKey The key within the map to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
```

```
    public static UpdateItemResponse removeMapAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        String mapKey) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #mapName.#mapKey")
            .expressionAttributeNames(Map.of(
                "#mapName", mapName,
                "#mapKey", mapKey))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Creates a map with multiple attributes in a single operation.
     *
     * <p>This method demonstrates how to create a map with multiple attributes
     * in a single update operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param attributes The attributes to set in the map
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse createMapWithAttributes(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        Map<String, AttributeValue> attributes) {

        // Create a map value from the attributes
```

```
        AttributeValue mapValue = AttributeValue.builder().m(attributes).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #mapName = :mapValue")
            .expressionAttributeNames(Map.of("#mapName", mapName))
            .expressionAttributeValues(Map.of(":mapValue", mapValue))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a map attribute.
     *
     * <p>Helper method to retrieve the current value of a map attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param mapName The name of the map attribute
     * @return The map attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, AttributeValue> getMapAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String mapName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(mapName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the map attribute if it exists, otherwise null
```

```
            if (response.item() != null && response.item().containsKey(mapName))
{
                return response.item().get(mapName).m();
            }

            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get map attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용한 맵 작업의 사용 예입니다.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating map operations in DynamoDB");

        try {
            // Example 1: Create a map with multiple attributes
            System.out.println("\nExample 1: Creating a map with multiple
attributes");
            Map<String, AttributeValue> productDetails = new HashMap<>();
            productDetails.put("Color",
AttributeValue.builder().s("Red").build());
            productDetails.put("Weight",
AttributeValue.builder().n("2.5").build());
            productDetails.put(
                "Dimensions", AttributeValue.builder().s("10x20x5").build());

            UpdateItemResponse createResponse =
                createMapWithAttributes(dynamoDbClient, tableName, key,
"Details", productDetails);

            System.out.println("Created map attribute: " +
createResponse.attributes());
```

```
            // Example 2: Update a specific attribute in a map
            System.out.println("\nExample 2: Updating a specific attribute in a
map");
            UpdateItemResponse updateResponse = updateMapAttributeSafe(
                dynamoDbClient,
                tableName,
                key,
                "Details",
                "Color",
                AttributeValue.builder().s("Blue").build());

            System.out.println("Updated map attribute: " +
updateResponse.attributes());

            // Example 3: Add an attribute to a nested map
            System.out.println("\nExample 3: Adding an attribute to a nested
map");
            UpdateItemResponse nestedResponse = addToNestedMap(
                dynamoDbClient,
                tableName,
                key,
                List.of("Specifications", "Technical", "Resolution"),
                AttributeValue.builder().s("1920x1080").build());

            System.out.println("Added to nested map: " +
nestedResponse.attributes());

            // Example 4: Remove an attribute from a map
            System.out.println("\nExample 4: Removing an attribute from a map");
            UpdateItemResponse removeResponse =
                removeMapAttribute(dynamoDbClient, tableName, key, "Details",
"Dimensions");

            System.out.println("Updated map after removal: " +
removeResponse.attributes());

            // Example 5: Get the current value of a map attribute
            System.out.println("\nExample 5: Getting the current value of a map
attribute");
            Map<String, AttributeValue> currentMap =
getMapAttribute(dynamoDbClient, tableName, key, "Details");

            if (currentMap != null) {
```

```
                System.out.println("Current map attribute:");
                for (Map.Entry<String, AttributeValue> entry :
currentMap.entrySet()) {
                    System.out.println("  " + entry.getKey() + ": " +
entry.getValue());
                }
            } else {
                System.out.println("Map attribute not found");
            }

            // Explain map operations
            System.out.println("\nKey points about DynamoDB map operations:");
            System.out.println("1. Maps are unordered collections of name-value
pairs");
            System.out.println("2. Use dot notation (map.key) to access or update
specific attributes");
            System.out.println("3. You can update individual attributes without
overwriting the entire map");
            System.out.println("4. Maps can be nested to create complex data
structures");
            System.out.println("5. Use REMOVE to delete attributes from a map");
            System.out.println("6. You can create a map with multiple attributes
in a single operation");
            System.out.println("7. Map keys are case-sensitive");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용한 맵 작업을 보여줍니다.

```
/**
 * Example of updating map attributes in DynamoDB.
```

```
 *
 * This module demonstrates how to update map attributes that may not exist,
 * how to update nested attributes, and how to handle various map update
 scenarios.
 */

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update a map attribute safely, handling the case where the map might not
 exist.
 *
 * This function demonstrates using the if_not_exists function to safely update
 * a map attribute that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeSafe(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
```

```
    UpdateExpression: `SET ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return response;
  } catch (error) {
    // If the error is because the map doesn't exist, create it
    if (error.name === "ValidationException" &&
        error.message.includes("The document path provided in the update
 expression is invalid")) {

      // Create the map with the specified key-value pair
      const createParams = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${mapName} = :map`,
        ExpressionAttributeValues: {
          ":map": { [mapKey]: value }
        },
        ReturnValues: "UPDATED_NEW"
      };

      return await docClient.send(new UpdateCommand(createParams));
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Update a map attribute using the if_not_exists function.
 *
 * This function demonstrates a more elegant approach using if_not_exists
 * to handle the case where the map doesn't exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
```

```
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeWithIfNotExists(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName} = if_not_exists(${mapName}, :emptyMap),
 ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":emptyMap": {},
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Add a value to a deeply nested map, creating parent maps if they don't exist.
 *
 * This function demonstrates how to update a deeply nested attribute,
 * creating any parent maps that don't exist along the way.
 *
 * @param {Object} config - AWS configuration object
```

```
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} path - The path to the nested attribute as an array of keys
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToNestedMap(
  config,
  tableName,
  key,
  path,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET";
  const expressionAttributeValues = {};

  // For each level in the path, create a map if it doesn't exist
  for (let i = 0; i < path.length; i++) {
    const currentPath = path.slice(0, i + 1).join(".");
    const parentPath = i > 0 ? path.slice(0, i).join(".") : null;

    if (parentPath) {
      updateExpression += ` ${parentPath} =
 if_not_exists(${parentPath}, :emptyMap${i}),`;
      expressionAttributeValues[`:emptyMap${i}`] = {};
    }
  }

  // Set the final value
  const fullPath = path.join(".");
  updateExpression += ` ${fullPath} = :value`;
  expressionAttributeValues[":value"] = value;

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
```

```
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update multiple fields in a map attribute in a single operation.
 *
 * This function demonstrates how to update multiple fields in a map
 * in a single DynamoDB operation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {Object} updates - Object containing key-value pairs to update
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMultipleMapFields(
  config,
  tableName,
  key,
  mapName,
  updates
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = `SET ${mapName} = if_not_exists(${mapName}, :emptyMap)`;
  const expressionAttributeValues = {
    ":emptyMap": {}
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([field, value], index) => {
    updateExpression += `, ${mapName}.${field} = :val${index}`;
    expressionAttributeValues[`:val${index}`] = value;
  });
```

```
  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));
```

```
  // Return the item if it exists, otherwise null
  return response.Item || null;
}

/**
 * Example of how to use the map attribute update functions.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating different approaches to update map attributes in
 DynamoDB");

  try {
    // Example 1: Update a map attribute that might not exist (two-step approach)
    console.log("\nExample 1: Updating a map attribute that might not exist (two-
step approach)");
    const response1 = await updateMapAttributeSafe(
      config,
      tableName,
      key,
      "Preferences",
      "Theme",
      "Dark"
    );

    console.log("Updated preferences:", response1.Attributes);

    // Example 2: Update a map attribute using if_not_exists (elegant approach)
    console.log("\nExample 2: Updating a map attribute using if_not_exists
 (elegant approach)");
    const response2 = await updateMapAttributeWithIfNotExists(
      config,
      tableName,
      key,
      "Settings",
      "NotificationsEnabled",
      true
    );

    console.log("Updated settings:", response2.Attributes);
```

```
    // Example 3: Update a deeply nested attribute
    console.log("\nExample 3: Updating a deeply nested attribute");
    const response3 = await addToNestedMap(
      config,
      tableName,
      key,
      ["Profile", "Address", "City"],
      "Seattle"
    );

    console.log("Updated nested attribute:", response3.Attributes);

    // Example 4: Update multiple fields in a map
    console.log("\nExample 4: Updating multiple fields in a map");
    const response4 = await updateMultipleMapFields(
      config,
      tableName,
      key,
      "ContactInfo",
      {
        Email: "user@example.com",
        Phone: "555-123-4567",
        PreferredContact: "Email"
      }
    );

    console.log("Updated multiple fields:", response4.Attributes);

    // Get the final state of the item
    console.log("\nFinal state of the item:");
    const item = await getItem(config, tableName, key);
    console.log(JSON.stringify(item, null, 2));

    // Explain the benefits of different approaches
    console.log("\nKey points about updating map attributes:");
    console.log("1. Use if_not_exists to handle maps that might not exist");
    console.log("2. Multiple updates can be combined in a single operation");
    console.log("3. Deeply nested attributes require creating parent maps");
    console.log("4. DynamoDB expressions are atomic - the entire update succeeds
or fails");
    console.log("5. Using a single operation is more efficient than multiple
separate updates");
```

```
  } catch (error) {
    console.error("Error:", error);
  }
}

// Export the functions
module.exports = {
  updateMapAttributeSafe,
  updateMapAttributeWithIfNotExists,
  addToNestedMap,
  updateMultipleMapFields,
  getItem,
  exampleUsage
};

// Run the example if this file is executed directly
if (require.main === module) {
  exampleUsage();
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용한 맵 작업을 보여줍니다.

```
"""
Example of updating map attributes in DynamoDB.

This module demonstrates how to update map attributes in DynamoDB, including
handling cases where the map attribute might not exist yet.
"""


import boto3
from typing import Any, Dict, Optional
```

```
def update_map_attribute_safe(
    table_name: str, key: Dict[str, Any], map_name: str, map_key: str, value: Any
) -> Dict[str, Any]:
    """
    Update a specific key in a map attribute, creating the map if it doesn't
 exist.

    This function demonstrates how to safely update a key within a map attribute,
    even if the map doesn't exist yet in the item.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        map_key (str): The key within the map to update.
        value (Any): The value to set for the map key.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use SET with attribute_not_exists to safely update the map
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #map.#key = :value",
        ExpressionAttributeNames={"#map": map_name, "#key": map_key},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def add_to_nested_map(
    table_name: str, key: Dict[str, Any], path: str, value: Any
) -> Dict[str, Any]:
    """
    Add or update a value in a deeply nested map structure.

    This function demonstrates how to update a value at a specific path in a
```

```
    nested map structure, creating any intermediate maps as needed.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path (str): The path to the nested attribute (e.g.,
"user.preferences.theme").
        value (Any): The value to set at the specified path.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Split the path into components
    path_parts = path.split(".")

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        name_placeholder = f"#attr{i}"
        expression_attribute_names[name_placeholder] = part

        if i == 0:
            path_expression = name_placeholder
        else:
            path_expression += f".{name_placeholder}"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": value},
```

```
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_map_with_if_not_exists(
    table_name: str,
    key: Dict[str, Any],
    map_name: str,
    map_key: str,
    value: Any,
    default_map: Optional[Dict[str, Any]] = None,
) -> Dict[str, Any]:
    """
    Update a key in a map, creating the map with default values if it doesn't
 exist.

    This function demonstrates how to use if_not_exists to initialize a map with
    default values if it doesn't exist yet, and then update a specific key.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        map_key (str): The key within the map to update.
        value (Any): The value to set for the map key.
        default_map (Optional[Dict[str, Any]]): Default map values if the map
 doesn't exist.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Set default map if not provided
    if default_map is None:
        default_map = {}

    # Create a map with the new key-value pair
    updated_map = default_map.copy()
```

```
    updated_map[map_key] = value

    # Use if_not_exists to initialize the map if it doesn't exist
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #map = if_not_exists(#map, :default_map)",
        ExpressionAttributeNames={"#map": map_name},
        ExpressionAttributeValues={":default_map": updated_map},
        ReturnValues="UPDATED_NEW",
    )

    return response


def merge_into_map(
    table_name: str, key: Dict[str, Any], map_name: str, values_to_merge:
 Dict[str, Any]
) -> Dict[str, Any]:
    """
    Merge multiple key-value pairs into a map attribute.

    This function demonstrates how to update multiple keys in a map attribute
    in a single operation, without overwriting the entire map.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        values_to_merge (Dict[str, Any]): Key-value pairs to merge into the map.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the update expression for each key-value pair
    update_expression = "SET "
    expression_attribute_names = {"#map": map_name}
    expression_attribute_values = {}

    # Add each key-value pair to the update expression
```

```
    for i, (k, v) in enumerate(values_to_merge.items()):
        key_placeholder = f"#key{i}"
        value_placeholder = f":value{i}"

        expression_attribute_names[key_placeholder] = k
        expression_attribute_values[value_placeholder] = v

        if i > 0:
            update_expression += ", "
        update_expression += f"#map.{key_placeholder} = {value_placeholder}"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues=expression_attribute_values,
        ReturnValues="UPDATED_NEW",
    )

    return response




def example_usage():
    """Example of how to use the map attribute update functions."""
    # Example parameters
    table_name = "UserProfiles"
    key = {"UserId": "user123"}

    print("Example 1: Updating a specific key in a map attribute")
    try:
        response = update_map_attribute_safe(
            table_name=table_name, key=key, map_name="Preferences",
 map_key="Theme", value="Dark"
        )
        print(f"Map attribute updated successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error updating map attribute: {e}")

    print("\nExample 2: Adding a value to a deeply nested map")
    try:
```

```
        response = add_to_nested_map(
            table_name=table_name, key=key, path="Settings.Notifications.Email",
value=True
        )
        print(f"Nested map updated successfully: {response.get('Attributes',
{})}")
    except Exception as e:
        print(f"Error updating nested map: {e}")

    print("\nExample 3: Initializing a map with default values if it doesn't
exist")
    try:
        default_map = {"Language": "English", "Currency": "USD", "TimeZone":
"UTC"}

        response = update_map_with_if_not_exists(
            table_name=table_name,
            key={"UserId": "newuser456"},
            map_name="Preferences",
            map_key="Theme",
            value="Light",
            default_map=default_map,
        )
        print(f"Map initialized with defaults: {response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error initializing map: {e}")

    print("\nExample 4: Merging multiple values into a map")
    try:
        values_to_merge = {
            "NotificationsEnabled": True,
            "EmailFrequency": "Daily",
            "PushNotifications": False,
        }

        response = merge_into_map(
            table_name=table_name,
            key=key,
            map_name="NotificationSettings",
            values_to_merge=values_to_merge,
        )
        print(f"Multiple values merged into map: {response.get('Attributes',
{})}")
    except Exception as e:
```

```
        print(f"Error merging values into map: {e}")

    print("\nBest practices for working with map attributes in DynamoDB:")
    print("1. Use dot notation to access and update nested attributes")
    print("2. Use ExpressionAttributeNames to handle reserved words and special
characters")
    print("3. Use if_not_exists() to handle cases where attributes might not
exist")
    print("4. Update specific map keys rather than overwriting the entire map")
    print("5. Use a single update operation to modify multiple map keys for
better performance")
    print("6. Consider your data model carefully to minimize the need for deeply
nested attributes")



if __name__ == "__main__":
    example_usage()
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 DynamoDB에서 집합 작업 수행

다음 코드 예제에서는 DynamoDB에서 집합 작업을 수행하는 방법을 보여줍니다.

- 집합 속성에 요소를 추가합니다.
- 집합 속성에서 요소를 제거합니다.
- 집합에 ADD 및 DELETE 작업을 사용합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용한 집합 작업을 보여줍니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

    /**
     * Adds values to a string set attribute.
     *
     * <p>This method demonstrates how to use the ADD operation to add values
     * to a string set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToAdd The values to add to the set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToStringSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<String> valuesToAdd) {

        // Create a string set value from the values to add
        AttributeValue setValue =
 AttributeValue.builder().ss(valuesToAdd).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #setAttr :valuesToAdd")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToAdd", setValue))
```

```
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Adds values to a number set attribute.
     *
     * <p>This method demonstrates how to use the ADD operation to add values
     * to a number set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToAdd The values to add to the set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToNumberSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<Number> valuesToAdd) {

        // Convert numbers to strings for DynamoDB
        Set<String> stringValues = new HashSet<>();
        for (Number value : valuesToAdd) {
            stringValues.add(value.toString());
        }

        // Create a number set value from the values to add
        AttributeValue setValue =
AttributeValue.builder().ns(stringValues).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #setAttr :valuesToAdd")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
```

```
            .expressionAttributeValues(Map.of(":valuesToAdd", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes values from a set attribute.
     *
     * <p>This method demonstrates how to use the DELETE operation to remove
values
     * from a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToRemove The values to remove from the set
     * @param isNumberSet Whether the set is a number set (true) or string set
(false)
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeFromSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<?> valuesToRemove,
        boolean isNumberSet) {

        AttributeValue setValue;

        if (isNumberSet) {
            // Convert numbers to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : valuesToRemove) {
                if (value instanceof Number) {
                    stringValues.add(value.toString());
                } else {
                    throw new IllegalArgumentException("Values must be numbers
for a number set");
```

```
                }
            }

            setValue = AttributeValue.builder().ns(stringValues).build();
        } else {
            // Convert objects to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : valuesToRemove) {
                stringValues.add(value.toString());
            }

            setValue = AttributeValue.builder().ss(stringValues).build();
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("DELETE #setAttr :valuesToRemove")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToRemove", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Checks if a value exists in a set attribute.
     *
     * <p>This method demonstrates how to use the contains function to check
     * if a value exists in a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to check
     * @param setAttributeName The name of the set attribute
     * @param valueToCheck The value to check for
     * @return Map containing the result of the check
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, Object> checkIfValueInSet(
        DynamoDbClient dynamoDbClient,
```

```
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        String valueToCheck) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the update parameters with a condition expression
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #tempAttr = :tempVal")
                .conditionExpression("contains(#setAttr, :valueToCheck)")
                .expressionAttributeNames(Map.of("#setAttr", setAttributeName,
"#tempAttr", "TempAttribute"))
                .expressionAttributeValues(Map.of(
                    ":valueToCheck",
AttributeValue.builder().s(valueToCheck).build(),
                    ":tempVal", AttributeValue.builder().s("TempValue").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Attempt the update operation
            dynamoDbClient.updateItem(request);

            // If we get here, the condition was met
            result.put("exists", true);
            result.put("message", "Value '" + valueToCheck + "' exists in the
set");

            // Clean up the temporary attribute
            UpdateItemRequest cleanupRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("REMOVE #tempAttr")
                .expressionAttributeNames(Map.of("#tempAttr", "TempAttribute"))
                .build();

            dynamoDbClient.updateItem(cleanupRequest);

        } catch (DynamoDbException e) {
            if (e.getMessage().contains("ConditionalCheckFailed")) {
                // The condition was not met
```

```
                result.put("exists", false);
                result.put("message", "Value '" + valueToCheck + "' does not
exist in the set");
            } else {
                // Some other error occurred
                result.put("exists", false);
                result.put("message", "Error checking set: " + e.getMessage());
                result.put("error", e.getClass().getSimpleName());
            }
        }

        return result;
    }

    /**
     * Creates a set with multiple values in a single operation.
     *
     * <p>This method demonstrates how to create a set with multiple values
     * in a single update operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param setValues The values to include in the set
     * @param isNumberSet Whether to create a number set (true) or string set
(false)
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse createSetWithValues(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<?> setValues,
        boolean isNumberSet) {

        AttributeValue setValue;

        if (isNumberSet) {
            // Convert numbers to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : setValues) {
```

```
                if (value instanceof Number) {
                    stringValues.add(value.toString());
                } else {
                    throw new IllegalArgumentException("Values must be numbers
for a number set");
                }
            }

            setValue = AttributeValue.builder().ns(stringValues).build();
        } else {
            // Convert objects to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : setValues) {
                stringValues.add(value.toString());
            }

            setValue = AttributeValue.builder().ss(stringValues).build();
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #setAttr = :setValue")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a set attribute.
     *
     * <p>Helper method to retrieve the current value of a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param setAttributeName The name of the set attribute
     * @return The set attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
```

```
     */
    public static AttributeValue getSetAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String setAttributeName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(setAttributeName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the set attribute if it exists, otherwise null
            if (response.item() != null &&
response.item().containsKey(setAttributeName)) {
                return response.item().get(setAttributeName);
            }

            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get set attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용한 집합 작업의 사용 예입니다.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating set operations in DynamoDB");

        try {
```

```
            // Example 1: Create a string set with multiple values
            System.out.println("\nExample 1: Creating a string set with multiple
values");
            Set<String> tags = new HashSet<>();
            tags.add("Electronics");
            tags.add("Gadget");
            tags.add("Smartphone");

            UpdateItemResponse createResponse = createSetWithValues(
                dynamoDbClient, tableName, key, "Tags", tags, false // Not a
number set
                );

            System.out.println("Created set attribute: " +
createResponse.attributes());

            // Example 2: Add values to a string set
            System.out.println("\nExample 2: Adding values to a string set");
            Set<String> additionalTags = new HashSet<>();
            additionalTags.add("Mobile");
            additionalTags.add("Wireless");

            UpdateItemResponse addResponse = addToStringSet(dynamoDbClient,
tableName, key, "Tags", additionalTags);

            System.out.println("Updated set attribute: " +
addResponse.attributes());

            // Example 3: Create a number set with multiple values
            System.out.println("\nExample 3: Creating a number set with multiple
values");
            Set<Number> ratings = new HashSet<>();
            ratings.add(4);
            ratings.add(5);
            ratings.add(4.5);

            UpdateItemResponse createNumberSetResponse = createSetWithValues(
                dynamoDbClient, tableName, key, "Ratings", ratings, true // Is a
number set
                );

            System.out.println("Created number set attribute: " +
createNumberSetResponse.attributes());
```

```
            // Example 4: Add values to a number set
            System.out.println("\nExample 4: Adding values to a number set");
            Set<Number> additionalRatings = new HashSet<>();
            additionalRatings.add(3.5);
            additionalRatings.add(4.2);

            UpdateItemResponse addNumberResponse =
                addToNumberSet(dynamoDbClient, tableName, key, "Ratings",
additionalRatings);

            System.out.println("Updated number set attribute: " +
addNumberResponse.attributes());

            // Example 5: Remove values from a set
            System.out.println("\nExample 5: Removing values from a set");
            Set<String> tagsToRemove = new HashSet<>();
            tagsToRemove.add("Gadget");

            UpdateItemResponse removeResponse = removeFromSet(
                dynamoDbClient, tableName, key, "Tags", tagsToRemove, false //
Not a number set
                );

            System.out.println("Updated set after removal: " +
removeResponse.attributes());

            // Example 6: Check if a value exists in a set
            System.out.println("\nExample 6: Checking if a value exists in a
set");
            Map<String, Object> checkResult = checkIfValueInSet(dynamoDbClient,
tableName, key, "Tags", "Electronics");

            System.out.println("Check result: " + checkResult.get("message"));

            // Example 7: Get the current value of a set attribute
            System.out.println("\nExample 7: Getting the current value of a set
attribute");
            AttributeValue currentStringSet = getSetAttribute(dynamoDbClient,
tableName, key, "Tags");

            if (currentStringSet != null && currentStringSet.ss() != null) {
                System.out.println("Current string set values: " +
currentStringSet.ss());
            } else {
```

```
                System.out.println("String set attribute not found");
            }

            AttributeValue currentNumberSet = getSetAttribute(dynamoDbClient,
tableName, key, "Ratings");

            if (currentNumberSet != null && currentNumberSet.ns() != null) {
                System.out.println("Current number set values: " +
currentNumberSet.ns());
            } else {
                System.out.println("Number set attribute not found");
            }

            // Explain set operations
            System.out.println("\nKey points about DynamoDB set operations:");
            System.out.println(
                "1. DynamoDB supports three set types: string sets (SS), number
sets (NS), and binary sets (BS)");
            System.out.println("2. Sets can only contain elements of the same
type");
            System.out.println("3. Use ADD to add elements to a set");
            System.out.println("4. Use DELETE to remove elements from a set");
            System.out.println("5. Sets automatically remove duplicate values");
            System.out.println("6. Sets are unordered collections");
            System.out.println("7. Use the contains function to check if a value
exists in a set");
            System.out.println("8. You can create a set with multiple values in a
single operation");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용한 집합 작업을 보여줍니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Add elements to a set attribute.
 *
 * This function demonstrates using the ADD operation to add elements to a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to add to the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }
```

```
  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove elements from a set attribute.
 *
 * This function demonstrates using the DELETE operation to remove elements from
 a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to remove from the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeFromSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using DELETE
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `DELETE ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a new set attribute with initial values.
 *
 * This function demonstrates using the SET operation to create a new set
 attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The initial values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createSet(
```

```
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using SET
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${setName} = :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Replace an entire set attribute with a new set of values.
 *
 * This function demonstrates using the SET operation to replace an entire set.
```

```
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The new values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function replaceSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // This is the same as createSet, but included for clarity of intent
  return await createSet(config, tableName, key, setName, values, setType);
}

/**
 * Remove the last element from a set and handle the empty set case.
 *
 * This function demonstrates what happens when you delete the last element of a
 set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @returns {Promise<Object>} - The result of the operation
 */
async function removeLastElementFromSet(
  config,
  tableName,
  key,
  setName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // First, get the current item to check the set
```

```
  const currentItem = await getItem(config, tableName, key);

  // Check if the set exists and has elements
  if (!currentItem || !currentItem[setName] || currentItem[setName].size === 0) {
    return {
      success: false,
      message: "Set doesn't exist or is already empty",
      item: currentItem
    };
  }

  // Get the set values
  const setValues = Array.from(currentItem[setName]);

  // If there's only one element left, remove the attribute entirely
  if (setValues.length === 1) {
    // Define the update parameters to remove the attribute
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: `REMOVE ${setName}`,
      ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Last element removed, attribute has been deleted",
      removedValue: setValues[0]
    };
  } else {
    // Otherwise, remove just the last element
    // Create a set with just the last element
    const lastElement = setValues[setValues.length - 1];
    const setType = typeof lastElement === 'number' ? 'number' : 'string';

    // Remove the last element
    const response = await removeFromSet(
      config,
      tableName,
      key,
      setName,
```

```
      [lastElement],
      setType
    );

    return {
      success: true,
      message: "Last element removed, set still contains elements",
      removedValue: lastElement,
      remainingSet: response.Attributes[setName]
    };
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

AWS SDK for JavaScript를 사용한 집합 작업의 사용 예입니다.

```
/**
 * Example of how to work with sets in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating set operations in DynamoDB");

  try {
    // Example 1: Create a string set
    console.log("\nExample 1: Creating a string set");
    const response1 = await createSet(
      config,
      tableName,
      key,
      "Interests",
      ["Reading", "Hiking", "Cooking"],
      "string"
    );

    console.log("Created set:", response1.Attributes);

    // Example 2: Add elements to a set
    console.log("\nExample 2: Adding elements to a set");
    const response2 = await addToSet(
      config,
      tableName,
      key,
      "Interests",
      ["Photography", "Travel"],
      "string"
    );

    console.log("Updated set after adding elements:", response2.Attributes);

    // Example 3: Remove elements from a set
```

```
    console.log("\nExample 3: Removing elements from a set");
    const response3 = await removeFromSet(
      config,
      tableName,
      key,
      "Interests",
      ["Cooking"],
      "string"
    );

    console.log("Updated set after removing elements:", response3.Attributes);

    // Example 4: Create a number set
    console.log("\nExample 4: Creating a number set");
    const response4 = await createSet(
      config,
      tableName,
      key,
      "FavoriteNumbers",
      [7, 42, 99],
      "number"
    );

    console.log("Created number set:", response4.Attributes);

    // Example 5: Replace an entire set
    console.log("\nExample 5: Replacing an entire set");
    const response5 = await replaceSet(
      config,
      tableName,
      key,
      "Interests",
      ["Gaming", "Movies", "Music"],
      "string"
    );

    console.log("Replaced set:", response5.Attributes);

    // Example 6: Remove the last element from a set
    console.log("\nExample 6: Removing the last element from a set");

    // First, create a set with just one element
    await createSet(
      config,
```

```
      tableName,
      { UserId: "U67890" },
      "Tags",
      ["LastTag"],
      "string"
    );

    // Then, remove the last element
    const response6 = await removeLastElementFromSet(
      config,
      tableName,
      { UserId: "U67890" },
      "Tags"
    );

    console.log(response6.message);
    console.log("Removed value:", response6.removedValue);

    // Get the final state of the items
    console.log("\nFinal state of the items:");
    const item1 = await getItem(config, tableName, key);
    console.log("User U12345:", JSON.stringify(item1, null, 2));

    const item2 = await getItem(config, tableName, { UserId: "U67890" });
    console.log("User U67890:", JSON.stringify(item2, null, 2));

    // Explain set operations
    console.log("\nKey points about set operations in DynamoDB:");
    console.log("1. Use ADD to add elements to a set (duplicates are
 automatically removed)");
    console.log("2. Use DELETE to remove elements from a set");
    console.log("3. Use SET to create a new set or replace an existing one");
    console.log("4. DynamoDB supports three types of sets: string sets, number
 sets, and binary sets");
    console.log("5. When you delete the last element from a set, the attribute
 remains as an empty set");
    console.log("6. To remove an empty set, use the REMOVE operation");
    console.log("7. Sets automatically maintain unique values (no duplicates)");
    console.log("8. You cannot mix data types within a set");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용한 집합 작업을 보여줍니다.

```
import boto3
from typing import Any, Dict, List


def create_set_attribute(
    table_name: str,
    key: Dict[str, Any],
    set_name: str,
    set_values: List[Any],
    set_type: str = "string",
) -> Dict[str, Any]:
    """
    Create a new set attribute or add elements to an existing set.

    This function demonstrates how to use the ADD operation to create a new set
    or add elements to an existing set.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        set_values (List[Any]): The values to add to the set.
        set_type (str, optional): The type of set to create: "string", "number",
 or "binary".
            Defaults to "string".

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Convert the list to a DynamoDB set based on the specified type
    if set_type == "string":
        dynamo_set = set(str(value) for value in set_values)
    elif set_type == "number":
        # We need to use actual float values for the DynamoDB API
        # but mypy expects strings in sets, so we need to use type: ignore
        dynamo_set = set(float(value) for value in set_values)  # type: ignore
    else:  # binary set is not directly supported in high-level API, handled
 differently
        raise ValueError("Binary sets are not supported in this example")

    # Use the ADD operation to create or update the set
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #set_attr :set_values",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":set_values": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response


def add_to_set(
    table_name: str, key: Dict[str, Any], set_name: str, values_to_add: List[Any]
) -> Dict[str, Any]:
    """
    Add elements to an existing set attribute.

    This function demonstrates how to use the ADD operation to add elements to an
 existing set.
    If the set doesn't exist, it will be created.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        values_to_add (List[Any]): The values to add to the set.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
```

```
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Convert the list to a set (assuming string set for simplicity)
    dynamo_set = set(str(value) for value in values_to_add)

    # Use the ADD operation to add values to the set
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #set_attr :values_to_add",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":values_to_add": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response


def remove_from_set(
    table_name: str, key: Dict[str, Any], set_name: str, values_to_remove:
 List[Any]
) -> Dict[str, Any]:
    """
    Remove elements from a set attribute.

    This function demonstrates how to use the DELETE operation to remove elements
 from a set.
    If the last element is removed, the attribute will be deleted entirely.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        values_to_remove (List[Any]): The values to remove from the set.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Convert the list to a set (assuming string set for simplicity)
    dynamo_set = set(str(value) for value in values_to_remove)

    # Use the DELETE operation to remove values from the set
    response = table.update_item(
        Key=key,
        UpdateExpression="DELETE #set_attr :values_to_remove",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":values_to_remove": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response


def check_if_set_exists(table_name: str, key: Dict[str, Any], set_name: str) ->
 bool:
    """
    Check if a set attribute exists in an item.

    This function demonstrates how to check if a set attribute exists after
    potentially removing all elements from it.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to check.
        set_name (str): The name of the set attribute.

    Returns:
        bool: True if the set attribute exists, False otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Get the item
    response = table.get_item(
        Key=key, ProjectionExpression="#set_attr",
 ExpressionAttributeNames={"#set_attr": set_name}
    )

    # Check if the item exists and has the set attribute
    return "Item" in response and set_name in response["Item"]
```

```
def demonstrate_last_element_removal(
    table_name: str, key: Dict[str, Any], set_name: str
) -> Dict[str, Any]:
    """
    Demonstrate what happens when you remove the last element from a set.

    This function creates a set with a single element, then removes that element,
    showing that the attribute is completely removed when the last element is
 deleted.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Step 1: Create a set with a single element
    create_response = create_set_attribute(
        table_name=table_name,
        key=key,
        set_name=set_name,
        set_values=["last_element"],
        set_type="string",
    )

    # Step 2: Check that the set exists
    exists_before = check_if_set_exists(table_name, key, set_name)

    # Step 3: Remove the last element
    delete_response = remove_from_set(
        table_name=table_name, key=key, set_name=set_name,
 values_to_remove=["last_element"]
    )

    # Step 4: Check if the set still exists
    exists_after = check_if_set_exists(table_name, key, set_name)

    # Return the results
    return {
        "create_response": create_response,
        "exists_before": exists_before,
```

```
        "delete_response": delete_response,
        "exists_after": exists_after,
    }


def work_with_number_set(
    table_name: str,
    key: Dict[str, Any],
    set_name: str,
    initial_values: List[float],
    values_to_add: List[float],
    values_to_remove: List[float],
) -> Dict[str, Any]:
    """
    Demonstrate working with a number set in DynamoDB.

    This function shows how to create and manipulate a set of numbers.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        initial_values (List[float]): The initial values for the set.
        values_to_add (List[float]): Values to add to the set.
        values_to_remove (List[float]): Values to remove from the set.

    Returns:
        Dict[str, Any]: A dictionary containing the responses from each
 operation.
    """
    # Step 1: Create the number set
    create_response = create_set_attribute(
        table_name=table_name,
        key=key,
        set_name=set_name,
        set_values=initial_values,
        set_type="number",
    )

    # Step 2: Add more numbers to the set
    add_response = add_to_set(
        table_name=table_name, key=key, set_name=set_name,
 values_to_add=values_to_add
    )
```

```
    # Step 3: Remove some numbers from the set
    remove_response = remove_from_set(
        table_name=table_name, key=key, set_name=set_name,
 values_to_remove=values_to_remove
    )

    # Step 4: Get the final state
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    get_response = table.get_item(
        Key=key,
        ProjectionExpression=f"#{set_name}",
        ExpressionAttributeNames={f"#{set_name}": set_name},
    )

    # Return all responses
    return {
        "create_response": create_response,
        "add_response": add_response,
        "remove_response": remove_response,
        "final_state": get_response.get("Item", {}),
    }
```

AWS SDK for Python (Boto3)를 사용한 집합 작업의 사용 예입니다.

```
def example_usage():
    """Example of how to use the set operations functions."""
    # Example parameters
    table_name = "UserPreferences"
    key = {"UserId": "user123"}

    print("Example 1: Creating a string set attribute")
    try:
        response = create_set_attribute(
            table_name=table_name,
            key=key,
            set_name="FavoriteTags",
            set_values=["AWS", "DynamoDB", "NoSQL"],
```

```
            set_type="string",
        )
        print(f"Set attribute created successfully: {response.get('Attributes',
{})}")
    except Exception as e:
        print(f"Error creating set attribute: {e}")

    print("\nExample 2: Adding elements to an existing set")
    try:
        response = add_to_set(
            table_name=table_name,
            key=key,
            set_name="FavoriteTags",
            values_to_add=["Database", "Serverless"],
        )
        print(f"Elements added to set successfully: {response.get('Attributes',
{})}")
    except Exception as e:
        print(f"Error adding to set: {e}")

    print("\nExample 3: Removing elements from a set")
    try:
        response = remove_from_set(
            table_name=table_name, key=key, set_name="FavoriteTags",
values_to_remove=["NoSQL"]
        )
        print(f"Elements removed from set successfully:
{response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error removing from set: {e}")

    print("\nExample 4: Demonstrating what happens when you remove the last
element from a set")
    try:
        results = demonstrate_last_element_removal(
            table_name=table_name, key={"UserId": "tempUser"},
set_name="SingleElementSet"
        )

        print(f"Set exists before removal: {results['exists_before']}")
        print(f"Set exists after removal: {results['exists_after']}")

        if not results["exists_after"]:
```

```
            print("The set attribute was completely removed when the last element
was deleted.")
        else:
            print("The set attribute still exists after removing the last
element.")
    except Exception as e:
        print(f"Error in last element removal demonstration: {e}")

    print("\nExample 5: Working with a number set")
    try:
        results = work_with_number_set(
            table_name=table_name,
            key={"UserId": "user456"},
            set_name="LuckyNumbers",
            initial_values=[7, 13, 42],
            values_to_add=[99, 100],
            values_to_remove=[13],
        )

        print(f"Initial number set: {results['create_response'].get('Attributes',
{})}")
        print(f"After adding numbers: {results['add_response'].get('Attributes',
{})}")
        print(f"After removing numbers:
{results['remove_response'].get('Attributes', {})}")
        print(f"Final state: {results['final_state']}")
    except Exception as e:
        print(f"Error working with number set: {e}")

    print("\nKey Points About DynamoDB Sets:")
    print("1. Sets can only contain elements of the same type (string, number, or
binary)")
    print("2. Sets automatically eliminate duplicate values")
    print("3. The ADD operation creates a set if it doesn't exist")
    print("4. The DELETE operation removes specified elements from a set")
    print("5. When the last element is removed from a set, the entire attribute
is deleted")
    print("6. Empty sets are not allowed in DynamoDB")
    print("7. Sets are unordered collections")
    print("8. The ADD operation is atomic for sets")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## PartiQL 문 배치 및 AWS SDK를 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 여러 SELECT 문을 실행하여 항목 배치를 가져옵니다.
- 여러 INSERT 문을 실행하여 항목 배치를 추가합니다.
- 여러 UPDATE 문을 실행하여 항목 배치를 업데이트합니다.
- 여러 DELETE 문을 실행하여 항목 배치를 삭제합니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Before you run this example, download 'movies.json' from
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
GettingStarted.Js.02.html,
// and put it in the same folder as the example.

// Separator for the console display.
var SepBar = new string('-', 80);
const string tableName = "movie_table";
const string movieFileName = @"..\..\..\..\..\..\..\resources\sample_files
\movies.json";

DisplayInstructions();

// Create the table and wait for it to be active.
```

```
Console.WriteLine($"Creating the movie table: {tableName}");

var success = await DynamoDBMethods.CreateMovieTableAsync(tableName);
if (success)
{
    Console.WriteLine($"Successfully created table: {tableName}.");
}

WaitForEnter();

// Add movie information to the table from moviedata.json. See the
// instructions at the top of this file to download the JSON file.
Console.WriteLine($"Inserting movies into the new table. Please wait...");
success = await PartiQLBatchMethods.InsertMovies(tableName, movieFileName);
if (success)
{
    Console.WriteLine("Movies successfully added to the table.");
}
else
{
    Console.WriteLine("Movies could not be added to the table.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var title1 = "Star Wars";
var year1 = 1977;
var title2 = "Wizard of Oz";
var year2 = 1939;

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.GetBatch(tableName, title1, title2, year1,
 year2);
if (success)
{
    Console.WriteLine($"Successfully retrieved {title1} and {title2}.");
}
else
{
    Console.WriteLine("Select statement failed.");
}
```

```
WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var producer1 = "LucasFilm";
var producer2 = "MGM";

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.UpdateBatch(tableName, producer1, title1,
 year1, producer2, title2, year2);
if (success)
{
    Console.WriteLine($"Successfully updated {title1} and {title2}.");
}
else
{
    Console.WriteLine("Update failed.");
}

WaitForEnter();

// Delete multiple movies by using the BatchExecute statement.
Console.WriteLine($"Now we will delete {title1} and {title2} from the table.");
success = await PartiQLBatchMethods.DeleteBatch(tableName, title1, year1, title2,
 year2);

if (success)
{
    Console.WriteLine($"Deleted {title1} and {title2}");
}
else
{
    Console.WriteLine($"could not delete {title1} or {title2}");
}

WaitForEnter();

// DNow that the PartiQL Batch scenario is complete, delete the movie table.
success = await DynamoDBMethods.DeleteTableAsync(tableName);

if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
```

```
else
{
    Console.WriteLine($"Could not delete {tableName}");
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
void DisplayInstructions()
{
    Console.Clear();
    Console.WriteLine();
    Console.Write(new string(' ', 24));
    Console.WriteLine("DynamoDB PartiQL Basics Example");
    Console.WriteLine(SepBar);
    Console.WriteLine("This demo application shows the basics of using Amazon
 DynamoDB with the AWS SDK for");
    Console.WriteLine(".NET version 3.7 and .NET 6.");
    Console.WriteLine(SepBar);
    Console.WriteLine("Creates a table by using the CreateTable method.");
    Console.WriteLine("Gets multiple movies by using a PartiQL SELECT
 statement.");
    Console.WriteLine("Updates multiple movies by using the ExecuteBatch
 method.");
    Console.WriteLine("Deletes multiple movies by using a PartiQL DELETE
 statement.");
    Console.WriteLine("Cleans up the resources created for the demo by deleting
 the table.");
    Console.WriteLine(SepBar);

    WaitForEnter();
}

/// <summary>
/// Simple method to wait for the <Enter> key to be pressed.
/// </summary>
void WaitForEnter()
{
    Console.WriteLine("\nPress <Enter> to continue.");
    Console.Write(SepBar);
    _ = Console.ReadLine();
}
```

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
= ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
```

```
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    if (r.Item.Any())
                    {

Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                    }
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }

        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
```

```
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
```

```
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }

        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
```

```
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
 = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
```

```
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }

        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
 year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },
```

```
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 BatchExecuteStatement를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        //  1. Create a table. (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::partiqlBatchExecuteScenario(clientConfig);

            // 7. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
```

```
        }

//! Scenario to modify and query a DynamoDB table using PartiQL batch statements.
/*!
  \sa partiqlBatchExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::partiqlBatchExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {

    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
              << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
```

```
        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

            // Create attribute for the info map.
            Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
= Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            ratingAttribute->SetN(ratings[i]);
            infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            plotAttribute->SetS(plots[i]);
            infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
            attributes.push_back(infoMapAttribute);
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
```

```
                    << std::endl;
            return false;
        }
    }

    std::cout << "Retrieving the movie data with a batch \"SELECT\" statement."
              << std::endl;

    // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();
```

```
            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

    for (size_t i = 0; i < titles.size(); ++i) {
        ratings[i] = askQuestionForFloatRange(
                Aws::String("\nLet's update your the movie, \"") + titles[i] +
                ".\nYou rated it  " + std::to_string(ratings[i])
                + ", what new rating would you give it? ", 1, 10);
    }

    std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";


        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
```

```
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);
        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Retrieving the updated movie data with a batch \"SELECT\"
statement."
              << std::endl;

    // 5. Get the updated data for multiple movies using "Select" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
```

```
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movies information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Deleting the movie data with a batch \"DELETE\" statement."
              << std::endl;

    // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";
```

```
        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

 attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
 dynamoClient.BatchExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movies: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;
```

```
        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
```

```
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
```

```
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
```

```
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 BatchExecuteStatement를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블을 생성하고 PartiQL 쿼리 배치를 실행하는 시나리오를 실행합니다.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLBatchScenario shows you how to use the AWS SDK for Go
// to run batches of PartiQL statements to query a table that stores data about
 movies.
//
//   - Use batches of PartiQL statements to add, get, update, and delete data for
//     individual movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
```

```
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLBatchScenario(ctx context.Context, sdkConfig aws.Config, tableName
 string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB PartiQL batch demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }
 runner := actions.PartiQLRunner{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }

 exists, err := tableBasics.TableExists(ctx)
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable(ctx)
  if err != nil {
   panic(err)
  } else {
   log.Printf("Created table %v.\n", tableName)
  }
 } else {
  log.Printf("Table %v already exists.\n", tableName)
 }
 log.Println(strings.Repeat("-", 88))

 currentYear, _, _ := time.Now().Date()
 customMovies := []actions.Movie{{
  Title: "House PartiQL",
  Year:  currentYear - 5,
  Info: map[string]interface{}{
```

```
  "plot":   "Wacky high jinks result from querying a mysterious database.",
  "rating": 8.5}}, {
 Title: "House PartiQL 2",
 Year:  currentYear - 3,
 Info: map[string]interface{}{
  "plot":   "Moderate high jinks result from querying another mysterious
database.",
  "rating": 6.5}}, {
 Title: "House PartiQL 3",
 Year:  currentYear - 1,
 Info: map[string]interface{}{
  "plot":   "Tepid high jinks result from querying yet another mysterious
database.",
  "rating": 2.5},
},
}

log.Printf("Inserting a batch of movies into table '%v'.\n", tableName)
err = runner.AddMovieBatch(ctx, customMovies)
if err == nil {
 log.Printf("Added %v movies to the table.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting data for a batch of movies.")
movies, err := runner.GetMovieBatch(ctx, customMovies)
if err == nil {
 for _, movie := range movies {
  log.Println(movie)
 }
}
log.Println(strings.Repeat("-", 88))

newRatings := []float64{7.7, 4.4, 1.1}
log.Println("Updating a batch of movies with new ratings.")
err = runner.UpdateMovieBatch(ctx, customMovies, newRatings)
if err == nil {
 log.Printf("Updated %v movies with new ratings.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting projected data from the table to verify our update.")
log.Println("Using a page size of 2 to demonstrate paging.")
projections, err := runner.GetAllMovies(ctx, 2)
```

```
 if err == nil {
  log.Println("All movies:")
  for _, projection := range projections {
   log.Println(projection)
  }
 }
 log.Println(strings.Repeat("-", 88))

 log.Println("Deleting a batch of movies.")
 err = runner.DeleteMovieBatch(ctx, customMovies)
 if err == nil {
  log.Printf("Deleted %v movies.\n", len(customMovies))
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

PartiQL 문을 실행하는 구조체와 메서드를 생성합니다.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
```

```
 return err
}



// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
```

```
}



// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}
```

```
// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}



// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
```

```
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 BatchExecuteStatement를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
public class ScenarioPartiQLBatch {
    public static void main(String[] args) throws IOException {
        String tableName = "MoviesPartiQBatch";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println("Creating an Amazon DynamoDB table named " + tableName
            + " with a key named year and a sort key named title.");
```

```
        createTable(ddb, tableName);

        System.out.println("Adding multiple records into the " + tableName
            + " table using a batch command.");
        putRecordBatch(ddb);

        // Update multiple movies by using the BatchExecute statement.
        String title1 = "Star Wars";
        int year1 = 1977;
        String title2 = "Wizard of Oz";
        int year2 = 1939;

        System.out.println("Query two movies.");
        getBatch(ddb, tableName, title1, title2, year1, year2);

        System.out.println("Updating multiple records using a batch command.");
        updateTableItemBatch(ddb);

        System.out.println("Deleting multiple records using a batch command.");
        deleteItemBatch(ddb);

        System.out.println("Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static boolean getBatch(DynamoDbClient ddb, String tableName, String
title1, String title2, int year1, int year2) {
        String getBatch = "SELECT * FROM " + tableName + " WHERE title = ? AND
year = ?";

        List<BatchStatementRequest> statements = new ArrayList<>();
        statements.add(BatchStatementRequest.builder()
            .statement(getBatch)
            .parameters(AttributeValue.builder().s(title1).build(),
                AttributeValue.builder().n(String.valueOf(year1)).build())
            .build());
        statements.add(BatchStatementRequest.builder()
            .statement(getBatch)
            .parameters(AttributeValue.builder().s(title2).build(),
                AttributeValue.builder().n(String.valueOf(year2)).build())
            .build());
```

```
        BatchExecuteStatementRequest batchExecuteStatementRequest =
BatchExecuteStatementRequest.builder()
            .statements(statements)
            .build();

        try {
            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchExecuteStatementRequest);
            if (!response.responses().isEmpty()) {
                response.responses().forEach(r -> {
                    System.out.println(r.item().get("title") + "\\t" +
r.item().get("year"));
                });
                return true;
            } else {
                System.out.println("Couldn't find either " + title1 + " or " +
title2 + ".");
                return false;
            }
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            return false;
        }
    }

    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
```

```
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE) // Sort
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse = dbWaiter
                .waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void putRecordBatch(DynamoDbClient ddb) {
        String sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
'title' : ?, 'info' : ?}";
        try {
            // Create three movies to add to the Amazon DynamoDB table.
            // Set data for Movie 1.
```

```
            List<AttributeValue> parameters = new ArrayList<>();

            AttributeValue att1 = AttributeValue.builder()
                .n("1977")
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s("Star Wars")
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s("No Information")
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            BatchStatementRequest statementRequestMovie1 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parameters)
                .build();

            // Set data for Movie 2.
            List<AttributeValue> parametersMovie2 = new ArrayList<>();
            AttributeValue attMovie2 = AttributeValue.builder()
                .n("1939")
                .build();

            AttributeValue attMovie2A = AttributeValue.builder()
                .s("Wizard of Oz")
                .build();

            AttributeValue attMovie2B = AttributeValue.builder()
                .s("No Information")
                .build();

            parametersMovie2.add(attMovie2);
            parametersMovie2.add(attMovie2A);
            parametersMovie2.add(attMovie2B);

            BatchStatementRequest statementRequestMovie2 =
BatchStatementRequest.builder()
```

```
                .statement(sqlStatement)
                .parameters(parametersMovie2)
                .build();

            // Set data for Movie 3.
            List<AttributeValue> parametersMovie3 = new ArrayList<>();
            AttributeValue attMovie3 = AttributeValue.builder()
                .n(String.valueOf("2022"))
                .build();

            AttributeValue attMovie3A = AttributeValue.builder()
                .s("My Movie 3")
                .build();

            AttributeValue attMovie3B = AttributeValue.builder()
                .s("No Information")
                .build();

            parametersMovie3.add(attMovie3);
            parametersMovie3.add(attMovie3A);
            parametersMovie3.add(attMovie3B);

            BatchStatementRequest statementRequestMovie3 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parametersMovie3)
                .build();

            // Add all three movies to the list.
            List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
            myBatchStatementList.add(statementRequestMovie1);
            myBatchStatementList.add(statementRequestMovie2);
            myBatchStatementList.add(statementRequestMovie3);

            BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
                .statements(myBatchStatementList)
                .build();

            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchRequest);
            System.out.println("ExecuteStatement successful: " +
response.toString());
            System.out.println("Added new movies using a batch command.");
```

```
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void updateTableItemBatch(DynamoDbClient ddb) {
        String sqlStatement = "UPDATE MoviesPartiQBatch SET info = 'directors\":
[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and title=?";
        List<AttributeValue> parametersRec1 = new ArrayList<>();

        // Update three records.
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("My Movie 1")
            .build();

        parametersRec1.add(att1);
        parametersRec1.add(att2);

        BatchStatementRequest statementRequestRec1 =
 BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec1)
            .build();

        // Update record 2.
        List<AttributeValue> parametersRec2 = new ArrayList<>();
        AttributeValue attRec2 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec2a = AttributeValue.builder()
            .s("My Movie 2")
            .build();

        parametersRec2.add(attRec2);
        parametersRec2.add(attRec2a);
        BatchStatementRequest statementRequestRec2 =
 BatchStatementRequest.builder()
```

```
            .statement(sqlStatement)
            .parameters(parametersRec2)
            .build();

        // Update record 3.
        List<AttributeValue> parametersRec3 = new ArrayList<>();
        AttributeValue attRec3 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec3a = AttributeValue.builder()
            .s("My Movie 3")
            .build();

        parametersRec3.add(attRec3);
        parametersRec3.add(attRec3a);
        BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec3)
            .build();

        // Add all three movies to the list.
        List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
        myBatchStatementList.add(statementRequestRec1);
        myBatchStatementList.add(statementRequestRec2);
        myBatchStatementList.add(statementRequestRec3);

        BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
            .statements(myBatchStatementList)
            .build();

        try {
            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchRequest);
            System.out.println("ExecuteStatement successful: " +
response.toString());
            System.out.println("Updated three movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
```

```
        System.out.println("Item was updated!");
    }

    public static void deleteItemBatch(DynamoDbClient ddb) {
        String sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and
title=?";
        List<AttributeValue> parametersRec1 = new ArrayList<>();

        // Specify three records to delete.
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("My Movie 1")
            .build();

        parametersRec1.add(att1);
        parametersRec1.add(att2);

        BatchStatementRequest statementRequestRec1 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec1)
            .build();

        // Specify record 2.
        List<AttributeValue> parametersRec2 = new ArrayList<>();
        AttributeValue attRec2 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec2a = AttributeValue.builder()
            .s("My Movie 2")
            .build();

        parametersRec2.add(attRec2);
        parametersRec2.add(attRec2a);
        BatchStatementRequest statementRequestRec2 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec2)
            .build();
```

```
        // Specify record 3.
        List<AttributeValue> parametersRec3 = new ArrayList<>();
        AttributeValue attRec3 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec3a = AttributeValue.builder()
            .s("My Movie 3")
            .build();

        parametersRec3.add(attRec3);
        parametersRec3.add(attRec3a);

        BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec3)
            .build();

        // Add all three movies to the list.
        List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
        myBatchStatementList.add(statementRequestRec1);
        myBatchStatementList.add(statementRequestRec2);
        myBatchStatementList.add(statementRequestRec3);

        BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
            .statements(myBatchStatementList)
            .build();

        try {
            ddb.batchExecuteStatement(batchRequest);
            System.out.println("Deleted three movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{
        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
```

```
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    private static ExecuteStatementResponse
 executeStatementRequest(DynamoDbClient ddb, String statement,

 List<AttributeValue> parameters) {
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .statement(statement)
            .parameters(parameters)
            .build();

        return ddb.executeStatement(request);
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 BatchExecuteStatement를 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

배치 PartiQL 명령문을 실행합니다.

```
import {
  BillingMode,
```

```
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({}, { confirmAll });
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
```

```
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */
```

```
  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statements: [
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
  });
  await docClient.send(addItemsStatementCommand);
  log("Cities inserted.");

  /**
   * Select items.
   */

  log("Selecting cities from the table.");
  const selectItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statements: [
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
```

```
      },
    ],
  });
  const selectItemResponse = await docClient.send(selectItemsStatementCommand);
  log(
    `Got cities: ${selectItemResponse.Responses.map(
      (r) => `${r.Item.name} (${r.Item.population})`,
    ).join(", ")}`,
  );

  /**
   * Update items.
   */

  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
  await docClient.send(updateItemStatementCommand);
  log("Updated cities.");

  /**
   * Delete the items.
   */

  log("Deleting the cities.");
  const deleteItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statements: [
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
```

```
      },
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Cities deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 BatchExecuteStatement를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQBatch"
    println("Creating an Amazon DynamoDB table named $tableName with a key named
 id and a sort key named title.")
    createTablePartiQLBatch(ddb, tableName, "year")
    putRecordBatch(ddb)
```

```
    updateTableItemBatchBatch(ddb)
    deleteItemsBatch(ddb)
    deleteTablePartiQLBatch(tableName)
}

suspend fun createTablePartiQLBatch(
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
```

```
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
}

suspend fun putRecordBatch(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
 'title' : ?, 'info' : ?}"

    // Create three movies to add to the Amazon DynamoDB table.
    val parametersMovie1 = mutableListOf<AttributeValue>()
    parametersMovie1.add(AttributeValue.N("2022"))
    parametersMovie1.add(AttributeValue.S("My Movie 1"))
    parametersMovie1.add(AttributeValue.S("No Information"))

    val statementRequestMovie1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie1
        }

    // Set data for Movie 2.
    val parametersMovie2 = mutableListOf<AttributeValue>()
    parametersMovie2.add(AttributeValue.N("2022"))
    parametersMovie2.add(AttributeValue.S("My Movie 2"))
    parametersMovie2.add(AttributeValue.S("No Information"))

    val statementRequestMovie2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie2
        }

    // Set data for Movie 3.
    val parametersMovie3 = mutableListOf<AttributeValue>()
    parametersMovie3.add(AttributeValue.N("2022"))
    parametersMovie3.add(AttributeValue.S("My Movie 3"))
    parametersMovie3.add(AttributeValue.S("No Information"))

    val statementRequestMovie3 =
        BatchStatementRequest {
            statement = sqlStatement
```

```
            parameters = parametersMovie3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestMovie1)
    myBatchStatementList.add(statementRequestMovie2)
    myBatchStatementList.add(statementRequestMovie3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }
    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: " + response.toString())
    println("Added new movies using a batch command.")
}

suspend fun updateTableItemBatchBatch(ddb: DynamoDbClient) {
    val sqlStatement =
        "UPDATE MoviesPartiQBatch SET info = 'directors\":[\"Merian C. Cooper\",
\"Ernest B. Schoedsack' where year=? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))
    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Update record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec2
        }

    // Update record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
    parametersRec3.add(AttributeValue.N("2022"))
```

```
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: $response")
    println("Updated three movies using a batch command.")
    println("Items were updated!")
}

suspend fun deleteItemsBatch(ddb: DynamoDbClient) {
    // Specify three records to delete.
    val sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))

    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Specify record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
```

```
            parameters = parametersRec2
        }

    // Specify record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    ddb.batchExecuteStatement(batchRequest)
    println("Deleted three movies using a batch command.")
}

suspend fun deleteTablePartiQLBatch(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 BatchExecuteStatement를 참조하세요.

## PHP

### SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithPartiQLBatch
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
 using PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
```

```
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQLBatch($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
```

```
        $service->updateItemByPartiQLBatch($statement, $parameters);
        echo "Movie added and updated.\n";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "\nThe movie {$movie['Responses'][0]['Item']['title']['S']}
        was released in {$movie['Responses'][0]['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Responses'][0]['Item']
['title']['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQLBatch($statement, $parameters);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "Okay, you have rated {$movie['Responses'][0]['Item']['title']
['S']}
        as a {$movie['Responses'][0]['Item']['rating']['N']}\n";

        $service->deleteItemByPartiQLBatch($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
```

```
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
```

```
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }

        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
```

```
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 BatchExecuteStatement를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

PartiQL 문 배치를 실행할 수 있는 클래스를 생성합니다.

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint

import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
```

```
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
the
        resource transforms input and output from plain old Python objects
(POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
with
                           each statement. This list must be in the same order as
the
                           statements.
        :return: The responses returned from running the statements, if any.
        """
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
table "
                    "does not exist."
                )
            else:
                logger.error(
                    "Couldn't execute batch of PartiQL statements. Here's why:
%s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

```
        else:
            return output
```

테이블을 생성하고 PartiQL 쿼리를 배치로 실행하는 시나리오를 실행합니다.

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL batch statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)

    movie_data = [
        {
            "title": f"House PartiQL",
            "year": datetime.now().year - 5,
            "info": {
                "plot": "Wacky high jinks result from querying a mysterious
 database.",
                "rating": Decimal("8.5"),
            },
        },
        {
            "title": f"House PartiQL 2",
            "year": datetime.now().year - 3,
            "info": {
                "plot": "Moderate high jinks result from querying another
 mysterious database.",
                "rating": Decimal("6.5"),
            },
        },
        {
            "title": f"House PartiQL 3",
            "year": datetime.now().year - 1,
            "info": {
```

```
                "plot": "Tepid high jinks result from querying yet another
mysterious database.",
                "rating": Decimal("2.5"),
            },
        },
    ]

    print(f"Inserting a batch of movies into table '{table_name}.")
    statements = [
        f'INSERT INTO "{table_name}" ' f"VALUE {{'title': ?, 'year': ?,
'info': ?}}"
    ] * len(movie_data)
    params = [list(movie.values()) for movie in movie_data]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting data for a batch of movies.")
    statements = [f'SELECT * FROM "{table_name}" WHERE title=? AND year=?'] *
len(
        movie_data
    )
    params = [[movie["title"], movie["year"]] for movie in movie_data]
    output = wrapper.run_partiql(statements, params)
    for item in output["Responses"]:
        print(f"\n{item['Item']['title']}, {item['Item']['year']}")
        pprint(item["Item"])
    print("-" * 88)

    ratings = [Decimal("7.7"), Decimal("5.5"), Decimal("1.3")]
    print(f"Updating a batch of movies with new ratings.")
    statements = [
        f'UPDATE "{table_name}" SET info.rating=? ' f"WHERE title=? AND year=?"
    ] * len(movie_data)
    params = [
        [rating, movie["title"], movie["year"]]
        for rating, movie in zip(ratings, movie_data)
    ]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting projected data from the table to verify our update.")
    output = wrapper.dyn_resource.meta.client.execute_statement(
```

```
            Statement=f'SELECT title, info.rating FROM "{table_name}"'
        )
        pprint(output["Items"])
        print("-" * 88)

        print(f"Deleting a batch of movies from the table.")
        statements = [f'DELETE FROM "{table_name}" WHERE title=? AND year=?'] * len(
            movie_data
        )
        params = [[movie["title"], movie["year"]] for movie in movie_data]
        wrapper.run_partiql(statements, params)
        print("Success!")
        print("-" * 88)

        print(f"Deleting table '{table_name}'...")
        scaffold.delete_table()
        print("-" * 88)

        print("\nThanks for watching!")
        print("-" * 88)


if __name__ == "__main__":
    try:
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLBatchWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 BatchExecuteStatement를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블을 생성하고 배치 PartiQL 쿼리를 실행하는 시나리오를 실행합니다.

```
 table_name = "doc-example-table-movies-partiql-#{rand(10**4)}"
 scaffold = Scaffold.new(table_name)
 sdk = DynamoDBPartiQLBatch.new(table_name)

 new_step(1, 'Create a new DynamoDB table if none already exists.')
 unless scaffold.exists?(table_name)
   puts("\nNo such table: #{table_name}. Creating it...")
   scaffold.create_table(table_name)
   print "Done!\n".green
 end

 new_step(2, 'Populate DynamoDB table with movie data.')
 download_file = 'moviedata.json'
 puts("Downloading movie database to #{download_file}...")
 movie_data = scaffold.fetch_movie_data(download_file)
 puts("Writing movie data from #{download_file} into your table...")
 scaffold.write_batch(movie_data)
 puts("Records added: #{movie_data.length}.")
 print "Done!\n".green

 new_step(3, 'Select a batch of items from the movies table.')
 puts "Let's select some popular movies for side-by-side comparison."
 response = sdk.batch_execute_select([['Mean Girls', 2004], ['Goodfellas',
1977], ['The Prancing of the Lambs', 2005]])
 puts("Items selected: #{response['responses'].length}\n")
 print "\nDone!\n".green

 new_step(4, 'Delete a batch of items from the movies table.')
 sdk.batch_execute_write([['Mean Girls', 2004], ['Goodfellas', 1977], ['The
Prancing of the Lambs', 2005]])
 print "\nDone!\n".green
```

```
  new_step(5, 'Delete the table.')
  return unless scaffold.exists?(table_name)

  scaffold.delete_table
end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 BatchExecuteStatement를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## PartiQL 및 AWS SDK를 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- SELECT 문을 실행하여 항목을 가져옵니다.
- INSERT 문을 실행하여 항목을 추가합니다.
- UPDATE 문을 실행하여 항목을 업데이트합니다.
- DELETE 문을 실행하여 항목을 삭제합니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
namespace PartiQL_Basics_Scenario
{
    public class PartiQLMethods
    {
        private static readonly AmazonDynamoDBClient Client = new
 AmazonDynamoDBClient();
```

```
        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table where the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
```

```
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
```

```
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }


        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
```

```
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

```
        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }


        /// <summary>
        /// Displays the list of movies returned from a database query.
        /// </summary>
        /// <param name="items">The list of movie information to display.</param>
        private static void DisplayMovies(List<Dictionary<string,
AttributeValue>> items)
        {
            if (items.Count > 0)
            {
                Console.WriteLine($"Found {items.Count} movies.");
                items.ForEach(item =>
Console.WriteLine($"{item["year"].N}\t{item["title"].S}"));
            }
            else
```

```
            {
                Console.WriteLine($"Didn't find a movie that matched the supplied
 criteria.");
            }
        }


    }
}



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
 GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }



        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
```

```
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";
```

```
            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ExecuteStatement를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        //  1. Create a table. (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::partiqlExecuteScenario(clientConfig);

            // 7. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }

//! Scenario to modify and query a DynamoDB table using single PartiQL
 statements.
/*!
  \sa partiqlExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::partiqlExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
    Aws::String title;
    float rating;
    int year;
    Aws::String plot;
    {
```

```
        title = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        year = askQuestionForInt("What year was it released? ");
        rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                          1, 10);
        plot = askQuestion("Summarize the plot for me: ");

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        request.SetStatement(sqlStream.str());

        // Create the parameter attributes.
        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        ratingAttribute->SetN(rating);
        infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        plotAttribute->SetS(plot);
        infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
        attributes.push_back(infoMapAttribute);
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add a movie: " <<
outcome.GetError().GetMessage()
```

```
                    << std::endl;
            return false;
        }
    }

    std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
              << std::endl;

    //  3. Get the data for the movie using a "Select" statement.
(ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            // Print the retrieved movie information.
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
```

```
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
    }

    //  4. Update the data for the movie using an "Update" statement.
(ExecuteStatement)
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update a movie: "
                      << outcome.GetError().GetMessage();
            return false;
        }
    }

    std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;
```

```
   //  5. Get the updated data for the movie using a "Select" statement.
(ExecuteStatement)
   {
       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       request.SetStatement(sqlStream.str());

       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
               request);
       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to retrieve the movie information: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
       else {
           const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

           if (items.size() == 1) {
               printMovieInfo(items[0]);
           }
           else {
               std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                         << " There should be only one movie." << std::endl;
           }
       }
   }

   std::cout << "Deleting the movie" << std::endl;

   // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
```

```
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
 dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movie: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Movie successfully deleted." << std::endl;
    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
```

```
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
```

```
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
```

```
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 ExecuteStatement를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블을 생성하고 PartiQL 쿼리를 실행하는 시나리오를 실행합니다.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLSingleScenario shows you how to use the AWS SDK for Go
// to use PartiQL to query a table that stores data about movies.
//
// * Use PartiQL statements to add, get, update, and delete data for individual
 movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLSingleScenario(ctx context.Context, sdkConfig aws.Config,
 tableName string) {
```

```
defer func() {
 if r := recover(); r != nil {
  fmt.Printf("Something went wrong with the demo.")
 }
}()

log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the Amazon DynamoDB PartiQL single action demo.")
log.Println(strings.Repeat("-", 88))

tableBasics := actions.TableBasics{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}
runner := actions.PartiQLRunner{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}

exists, err := tableBasics.TableExists(ctx)
if err != nil {
 panic(err)
}
if !exists {
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}
log.Println(strings.Repeat("-", 88))

currentYear, _, _ := time.Now().Date()
customMovie := actions.Movie{
 Title: "24 Hour PartiQL People",
 Year:  currentYear,
 Info: map[string]interface{}{
  "plot":   "A group of data developers discover a new query language they can't
stop using.",
  "rating": 9.9,
```

```
 },
}

log.Printf("Inserting movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
err = runner.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting data for movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
movie, err := runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

newRating := 6.6
log.Printf("Updating movie '%v' with a rating of %v.", customMovie.Title,
newRating)
err = runner.UpdateMovie(ctx, customMovie, newRating)
if err == nil {
 log.Printf("Updated %v with a new rating.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting data again to verify the update.")
movie, err = runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Deleting movie '%v'.\n", customMovie.Title)
err = runner.DeleteMovie(ctx, customMovie)
if err == nil {
 log.Printf("Deleted %v.\n", customMovie.Title)
}

err = tableBasics.DeleteTable(ctx)
if err == nil {
 log.Printf("Deleted table %v.\n", tableBasics.TableName)
```

```
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

이 예시에서 사용되는 Movie 구조체를 정의합니다.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
```

```
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

PartiQL 문을 실행하는 구조체와 메서드를 생성합니다.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
```

```
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}



// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
```

```
 }
 return movie, err
}



// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}



// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
```

```
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 ExecuteStatement를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
public class ScenarioPartiQ {
    public static void main(String[] args) throws IOException {
        String fileName = "../../../resources/sample_files/movies.json";
        String tableName = "MoviesPartiQ";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println(
            "******* Creating an Amazon DynamoDB table named MoviesPartiQ with a
 key named year and a sort key named title.");
        createTable(ddb, tableName);

        System.out.println("Loading data into the MoviesPartiQ table.");
        loadData(ddb, fileName);

        System.out.println("Getting data from the MoviesPartiQ table.");
        getItem(ddb);
```

```
        System.out.println("Putting a record into the MoviesPartiQ table.");
        putRecord(ddb);

        System.out.println("Updating a record.");
        updateTableItem(ddb);

        System.out.println("Querying the movies released in 2013.");
        queryTable(ddb);

        System.out.println("Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE) // Sort
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);
```

```
        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //Scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Load data into the table.
    public static void loadData(DynamoDbClient ddb, String fileName) throws
IOException {

        String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
        ObjectNode currentNode;
        int t = 0;
        List<AttributeValue> parameters = new ArrayList<>();
        while (iter.hasNext()) {

            // Add 200 movies to the table.
            if (t == 200)
```

```
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf(year))
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s(title)
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s(info)
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            // Insert the movie into the Amazon DynamoDB table.
            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added Movie " + title);

            parameters.remove(att1);
            parameters.remove(att2);
            parameters.remove(att3);
            t++;
        }
    }

    public static void getItem(DynamoDbClient ddb) {

        String sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and
title=?";
        List<AttributeValue> parameters = new ArrayList<>();
        AttributeValue att1 = AttributeValue.builder()
            .n("2012")
            .build();

        AttributeValue att2 = AttributeValue.builder()
```

```
            .s("The Perks of Being a Wallflower")
            .build();

        parameters.add(att1);
        parameters.add(att2);

        try {
            ExecuteStatementResponse response = executeStatementRequest(ddb,
sqlStatement, parameters);
            System.out.println("ExecuteStatement successful: " +
response.toString());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void putRecord(DynamoDbClient ddb) {

        String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
        try {
            List<AttributeValue> parameters = new ArrayList<>();

            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf("2020"))
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s("My Movie")
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s("No Information")
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added new movie.");
```

```
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void updateTableItem(DynamoDbClient ddb) {

        String sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":
[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and title=?";
        List<AttributeValue> parameters = new ArrayList<>();
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2013"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("The East")
            .build();

        parameters.add(att1);
        parameters.add(att2);

        try {
            executeStatementRequest(ddb, sqlStatement, parameters);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Item was updated!");
    }

    // Query the table where the year is 2013.
    public static void queryTable(DynamoDbClient ddb) {
        String sqlStatement = "SELECT * FROM MoviesPartiQ where year = ? ORDER BY
 year";
        try {

            List<AttributeValue> parameters = new ArrayList<>();
            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf("2013"))
                .build();
            parameters.add(att1);
```

```
            // Get items in the table and write out the ID value.
            ExecuteStatementResponse response = executeStatementRequest(ddb,
sqlStatement, parameters);
            System.out.println("ExecuteStatement successful: " +
response.toString());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{

        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    private static ExecuteStatementResponse
executeStatementRequest(DynamoDbClient ddb, String statement,

List<AttributeValue> parameters) {
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .statement(statement)
            .parameters(parameters)
            .build();

        return ddb.executeStatement(request);
    }

    private static void processResults(ExecuteStatementResponse
executeStatementResult) {
```

```
        System.out.println("ExecuteStatement successful: " +
 executeStatementResult.toString());
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 ExecuteStatement를 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

단일 PartiQL 문을 실행합니다.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async (confirmAll = false) => {
  /**
```

```
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
```

```
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert an item.
   */

  log("Inserting a coffee into the table.");
  const addItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
    Parameters: ["arabica", ["chocolate", "floral"]],
  });
  await client.send(addItemStatementCommand);
  log("Coffee inserted.");
```

```
  /**
   * Select an item.
   */

  log("Selecting the coffee from the table.");
  const selectItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  const selectItemResponse = await docClient.send(selectItemStatementCommand);
  log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

  /**
   * Update the item.
   */

  log("Add a flavor profile to the coffee.");
  const updateItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
 varietal=?`,
    Parameters: [["fruity"], "arabica"],
  });
  await client.send(updateItemStatementCommand);
  log("Updated coffee");

  /**
   * Delete the item.
   */

  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Coffee deleted.");

  /**
```

```
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 ExecuteStatement를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQ"
    val fileName = "../../../resources/sample_files/movies.json"
    println("Creating an Amazon DynamoDB table named MoviesPartiQ with a key
 named id and a sort key named title.")
    createTablePartiQL(ddb, tableName, "year")
    loadDataPartiQL(ddb, fileName)

    println("******* Getting data from the MoviesPartiQ table.")
    getMoviePartiQL(ddb)

    println("******* Putting a record into the MoviesPartiQ table.")
    putRecordPartiQL(ddb)

    println("******* Updating a record.")
    updateTableItemPartiQL(ddb)

    println("******* Querying the movies released in 2013.")
    queryTablePartiQL(ddb)
```

```
    println("******* Deleting the MoviesPartiQ table.")
    deleteTablePartiQL(tableName)
}

suspend fun createTablePartiQL(
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
```

```
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
}

suspend fun loadDataPartiQL(
    ddb: DynamoDbClient,
    fileName: String,
) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode
    var t = 0

    while (iter.hasNext()) {
        if (t == 200) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()

        val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
        parameters.add(AttributeValue.N(year.toString()))
        parameters.add(AttributeValue.S(title))
        parameters.add(AttributeValue.S(info))

        executeStatementPartiQL(ddb, sqlStatement, parameters)
        println("Added Movie $title")
        parameters.clear()
        t++
    }
}

suspend fun getMoviePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and title=?"
    val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
```

```
    parameters.add(AttributeValue.N("2012"))
    parameters.add(AttributeValue.S("The Perks of Being a Wallflower"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("ExecuteStatement successful: $response")
}

suspend fun putRecordPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2020"))
    parameters.add(AttributeValue.S("My Movie"))
    parameters.add(AttributeValue.S("No Info"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Added new movie.")
}

suspend fun updateTableItemPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":[\"Merian C.
 Cooper\",\"Ernest B. Schoedsack\' where year=? and title=?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    parameters.add(AttributeValue.S("The East"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Item was updated!")
}

// Query the table where the year is 2013.
suspend fun queryTablePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year = ?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("ExecuteStatement successful: $response")
}

suspend fun deleteTablePartiQL(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
```

```
        println("$tableNameVal was deleted")
    }
}

suspend fun executeStatementPartiQL(
    ddb: DynamoDbClient,
    statementVal: String,
    parametersVal: List<AttributeValue>,
): ExecuteStatementResponse {
    val request =
        ExecuteStatementRequest {
            statement = statementVal
            parameters = parametersVal
        }

    return ddb.executeStatement(request)
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 ExecuteStatement를 참조하세요.

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\testable_readline;
use function AwsUtilities\loadMovieData;

class GettingStartedWithPartiQL
{
```

```
    public function run()
    {
        echo("\n");
        echo("---------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
using PHP!\n");
        echo("---------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
$tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
```

```
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQL($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);
        echo "Movie added and updated.\n";



        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQL($tableName, $key);
        echo "\nThe movie {$movie['Items'][0]['title']['S']} was released in
 {$movie['Items'][0]['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Items'][0]['title']
['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
```

```
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);

        $movie = $service->getItemByPartiQL($tableName, $key);
        echo "Okay, you have rated {$movie['Items'][0]['title']['S']} as a
 {$movie['Items'][0]['rating']['N']}\n";

        $service->deleteItemByPartiQL($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
```

```
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

```
    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- API 세부 정보는 AWS SDK for PHP API 참조의 ExecuteStatement를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

PartiQL 문을 실행할 수 있는 클래스를 생성합니다.

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint

import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
```

```
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statement, params):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
the
        resource transforms input and output from plain old Python objects
(POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statement: The PartiQL statement.
        :param params: The list of PartiQL parameters. These are applied to the
                       statement in the order they are listed.
        :return: The items returned from the statement, if any.
        """
        try:
            output = self.dyn_resource.meta.client.execute_statement(
                Statement=statement, Parameters=params
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute PartiQL '%s' because the table does not
exist.",
                    statement,
                )
            else:
                logger.error(
                    "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                    statement,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

테이블을 생성하고 PartiQL 쿼리를 실행하는 시나리오를 실행합니다.

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL single statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)

    title = "24 Hour PartiQL People"
    year = datetime.now().year
    plot = "A group of data developers discover a new query language they can't
 stop using."
    rating = Decimal("9.9")

    print(f"Inserting movie '{title}' released in {year}.")
    wrapper.run_partiql(
        f"INSERT INTO \"{table_name}\" VALUE {{'title': ?, 'year': ?,
 'info': ?}}",
        [title, year, {"plot": plot, "rating": rating}],
    )
    print("Success!")
    print("-" * 88)

    print(f"Getting data for movie '{title}' released in {year}.")
    output = wrapper.run_partiql(
        f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    for item in output["Items"]:
        print(f"\n{item['title']}, {item['year']}")
        pprint(output["Items"])
    print("-" * 88)

    rating = Decimal("2.4")
    print(f"Updating movie '{title}' with a rating of {float(rating)}.")
    wrapper.run_partiql(
        f'UPDATE "{table_name}" SET info.rating=? WHERE title=? AND year=?',
```

```
            [rating, title, year],
        )
        print("Success!")
        print("-" * 88)

        print(f"Getting data again to verify our update.")
        output = wrapper.run_partiql(
            f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
        )
        for item in output["Items"]:
            print(f"\n{item['title']}, {item['year']}")
            pprint(output["Items"])
        print("-" * 88)

        print(f"Deleting movie '{title}' released in {year}.")
        wrapper.run_partiql(
            f'DELETE FROM "{table_name}" WHERE title=? AND year=?', [title, year]
        )
        print("Success!")
        print("-" * 88)

        print(f"Deleting table '{table_name}'...")
        scaffold.delete_table()
        print("-" * 88)

        print("\nThanks for watching!")
        print("-" * 88)


if __name__ == "__main__":
    try:
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 ExecuteStatement를 참조하세요.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

테이블을 생성하고 PartiQL 쿼리를 실행하는 시나리오를 실행합니다.

```
table_name = "doc-example-table-movies-partiql-#{rand(10**8)}"
scaffold = Scaffold.new(table_name)
sdk = DynamoDBPartiQLSingle.new(table_name)

new_step(1, 'Create a new DynamoDB table if none already exists.')
unless scaffold.exists?(table_name)
  puts("\nNo such table: #{table_name}. Creating it...")
  scaffold.create_table(table_name)
  print "Done!\n".green
end

new_step(2, 'Populate DynamoDB table with movie data.')
download_file = 'moviedata.json'
puts("Downloading movie database to #{download_file}...")
movie_data = scaffold.fetch_movie_data(download_file)
puts("Writing movie data from #{download_file} into your table...")
scaffold.write_batch(movie_data)
puts("Records added: #{movie_data.length}.")
print "Done!\n".green

new_step(3, 'Select a single item from the movies table.')
response = sdk.select_item_by_title('Star Wars')
puts("Items selected for title 'Star Wars': #{response.items.length}\n")
print response.items.first.to_s.yellow
print "\n\nDone!\n".green

new_step(4, 'Update a single item from the movies table.')
puts "Let's correct the rating on The Big Lebowski to 10.0."
sdk.update_rating_by_title('The Big Lebowski', 1998, 10.0)
print "\nDone!\n".green
```

```
  new_step(5, 'Delete a single item from the movies table.')
  puts "Let's delete The Silence of the Lambs because it's just too scary."
  sdk.delete_item_by_title('The Silence of the Lambs', 1991)
  print "\nDone!\n".green

  new_step(6, 'Insert a new item into the movies table.')
  puts "Let's create a less-scary movie called The Prancing of the Lambs."
  sdk.insert_item('The Prancing of the Lambs', 2005, 'A movie about happy
 livestock.', 5.0)
  print "\nDone!\n".green

  new_step(7, 'Delete the table.')
  return unless scaffold.exists?(table_name)

  scaffold.delete_table
end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 ExecuteStatement를 참조하세요.

## Rust

### SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
async fn make_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<(), SdkError<CreateTableError>> {
    let ad = AttributeDefinition::builder()
        .attribute_name(key)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .expect("creating AttributeDefinition");

    let ks = KeySchemaElement::builder()
```

```
        .attribute_name(key)
        .key_type(KeyType::Hash)
        .build()
        .expect("creating KeySchemaElement");

    match client
        .create_table()
        .table_name(table)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn add_item(client: &Client, item: Item) -> Result<(),
 SdkError<ExecuteStatementError>> {
    match client
        .execute_statement()
        .statement(format!(
            r#"INSERT INTO "{}" VALUE {{
                "{}": ?,
                "acount_type": ?,
                "age": ?,
                "first_name": ?,
                "last_name": ?
        }} "#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![
            AttributeValue::S(item.utype),
            AttributeValue::S(item.age),
            AttributeValue::S(item.first_name),
            AttributeValue::S(item.last_name),
        ]))
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
```

```
    }
}

async fn query_item(client: &Client, item: Item) -> bool {
    match client
        .execute_statement()
        .statement(format!(
            r#"SELECT * FROM "{}" WHERE "{}" = ?"#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![AttributeValue::S(item.value)]))
        .send()
        .await
    {
        Ok(resp) => {
            if !resp.items().is_empty() {
                println!("Found a matching entry in the table:");
                println!("{:?}", resp.items.unwrap_or_default().pop());
                true
            } else {
                println!("Did not find a match.");
                false
            }
        }
        Err(e) => {
            println!("Got an error querying table:");
            println!("{}", e);
            process::exit(1);
        }
    }
}

async fn remove_item(client: &Client, table: &str, key: &str, value: String) ->
 Result<(), Error> {
    client
        .execute_statement()
        .statement(format!(r#"DELETE FROM "{table}" WHERE "{key}" = ?"#))
        .set_parameters(Some(vec![AttributeValue::S(value)]))
        .send()
        .await?;

    println!("Deleted item.");

    Ok(())
```

```
}

async fn remove_table(client: &Client, table: &str) -> Result<(), Error> {
    client.delete_table().table_name(table).send().await?;

    Ok(())
}
```

- API에 대한 세부 정보는 AWS Rust용 SDK API 참조의 ExecuteStatement를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK로 글로벌 보조 인덱스를 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제는 글로벌 보조 인덱스를 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- 프라이머리 키를 사용하여 DynamoDB 테이블을 쿼리합니다.
- 글로벌 보조 인덱스(GSI)에서 대체 액세스 패턴을 쿼리합니다.
- 테이블 쿼리와 GSI 쿼리를 비교합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x로 프라이머리 키와 글로벌 보조 인덱스(GSI)를 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryTable(
```

```
        final String tableName, final String partitionKeyName, final String
partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on base table successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw new DynamoDbQueryException("Table not found: " + tableName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying base table: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on base
table", e);
        }
    }

    /**
     * Queries a DynamoDB Global Secondary Index (GSI) by partition key.
```

```
     *
     * @param tableName         The name of the DynamoDB table
     * @param indexName         The name of the GSI
     * @param partitionKeyName  The name of the GSI partition key attribute
     * @param partitionKeyValue The value of the GSI partition key to query
     * @return The query response from DynamoDB
     * @throws ResourceNotFoundException if the table or index doesn't exist
     * @throws DynamoDbException if the query fails
     */
    public QueryResponse queryGlobalSecondaryIndex(
        final String tableName, final String indexName, final String
partitionKeyName, final String partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Index name", indexName);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_IK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_IK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .indexName(indexName)
            .keyConditionExpression(GSI_KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on GSI successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
```

```
            System.err.format(
                "Error: The Amazon DynamoDB table \"%s\" or index \"%s\" can't be
found.\n", tableName, indexName);
            throw new DynamoDbQueryException("Table or index not found: " +
tableName + "/" + indexName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying GSI: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on GSI",
e);
        }
    }
```

AWS SDK for Java 2.x로 GSI를 쿼리하는 것과 테이블을 직접 쿼리하는 것을 비교해 보세요.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <basePartitionKeyName> <basePartitionKeyValue>
<gsiName> <gsiPartitionKeyName> <gsiPartitionKeyValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    basePartitionKeyName - The name of the base table partition
key attribute.
                    basePartitionKeyValue - The value of the base table partition
key to query.
                    gsiName - The name of the Global Secondary Index.
                    gsiPartitionKeyName - The name of the GSI partition key
attribute.
                    gsiPartitionKeyValue - The value of the GSI partition key to
query.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 6) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String basePartitionKeyName = args[1];
```

```
        final String basePartitionKeyValue = args[2];
        final String gsiName = args[3];
        final String gsiPartitionKeyName = args[4];
        final String gsiPartitionKeyValue = args[5];
        final Region region = args.length > 6 ? Region.of(args[6]) :
Region.US_EAST_1;

        try (DynamoDbClient ddb =
DynamoDbClient.builder().region(region).build()) {
            final QueryTableAndGSI queryHelper = new QueryTableAndGSI(ddb);

            // Query the base table
            System.out.println("Querying base table where " +
basePartitionKeyName + " = " + basePartitionKeyValue);
            final QueryResponse tableResponse =
                queryHelper.queryTable(tableName, basePartitionKeyName,
basePartitionKeyValue);

            System.out.println("Found " + tableResponse.count() + " items in base
table:");
            tableResponse.items().forEach(item -> System.out.println(item));

            // Query the GSI
            System.out.println(
                "\nQuerying GSI '" + gsiName + "' where " + gsiPartitionKeyName +
" = " + gsiPartitionKeyValue);
            final QueryResponse gsiResponse =
                queryHelper.queryGlobalSecondaryIndex(tableName, gsiName,
gsiPartitionKeyName, gsiPartitionKeyValue);

            System.out.println("Found " + gsiResponse.count() + " items in
GSI:");
            gsiResponse.items().forEach(item -> System.out.println(item));

            // Explain the differences between querying a table and a GSI
            System.out.println("\nKey differences between querying a table and a
GSI:");
            System.out.println("1. When querying a GSI, you must specify the
indexName parameter");
            System.out.println("2. GSIs may not contain all attributes from the
base table (projection)");
            System.out.println("3. GSIs consume read capacity units from the
GSI's capacity, not the base table's");
```

```
            System.out.println("4. GSIs may have eventually consistent data
 (cannot use ConsistentRead=true)");

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table or index not found: " + e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript로 프라이머리 키를 사용하여 DynamoDB 테이블을 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
```

```
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
  }
}
```

AWS SDK for JavaScript로 DynamoDB 글로벌 보조 인덱스(GSI)를 쿼리합니다.

```
/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);
```

```
    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)로 프라이머리 키와 글로벌 보조 인덱스(GSI)를 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import boto3
from boto3.dynamodb.conditions import Key


def query_table(table_name, partition_key_name, partition_key_value):
    """
    Query a DynamoDB table using its primary key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.

    Returns:
```

```
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the table's primary key
    response =
 table.query(KeyConditionExpression=Key(partition_key_name).eq(partition_key_value))

    return response


def query_gsi(table_name, index_name, partition_key_name, partition_key_value):
    """
    Query a Global Secondary Index (GSI) on a DynamoDB table.

    Args:
        table_name (str): The name of the DynamoDB table.
        index_name (str): The name of the Global Secondary Index.
        partition_key_name (str): The name of the GSI's partition key attribute.
        partition_key_value (str): The value of the GSI's partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the GSI
    response = table.query(
        IndexName=index_name,
 KeyConditionExpression=Key(partition_key_name).eq(partition_key_value)
    )

    return response
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK로 begins_with 조건을 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제에서는 begins_with 조건을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- 키 조건 표현식에서 begins_with 함수를 사용합니다.
- 정렬 키의 접두사 패턴을 기준으로 항목을 필터링합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x로 정렬 키에 begins_with 조건을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithBeginsWithCondition(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String sortKeyName,
        final String sortKeyPrefix) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Sort key name", sortKeyName);
        CodeSampleUtils.validateStringParameter("Sort key prefix",
 sortKeyPrefix);
```

```
        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, sortKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_SK_PREFIX,
            AttributeValue.builder().s(sortKeyPrefix).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with begins_with condition successful.
Found {0} items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with begins_with condition",
e);
            throw e;
        }
    }
```

AWS SDK for Java 2.x로 다양한 접두사 길이의 begins_with를 사용하는 방법을 보여줍니다.

```
    public static void main(String[] args) {
```

```
        try {
            CodeSampleUtils.BeginsWithQueryConfig config =
CodeSampleUtils.BeginsWithQueryConfig.fromArgs(args);
            LOGGER.log(Level.INFO, "Querying items where {0} = {1} and {2} begins
with ''{3}''", new Object[] {
                config.getPartitionKeyName(),
                config.getPartitionKeyValue(),
                config.getSortKeyName(),
                config.getSortKeyPrefix()
            });

            // Using the builder pattern to create and execute the query
            final QueryResponse response = new BeginsWithQueryBuilder()
                .withTableName(config.getTableName())
                .withPartitionKeyName(config.getPartitionKeyName())
                .withPartitionKeyValue(config.getPartitionKeyValue())
                .withSortKeyName(config.getSortKeyName())
                .withSortKeyPrefix(config.getSortKeyPrefix())
                .withRegion(config.getRegion())
                .execute();

            // Process the results
            LOGGER.log(Level.INFO, "Found {0} items:", response.count());
            response.items().forEach(item -> LOGGER.info(item.toString()));

            // Demonstrate with a different prefix
            if (!config.getSortKeyPrefix().isEmpty()) {
                String shorterPrefix = config.getSortKeyPrefix()
                    .substring(0, Math.max(1,
config.getSortKeyPrefix().length() / 2));
                LOGGER.log(Level.INFO, "\nNow querying with a shorter prefix:
''{0}''", shorterPrefix);

                final QueryResponse response2 = new BeginsWithQueryBuilder()
                    .withTableName(config.getTableName())
                    .withPartitionKeyName(config.getPartitionKeyName())
                    .withPartitionKeyValue(config.getPartitionKeyValue())
                    .withSortKeyName(config.getSortKeyName())
                    .withSortKeyPrefix(shorterPrefix)
                    .withRegion(config.getRegion())
                    .execute();

                LOGGER.log(Level.INFO, "Found {0} items with shorter prefix:",
response2.count());
```

```
                response2.items().forEach(item -> LOGGER.info(item.toString()));
            }
        } catch (IllegalArgumentException e) {
            LOGGER.log(Level.SEVERE, "Invalid input: {0}", e.getMessage());
            printUsage();
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found", e);
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "DynamoDB error", e);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Unexpected error", e);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

AWS SDK for JavaScript로 정렬 키에 begins_with 조건을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items where the sort key begins with a specific
 prefix
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key
 * @param {string} prefix - The prefix to match at the beginning of the sort key
 * @returns {Promise<Object>} - The query response
 */
async function queryWithBeginsWith(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
```

```
  sortKeyName,
  prefix
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND begins_with(#sk, :prefix)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":prefix": { S: prefix }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with begins_with: ${error}`);
    throw error;
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)로 정렬 키에 begins_with 조건을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import boto3
from boto3.dynamodb.conditions import Key
```

```
def query_with_begins_with(
    table_name, partition_key_name, partition_key_value, sort_key_name, prefix
):
    """
    Query a DynamoDB table with a begins_with condition on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute.
        prefix (str): The prefix to match at the beginning of the sort key.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query with a begins_with condition on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) & Key(
        sort_key_name
    ).begins_with(prefix)
    response = table.query(KeyConditionExpression=key_condition)

    return response
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK로 정렬 키에 날짜 범위를 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제에서는 정렬 키에 날짜 범위를 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- 특정 날짜 범위 내의 항목을 쿼리합니다.

- 날짜 형식 정렬 키에 비교 연산자를 사용합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x로 DynamoDB 테이블에서 특정 날짜 범위 내의 항목을 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.LocalDate;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithDateRange(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDate startDate,
        final LocalDate endDate) {

        // Focus on query logic, assuming parameters are valid
        if (startDate == null || endDate == null) {
            throw new IllegalArgumentException("Start date and end date cannot be
 null");
        }

        if (endDate.isBefore(startDate)) {
            throw new IllegalArgumentException("End date must be after start
 date");
        }

        // Format dates as ISO strings for DynamoDB (using just the date part)
        final String formattedStartDate = startDate.toString();
```

```
        final String formattedEndDate = endDate.toString();

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_START_DATE,
            AttributeValue.builder().s(formattedStartDate).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_END_DATE,
            AttributeValue.builder().s(formattedEndDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query by date range successful. Found {0}
items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying by date range: {0}",
e.getMessage());
            throw e;
        }
    }
```

날짜 범위 필터링을 사용하여 DynamoDB 테이블을 쿼리하는 방법을 보여줍니다.

```
   public static void main(String[] args) {
       final String usage =
           """
               Usage:
                   <tableName> <partitionKeyName> <partitionKeyValue>
<dateKeyName> <startDate> <endDate> [region]
               Where:
                   tableName - The Amazon DynamoDB table to query.
                   partitionKeyName - The name of the partition key attribute.
                   partitionKeyValue - The value of the partition key to query.
                   dateKeyName - The name of the date attribute to filter on.
                   startDate - The start date for the range query (YYYY-MM-DD).
                   endDate - The end date for the range query (YYYY-MM-DD).
                   region (optional) - The AWS region where the table exists.
(Default: us-east-1)
               """;

       if (args.length < 6) {
           System.out.println(usage);
           System.exit(1);
       }

       try {
           // Parse command line arguments into a config object
           CodeSampleUtils.DateRangeQueryConfig config =
CodeSampleUtils.DateRangeQueryConfig.fromArgs(args);

           LOGGER.log(
               Level.INFO, "Querying items from {0} to {1}", new Object[]
{config.getStartDate(), config.getEndDate()
               });

           // Using the builder pattern to create and execute the query
           final QueryResponse response = new DateRangeQueryBuilder()
               .withTableName(config.getTableName())
               .withPartitionKeyName(config.getPartitionKeyName())
               .withPartitionKeyValue(config.getPartitionKeyValue())
               .withDateKeyName(config.getDateKeyName())
               .withStartDate(config.getStartDate())
```

```
                .withEndDate(config.getEndDate())
                .withRegion(config.getRegion())
                .execute();

            // Process the results
            LOGGER.log(Level.INFO, "Found {0} items:", response.count());
            response.items().forEach(item -> {
                LOGGER.info(item.toString());

                // Extract and display the date attribute for clarity
                if (item.containsKey(config.getDateKeyName())) {
                    LOGGER.log(
                        Level.INFO,
                        "  Date attribute: {0}",
                        item.get(config.getDateKeyName()).s());
                }
            });

            // Demonstrate with a different date range
            LocalDate narrowerStartDate = config.getStartDate().plusDays(1);
            LocalDate narrowerEndDate = config.getEndDate().minusDays(1);

            if (!narrowerStartDate.isAfter(narrowerEndDate)) {
                LOGGER.log(Level.INFO, "\nNow querying with a narrower date
range: {0} to {1}", new Object[] {
                    narrowerStartDate, narrowerEndDate
                });

                final QueryResponse response2 = new DateRangeQueryBuilder()
                    .withTableName(config.getTableName())
                    .withPartitionKeyName(config.getPartitionKeyName())
                    .withPartitionKeyValue(config.getPartitionKeyValue())
                    .withDateKeyName(config.getDateKeyName())
                    .withStartDate(narrowerStartDate)
                    .withEndDate(narrowerEndDate)
                    .withRegion(config.getRegion())
                    .execute();

                LOGGER.log(Level.INFO, "Found {0} items with narrower date
range:", response2.count());
                response2.items().forEach(item -> LOGGER.info(item.toString()));
            }

            LOGGER.info("\nNote: When storing dates in DynamoDB:");
```

```
            LOGGER.info("1. Use ISO format (YYYY-MM-DD) for lexicographical
 ordering");
            LOGGER.info("2. Use the BETWEEN operator for inclusive date range
 queries");
            LOGGER.info("3. Consider using ISO-8601 format for timestamps with
 time components");

        } catch (IllegalArgumentException e) {
            LOGGER.log(Level.SEVERE, "Invalid input: {0}", e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "DynamoDB error: {0}", e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Unexpected error: {0}", e.getMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

AWS SDK for JavaScript로 DynamoDB 테이블에서 특정 날짜 범위 내의 항목을 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort
 key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
```

```
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
```

```
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)로 DynamoDB 테이블에서 특정 날짜 범위 내의 항목을 쿼리합니다.

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_date_range(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        start_date (datetime): The start date for the query range.
        end_date (datetime): The end date for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date values as ISO 8601 strings
    # DynamoDB works well with ISO format for date values
```

```
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key using BETWEEN operator
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response


def query_with_date_range_by_month(
    table_name, partition_key_name, partition_key_value, sort_key_name, year,
 month
):
    """
    Query a DynamoDB table for a specific month's data.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        year (int): The year to query.
        month (int): The month to query (1-12).

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Calculate the start and end dates for the specified month
    if month == 12:
        next_year = year + 1
        next_month = 1
    else:
        next_year = year
        next_month = month + 1

    start_date = datetime(year, month, 1)
    end_date = datetime(next_year, next_month, 1) - timedelta(microseconds=1)

    # Format the date values as ISO 8601 strings
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(KeyConditionExpression=key_condition)

    return response
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK로 복잡한 필터 표현식을 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제에서는 복잡한 필터 표현식을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- 쿼리 결과에 복잡한 필터 표현식을 적용합니다.
- 논리적 연산자를 사용하여 여러 조건을 결합합니다.
- 키가 아닌 속성을 기준으로 항목을 필터링합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x로 복잡한 필터 표현식을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithComplexFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String statusAttrName,
        final String activeStatus,
        final String pendingStatus,
        final String priceAttrName,
        final double minPrice,
        final double maxPrice,
        final String categoryAttrName) {

        // Validate parameters
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Status attribute name",
 statusAttrName);
        CodeSampleUtils.validateStringParameter("Active status", activeStatus);
        CodeSampleUtils.validateStringParameter("Pending status", pendingStatus);
        CodeSampleUtils.validateStringParameter("Price attribute name",
 priceAttrName);
        CodeSampleUtils.validateStringParameter("Category attribute name",
 categoryAttrName);
        CodeSampleUtils.validateNumericRange("Minimum price", minPrice, 0.0,
 Double.MAX_VALUE);
```

```
        CodeSampleUtils.validateNumericRange("Maximum price", maxPrice, minPrice,
Double.MAX_VALUE);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pk", partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_STATUS,
statusAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PRICE,
priceAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_CATEGORY,
categoryAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":pkValue", AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_ACTIVE,
            AttributeValue.builder().s(activeStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PENDING,
            AttributeValue.builder().s(pendingStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MIN_PRICE,
            AttributeValue.builder().n(String.valueOf(minPrice)).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MAX_PRICE,
            AttributeValue.builder().n(String.valueOf(maxPrice)).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        return dynamoDbClient.query(queryRequest);
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript로 복잡한 필터 표현식을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
```

```
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)로 복잡한 필터 표현식을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_complex_filter(
    table_name,
    partition_key_name,
    partition_key_value,
    min_rating=None,
    status_list=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression.
```

```
    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        min_rating (float, optional): Minimum rating value for filtering.
        status_list (list, optional): List of status values to include.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize the filter expression and expression attribute values
    filter_expression = None
    expression_attribute_values = {}

    # Build the filter expression based on provided parameters
    if min_rating is not None:
        filter_expression = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if status_list and len(status_list) > 0:
        status_condition = None
        for i, status in enumerate(status_list):
            status_value_name = f":status{i}"
            expression_attribute_values[status_value_name] = status

            if status_condition is None:
                status_condition = Attr("status").eq(status)
            else:
                status_condition = status_condition | Attr("status").eq(status)

        if filter_expression is None:
            filter_expression = status_condition
        else:
            filter_expression = filter_expression & status_condition

    if max_price is not None:
```

```
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if filter_expression is None:
            filter_expression = price_condition
        else:
            filter_expression = filter_expression & price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
 expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response


def query_with_complex_filter_and_or(
    table_name,
    partition_key_name,
    partition_key_value,
    category=None,
    min_rating=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression using AND and OR
 operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        category (str, optional): Category value for filtering.
        min_rating (float, optional): Minimum rating value for filtering.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
```

```
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build a complex filter expression with AND and OR operators
    filter_expression = None
    expression_attribute_values = {}

    # Build the category condition
    if category:
        filter_expression = Attr("category").eq(category)
        expression_attribute_values[":category"] = category

    # Build the rating and price condition (rating >= min_rating OR price <=
max_price)
    rating_price_condition = None

    if min_rating is not None:
        rating_price_condition = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if rating_price_condition is None:
            rating_price_condition = price_condition
        else:
            rating_price_condition = rating_price_condition | price_condition

    # Combine the conditions
    if rating_price_condition:
        if filter_expression is None:
            filter_expression = rating_price_condition
        else:
            filter_expression = filter_expression & rating_price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}
```

```
    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK로 동적 필터 표현식을 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제에서는 동적 필터 표현식을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- 런타임에 동적으로 필터 표현식을 작성합니다.
- 사용자 입력 또는 애플리케이션 상태를 기반으로 필터 조건을 구성합니다.
- 조건부로 필터 기준을 추가하거나 제거합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x로 동적으로 구성된 필터 표현식을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
```

```
import java.util.HashMap;
import java.util.Map;

    public static QueryResponse queryWithDynamicFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final Map<String, Object> filterCriteria,
        final Region region,
        final DynamoDbClient dynamoDbClient) {

        validateParameters(tableName, partitionKeyName, partitionKeyValue,
 filterCriteria);

        DynamoDbClient ddbClient = dynamoDbClient;
        boolean shouldClose = false;

        try {
            if (ddbClient == null) {
                ddbClient = createClient(region);
                shouldClose = true;
            }

            final QueryWithDynamicFilter queryHelper = new
 QueryWithDynamicFilter(ddbClient);
            return queryHelper.queryWithDynamicFilter(tableName,
 partitionKeyName, partitionKeyValue, filterCriteria);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Failed to execute dynamic filter query: " +
 e.getMessage());
            throw e;
        } catch (Exception e) {
            System.err.println("Unexpected error during query: " +
 e.getMessage());
            throw e;
        } finally {
            if (shouldClose && ddbClient != null) {
                ddbClient.close();
            }
        }
    }
```

AWS SDK for Java 2.x로 동적 필터 표현식을 사용하는 방법을 보여줍니다.

```
   public static void main(String[] args) {
       final String usage =
           """
               Usage:
                   <tableName> <partitionKeyName> <partitionKeyValue>
<filterAttrName> <filterAttrValue> [region]
               Where:
                   tableName - The Amazon DynamoDB table to query.
                   partitionKeyName - The name of the partition key attribute.
                   partitionKeyValue - The value of the partition key to query.
                   filterAttrName - The name of the attribute to filter on.
                   filterAttrValue - The value to filter by.
                   region (optional) - The AWS region where the table exists.
(Default: us-east-1)
               """;

       if (args.length < 5) {
           System.out.println(usage);
           System.exit(1);
       }

       final String tableName = args[0];
       final String partitionKeyName = args[1];
       final String partitionKeyValue = args[2];
       final String filterAttrName = args[3];
       final String filterAttrValue = args[4];
       final Region region = args.length > 5 ? Region.of(args[5]) :
Region.US_EAST_1;

       System.out.println("Querying items with dynamic filter: " +
filterAttrName + " = " + filterAttrValue);

       try {
           // Using the builder pattern to create and execute the query
           final QueryResponse response = new DynamicFilterQueryBuilder()
               .withTableName(tableName)
               .withPartitionKeyName(partitionKeyName)
               .withPartitionKeyValue(partitionKeyValue)
               .withFilterCriterion(filterAttrName, filterAttrValue)
```

```
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> System.out.println(item));

            // Demonstrate multiple filter criteria
            System.out.println("\nNow querying with multiple filter criteria:");

            Map<String, Object> multipleFilters = new HashMap<>();
            multipleFilters.put(filterAttrName, filterAttrValue);
            multipleFilters.put("status", "active");

            final QueryResponse multiFilterResponse = new
DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriteria(multipleFilters)
                .withRegion(region)
                .execute();

            System.out.println("Found " + multiFilterResponse.count() + " items
with multiple filters:");
            multiFilterResponse.items().forEach(item ->
System.out.println(item));

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript로 동적으로 구성된 필터 표현식을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
```

```
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
 filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)로 동적으로 구성된 필터 표현식을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_dynamic_filter(
    table_name, partition_key_name, partition_key_value, filter_conditions=None
):
    """
    Query a DynamoDB table with a dynamically constructed filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_conditions (dict, optional): A dictionary of filter conditions
 where
            keys are attribute names and values are dictionaries with 'operator'
 and 'value'.
            Example: {'rating': {'operator': '>=', 'value': 4}, 'status':
 {'operator': '=', 'value': 'active'}}

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize variables for the filter expression and attribute values
    filter_expression = None
    expression_attribute_values = {":pk_val": partition_key_value}

    # Dynamically build the filter expression if filter conditions are provided
    if filter_conditions:
```

```
        for attr_name, condition in filter_conditions.items():
            operator = condition.get("operator")
            value = condition.get("value")
            attr_value_name = f":{attr_name}"
            expression_attribute_values[attr_value_name] = value

            # Create the appropriate filter expression based on the operator
            current_condition = None
            if operator == "=":
                current_condition = Attr(attr_name).eq(value)
            elif operator == "!=":
                current_condition = Attr(attr_name).ne(value)
            elif operator == ">":
                current_condition = Attr(attr_name).gt(value)
            elif operator == ">=":
                current_condition = Attr(attr_name).gte(value)
            elif operator == "<":
                current_condition = Attr(attr_name).lt(value)
            elif operator == "<=":
                current_condition = Attr(attr_name).lte(value)
            elif operator == "contains":
                current_condition = Attr(attr_name).contains(value)
            elif operator == "begins_with":
                current_condition = Attr(attr_name).begins_with(value)

            # Combine with existing filter expression using AND
            if current_condition:
                if filter_expression is None:
                    filter_expression = current_condition
                else:
                    filter_expression = filter_expression & current_condition

    # Perform the query with the dynamically built filter expression
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression

    response = table.query(**query_params)
    return response
```

AWS SDK for Python (Boto3)로 동적 필터 표현식을 사용하는 방법을 보여줍니다.

```
def example_usage():
    """Example of how to use the query_with_dynamic_filter function."""
    # Example parameters
    table_name = "Products"
    partition_key_name = "Category"
    partition_key_value = "Electronics"

    # Define dynamic filter conditions based on user input or runtime conditions
    user_min_rating = 4  # This could come from user input
    user_status_filter = "active"  # This could come from user input

    filter_conditions = {}

    # Only add conditions that are actually specified
    if user_min_rating is not None:
        filter_conditions["rating"] = {"operator": ">=", "value":
 user_min_rating}

    if user_status_filter:
        filter_conditions["status"] = {"operator": "=", "value":
 user_status_filter}

    print(
        f"Querying products in category '{partition_key_value}' with filter
 conditions: {filter_conditions}"
    )

    # Execute the query with dynamic filter
    response = query_with_dynamic_filter(
        table_name, partition_key_name, partition_key_value, filter_conditions
    )

    # Process the results
    items = response.get("Items", [])
    print(f"Found {len(items)} items")

    for item in items:
        print(f"Product: {item}")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK로 필터 표현식 및 제한을 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제에서는 필터 표현식 및 제한을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- 평가되는 항목에 제한을 두어 쿼리 결과에 필터 표현식을 적용합니다.
- 제한이 필터링된 쿼리 결과에 미치는 영향을 이해합니다.
- 쿼리에서 처리되는 최대 항목 수를 제어합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x로 필터 표현식 및 제한을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithFilterAndLimit(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String filterAttrName,
        final String filterAttrValue,
        final int limit) {
```

```
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Filter attribute name",
filterAttrName);
        CodeSampleUtils.validateStringParameter("Filter attribute value",
filterAttrValue);
        CodeSampleUtils.validatePositiveInteger("Limit", limit);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_FILTER,
filterAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_FILTER,
            AttributeValue.builder().s(filterAttrValue).build());

        // Create the filter expression
        final String filterExpression = "#filterAttr = :filterValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(limit)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with filter and limit successful. Found
{0} items", response.count());
            LOGGER.log(
```

```
                Level.INFO, "ScannedCount: {0} (total items evaluated before
 filtering)", response.scannedCount());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with filter and limit: {0}",
 e.getMessage());
            throw e;
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)로 필터 표현식 및 제한을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_filter_and_limit(
    table_name,
    partition_key_name,
    partition_key_value,
    filter_attribute=None,
    filter_value=None,
    limit=10,
):
    """
    Query a DynamoDB table with a filter expression and limit the number of
 results.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
```

```
        filter_attribute (str, optional): The attribute name to filter on.
        filter_value (any, optional): The value to compare against in the filter.
        limit (int, optional): The maximum number of items to evaluate. Defaults
 to 10.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition, "Limit": limit}

    # Add the filter expression if filter attributes are provided
    if filter_attribute and filter_value is not None:
        query_params["FilterExpression"] =
 Attr(filter_attribute).gt(filter_value)
        query_params["ExpressionAttributeValues"] = {":filter_value":
 filter_value}

    # Execute the query
    response = table.query(**query_params)
    return response
```

AWS SDK for Python (Boto3)으로 동적 필터 표현식과 제한을 사용하는 방법을 보여줍니다.

```
def example_usage():
    """Example of how to use the query_with_filter_and_limit function."""
    # Example parameters
    table_name = "ProductReviews"
    partition_key_name = "ProductId"
    partition_key_value = "P123456"
    filter_attribute = "Rating"
    filter_value = 3  # Filter for ratings > 3
    limit = 5
```

```
    print(f"Querying reviews for product '{partition_key_value}' with rating >
{filter_value}")
    print(f"Limiting to {limit} evaluated items")

    # Execute the query with filter and limit
    response = query_with_filter_and_limit(
        table_name, partition_key_name, partition_key_value, filter_attribute,
filter_value, limit
    )

    # Process the results
    items = response.get("Items", [])
    print(f"\nReturned {len(items)} items that passed the filter")

    for item in items:
        print(f"Review: {item}")

    # Explain the difference between Limit and actual results
    explain_limit_vs_results(response)

    # Check if there are more results
    if "LastEvaluatedKey" in response:
        print("\nThere are more results available. Use the LastEvaluatedKey for
pagination.")
    else:
        print("\nAll matching results have been retrieved.")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK로 중첩 속성을 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제에서는 중첩 속성을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- DynamoDB 항목의 중첩 속성을 기준으로 액세스하고 필터링합니다.
- 문서 경로 표현식을 사용하여 중첩된 요소를 참조합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x로 중첩 속성을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithNestedAttributes(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String nestedPath,
        final String nestedAttr,
        final String nestedValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Nested path", nestedPath);
        CodeSampleUtils.validateStringParameter("Nested attribute", nestedAttr);
        CodeSampleUtils.validateStringParameter("Nested value", nestedValue);

        // Split the nested path into components
        final String[] pathComponents = nestedPath.split("\\.");

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Build the nested attribute reference using document path notation
        final StringBuilder nestedAttributeRef = new StringBuilder();
        for (int i = 0; i < pathComponents.length; i++) {
            final String aliasName = "#n" + i;
            expressionAttributeNames.put(aliasName, pathComponents[i]);
```

```
            if (i > 0) {
                nestedAttributeRef.append(".");
            }
            nestedAttributeRef.append(aliasName);
        }

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_NESTED,
            AttributeValue.builder().s(nestedValue).build());

        // Create the filter expression using the nested attribute reference
        final String filterExpression = nestedAttributeRef + " = :nestedValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query with nested attribute filter successful.
Found " + response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with nested attribute filter: " +
e.getMessage());
            throw e;
        }
    }
```

중첩 속성을 사용하여 DynamoDB 테이블을 쿼리하는 방법을 보여줍니다.

```
   public static void main(String[] args) {
       final String usage =
           """
               Usage:
                   <tableName> <partitionKeyName> <partitionKeyValue>
<nestedPath> <nestedAttr> <nestedValue> [region]
               Where:
                   tableName - The Amazon DynamoDB table to query.
                   partitionKeyName - The name of the partition key attribute.
                   partitionKeyValue - The value of the partition key to query.
                   nestedPath - The path to the nested map attribute (e.g.,
"address").
                   nestedAttr - The name of the nested attribute (e.g., "city").
                   nestedValue - The value to filter by (e.g., "Seattle").
                   region (optional) - The AWS region where the table exists.
(Default: us-east-1)
               """;

       if (args.length < 6) {
           System.out.println(usage);
           System.exit(1);
       }

       final String tableName = args[0];
       final String partitionKeyName = args[1];
       final String partitionKeyValue = args[2];
       final String nestedPath = args[3];
       final String nestedAttr = args[4];
       final String nestedValue = args[5];
       final Region region = args.length > 6 ? Region.of(args[6]) :
Region.US_EAST_1;

       System.out.println("Querying items where " + partitionKeyName + " = " +
partitionKeyValue + " and " + nestedPath
           + "." + nestedAttr + " = " + nestedValue);

       try {
           // Using the builder pattern to create and execute the query
           final QueryResponse response = new NestedAttributeQueryBuilder()
```

```
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withNestedPath(nestedPath)
                .withNestedAttribute(nestedAttr)
                .withNestedValue(nestedValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> {
                System.out.println(item);

                // Extract and display the nested attribute for clarity
                if (item.containsKey(nestedPath) && item.get(nestedPath).hasM())
{
                    Map<String, AttributeValue> nestedMap =
item.get(nestedPath).m();
                    if (nestedMap.containsKey(nestedAttr)) {
                        System.out.println("  Nested attribute " + nestedPath +
"." + nestedAttr + ": "
                                + formatAttributeValue(nestedMap.get(nestedAttr)));
                    }
                }
            });

            System.out.println("\nNote: When working with nested attributes in
DynamoDB:");
            System.out.println("1. Use dot notation in filter expressions to
access nested attributes");
            System.out.println("2. Use expression attribute names for each
component of the path");
            System.out.println("3. Check if the nested attribute exists before
accessing it");

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
```

```
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript로 중첩 속성을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table filtering on a nested attribute
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} productId - The product ID to query by (partition key)
 * @param {string} category - The category to filter by (nested attribute)
 * @returns {Promise<Object>} - The query response
 */
async function queryWithNestedAttribute(
  config,
  tableName,
  productId,
  category
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "product_id = :productId",
      FilterExpression: "details.category = :category",
      ExpressionAttributeValues: {
```

```
        ":productId": { S: productId },
        ":category": { S: category }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with nested attribute: ${error}`);
    throw error;
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)로 중첩 속성을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
from typing import Any, Dict, List

import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_nested_attributes(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    nested_path: str,
    comparison_operator: str,
    comparison_value: Any,
) -> Dict[str, Any]:
    """
    Query a DynamoDB table and filter by nested attributes.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
```

```
        partition_key_value (str): The value of the partition key to query.
        nested_path (str): The path to the nested attribute (e.g.,
'specs.weight').
        comparison_operator (str): The comparison operator to use ('=', '!=',
'<', '<=', '>', '>=').
        comparison_value (any): The value to compare against.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build the filter expression based on the nested attribute path and
comparison operator
    filter_expression = None
    if comparison_operator == "=":
        filter_expression = Attr(nested_path).eq(comparison_value)
    elif comparison_operator == "!=":
        filter_expression = Attr(nested_path).ne(comparison_value)
    elif comparison_operator == "<":
        filter_expression = Attr(nested_path).lt(comparison_value)
    elif comparison_operator == "<=":
        filter_expression = Attr(nested_path).lte(comparison_value)
    elif comparison_operator == ">":
        filter_expression = Attr(nested_path).gt(comparison_value)
    elif comparison_operator == ">=":
        filter_expression = Attr(nested_path).gte(comparison_value)
    elif comparison_operator == "contains":
        filter_expression = Attr(nested_path).contains(comparison_value)
    elif comparison_operator == "begins_with":
        filter_expression = Attr(nested_path).begins_with(comparison_value)

    # Execute the query with the filter expression
    response = table.query(KeyConditionExpression=key_condition,
FilterExpression=filter_expression)

    return response
```

```
def query_with_multiple_nested_attributes(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    nested_conditions: List[Dict[str, Any]],
) -> Dict[str, Any]:
    """
    Query a DynamoDB table and filter by multiple nested attributes.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        nested_conditions (list): A list of dictionaries, each containing:
            - path (str): The path to the nested attribute
            - operator (str): The comparison operator
            - value (any): The value to compare against

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build the combined filter expression for all nested attributes
    combined_filter = None

    for condition in nested_conditions:
        if not isinstance(condition, dict):
            continue
        path = condition.get("path", "")
        operator = condition.get("operator", "")
        value = condition.get("value")

        if not path or not operator:
            continue

        # Build the individual filter expression
        current_filter = None
        if operator == "=":
```

```
            current_filter = Attr(path).eq(value)
        elif operator == "!=":
            current_filter = Attr(path).ne(value)
        elif operator == "<":
            current_filter = Attr(path).lt(value)
        elif operator == "<=":
            current_filter = Attr(path).lte(value)
        elif operator == ">":
            current_filter = Attr(path).gt(value)
        elif operator == ">=":
            current_filter = Attr(path).gte(value)
        elif operator == "contains":
            current_filter = Attr(path).contains(value)
        elif operator == "begins_with":
            current_filter = Attr(path).begins_with(value)

        # Combine with the existing filter using AND
        if current_filter:
            if combined_filter is None:
                combined_filter = current_filter
            else:
                combined_filter = combined_filter & current_filter

    # Execute the query with the combined filter expression
    response = table.query(KeyConditionExpression=key_condition,
FilterExpression=combined_filter)

    return response
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK로 페이지 매김을 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제에서는 페이지 매김을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- DynamoDB 쿼리 결과에 페이지 매김을 구현합니다.

- LastEvaluatedKey를 사용하여 후속 페이지를 검색합니다.
- Limit 파라미터를 사용하여 페이지당 항목 수를 제어합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x로 페이지 매김을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    public List<Map<String, AttributeValue>> queryWithPagination(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue, final int pageSize) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validatePositiveInteger("Page size", pageSize);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
```

```
        // Create the query request
        QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(pageSize);

        // List to store all items from all pages
        final List<Map<String, AttributeValue>> allItems = new ArrayList<>();

        // Map to store the last evaluated key for pagination
        Map<String, AttributeValue> lastEvaluatedKey = null;
        int pageNumber = 1;

        try {
            do {
                // If we have a last evaluated key, use it for the next page
                if (lastEvaluatedKey != null) {
                    queryRequestBuilder.exclusiveStartKey(lastEvaluatedKey);
                }

                // Execute the query
                final QueryResponse response =
dynamoDbClient.query(queryRequestBuilder.build());

                // Process the current page of results
                final List<Map<String, AttributeValue>> pageItems =
response.items();
                allItems.addAll(pageItems);

                // Get the last evaluated key for the next page
                lastEvaluatedKey = response.lastEvaluatedKey();
                if (lastEvaluatedKey != null && lastEvaluatedKey.isEmpty()) {
                    lastEvaluatedKey = null;
                }

                System.out.println("Page " + pageNumber + ": Retrieved " +
pageItems.size() + " items (Running total: "
                    + allItems.size() + ")");

                pageNumber++;

            } while (lastEvaluatedKey != null);
```

```
            System.out.println("Query with pagination complete. Retrieved a total
of " + allItems.size()
                + " items across " + (pageNumber - 1) + " pages");

            return allItems;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with pagination: " +
e.getMessage());
            throw e;
        }
    }
```

페이지 매김을 사용하여 DynamoDB 테이블을 쿼리하는 방법을 보여줍니다.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue> [pageSize]
[region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    pageSize (optional) - The maximum number of items to return
per page. (Default: 10)
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 3) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
```

```
        final String partitionKeyValue = args[2];
        final int pageSize = args.length > 3 ? Integer.parseInt(args[3]) : 10;
        final Region region = args.length > 4 ? Region.of(args[4]) :
Region.US_EAST_1;

        System.out.println("Querying items with pagination (page size: " +
pageSize + ")");

        try {
            // Using the builder pattern to create and execute the query
            final List<Map<String, AttributeValue>> allItems = new
PaginationQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withPageSize(pageSize)
                .withRegion(region)
                .executeWithPagination();

            // Process the results
            System.out.println("\nSummary: Retrieved a total of " +
allItems.size() + " items");

            // Display the first few items as a sample
            final int sampleSize = Math.min(5, allItems.size());
            if (sampleSize > 0) {
                System.out.println("\nSample of retrieved items (first " +
sampleSize + "):");
                for (int i = 0; i < sampleSize; i++) {
                    System.out.println(allItems.get(i));
                }

                if (allItems.size() > sampleSize) {
                    System.out.println("... and " + (allItems.size() -
sampleSize) + " more items");
                }
            }
        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
```

```
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript로 페이지 매김을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
```

```
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };

      // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
      if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
      }

      // Execute the query
      const command = new QueryCommand(input);
      const response = await client.send(command);

      // Process the current page of results
      pageCount++;
      console.log(`Processing page ${pageCount} with ${response.Items.length}
 items`);

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
```

```
        allItems.push(...response.Items);
      }

      // Get the LastEvaluatedKey for the next page
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages

    console.log(`Query complete. Retrieved ${allItems.length} items in
 ${pageCount} pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to
 retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)로 페이지 매김을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_pagination(
    table_name, partition_key_name, partition_key_value, page_size=25,
 max_pages=None
):
    """
    Query a DynamoDB table with pagination to handle large result sets.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.
        max_pages (int, optional): The maximum number of pages to retrieve. If
 None, retrieves all pages.

    Returns:
        list: All items retrieved from the query across all pages.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_count = 0
    all_items = []

    # Paginate through the results
    while True:
        # Check if we've reached the maximum number of pages
        if max_pages is not None and page_count >= max_pages:
            break
```

```
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
 Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Process the current page of results
        items = response.get("Items", [])
        all_items.extend(items)

        # Update pagination tracking
        page_count += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if not last_evaluated_key:
            break

    return all_items


def query_with_pagination_generator(
    table_name, partition_key_name, partition_key_value, page_size=25
):
    """
    Query a DynamoDB table with pagination using a generator to handle large
 result sets.
    This approach is memory-efficient as it yields one page at a time.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
```

```
        page_size (int, optional): The number of items to return per page.
Defaults to 25.

    Yields:
        tuple: A tuple containing (items, page_number, last_page) where:
            - items is a list of items for the current page
            - page_number is the current page number (starting from 1)
            - last_page is a boolean indicating if this is the last page
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_number = 0

    # Paginate through the results
    while True:
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Get the current page of results
        items = response.get("Items", [])
        page_number += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # Determine if this is the last page
        is_last_page = last_evaluated_key is None
```

```
        # Yield the current page of results
        yield (items, page_number, is_last_page)

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if is_last_page:
            break
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK로 강력하게 일관된 읽기를 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제에서는 강력하게 일관된 읽기를 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- DynamoDB 쿼리의 일관성 수준을 구성합니다.
- 강력하게 일관된 읽기를 사용하여 최신 데이터를 가져옵니다.
- 최종 일관성과 강력한 일관성의 장단점을 이해합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x로 구성 가능한 읽기 일관성을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```
   public QueryResponse queryWithConsistentReads(
       final String tableName,
       final String partitionKeyName,
       final String partitionKeyValue,
       final boolean useConsistentRead) {

       CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);

       // Create expression attribute names for the column names
       final Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);

       // Create expression attribute values for the column values
       final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
       expressionAttributeValues.put(
           EXPRESSION_ATTRIBUTE_VALUE_PK,
           AttributeValue.builder().s(partitionKeyValue).build());

       // Create the query request
       final QueryRequest queryRequest = QueryRequest.builder()
           .tableName(tableName)
           .keyConditionExpression(KEY_CONDITION_EXPRESSION)
           .expressionAttributeNames(expressionAttributeNames)
           .expressionAttributeValues(expressionAttributeValues)
           .consistentRead(useConsistentRead)
           .build();

       try {
           final QueryResponse response = dynamoDbClient.query(queryRequest);
           LOGGER.log(Level.INFO, "Query successful. Found {0} items",
response.count());
           return response;
       } catch (ResourceNotFoundException e) {
           LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
           throw e;
       } catch (DynamoDbException e) {
           LOGGER.log(Level.SEVERE, "Error querying with consistent reads", e);
           throw e;
       }
   }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript로 구성 가능한 읽기 일관성을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
```

```
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)으로 강력하게 일관된 읽기 옵션을 사용하여 DynamoDB 테이블을 쿼리합니다.

```
import time

import boto3
from boto3.dynamodb.conditions import Key


def query_with_consistent_read(
    table_name,
    partition_key_name,
    partition_key_value,
    sort_key_name=None,
    sort_key_value=None,
    consistent_read=True,
):
    """
    Query a DynamoDB table with the option for strongly consistent reads.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
```

```
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str, optional): The name of the sort key attribute.
        sort_key_value (str, optional): The value of the sort key to query.
        consistent_read (bool, optional): Whether to use strongly consistent
reads. Defaults to True.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    if sort_key_name and sort_key_value:
        key_condition = key_condition & Key(sort_key_name).eq(sort_key_value)

    # Perform the query with the consistent read option
    response = table.query(KeyConditionExpression=key_condition,
ConsistentRead=consistent_read)

    return response
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK와 함께 PartiQL SELECT 문을 사용하여 DynamoDB 데이터 쿼리

다음 코드 예제에서는 PartiQL SELECT 문을 사용하여 데이터를 쿼리하는 방법을 보여줍니다.

JavaScript

JavaScript용 SDK(v3)

AWS SDK for JavaScript와 함께 PartiQL SELECT 문을 사용하여 DynamoDB 테이블에서 항목을 쿼리합니다.

```
/**
 * This example demonstrates how to query items from a DynamoDB table using
 PartiQL.
 * It shows different ways to select data with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Select all items from a DynamoDB table using PartiQL.
 * Note: This should be used with caution on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @returns The response from the ExecuteStatementCommand
 */
export const selectAllItems = async (tableName: string) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}"`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};
```

```
/**
 * Select an item by its primary key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByCompositeKey = async (
  tableName: string,
```

```
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select items using a filter condition with PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsWithFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };
```

```
  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select items using a begins_with function for prefix matching.
 * This is useful for querying hierarchical data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for prefix
 * @param prefix - The prefix to match
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByPrefix = async (
  tableName: string,
  attributeName: string,
  prefix: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE
 begins_with(${attributeName}, ?)`,
    Parameters: [prefix],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};
```

```
/**
 * Select items using a between condition for range queries.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for range
 * @param startValue - The start value of the range
 * @param endValue - The end value of the range
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByRange = async (
  tableName: string,
  attributeName: string,
  startValue: number | string,
  endValue: number | string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${attributeName} BETWEEN ?
 AND ?`,
    Parameters: [startValue, endValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to select items with different index types
 */
export const selectExamples = async () => {
  // Select all items from a table (use with caution on large tables)
  await selectAllItems("UsersTable");

  // Select by partition key (simple primary key)
  await selectItemByPartitionKey("UsersTable", "userId", "user123");
```

```
  // Select by composite key (partition key + sort key)
  await selectItemByCompositeKey("OrdersTable", "orderId", "order456",
 "productId", "prod789");

  // Select with a filter condition (can use any attribute)
  await selectItemsWithFilter("UsersTable", "userType", "premium");

  // Select items with a prefix (useful for hierarchical data)
  await selectItemsByPrefix("ProductsTable", "category", "electronics");

  // Select items within a range (useful for numeric or date ranges)
  await selectItemsByRange("OrdersTable", "orderDate", "2023-01-01",
 "2023-12-31");
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - BatchExecuteStatement
  - ExecuteStatement

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 TTL 항목에 대한 DynamoDB 테이블 쿼리

다음 코드 예제는 TTL 항목을 쿼리하는 방법을 보여줍니다.

Java

SDK for Java 2.x

필터링된 표현식을 쿼리하여 AWS SDK for Java 2.x를 사용해 DynamoDB 테이블에서 TTL 항목을 수집합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
```

```
import java.util.Map;
import java.util.Optional;

        final QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final QueryResponse response = ddb.query(request);
            System.out.println("Query successful. Found " + response.count() + "
 items that have not expired yet.");

            // Print each item
            response.items().forEach(item -> {
                System.out.println("Item: " + item);
            });

            return 0;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

JavaScript

SDK for JavaScript (v3)

필터링된 표현식을 쿼리하여 AWS SDK for JavaScript를 사용해 DynamoDB 테이블에서 TTL 항목을 수집합니다.

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1')
 => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

필터링된 표현식을 쿼리하여 AWS SDK for Python (Boto3)를 사용해 DynamoDB 테이블에서 TTL 항목을 수집합니다.

```
from datetime import datetime

import boto3


def query_dynamodb_items(table_name, partition_key):
    """

    :param table_name: Name of the DynamoDB table
    :param partition_key:
    :return:
    """
    try:
        # Initialize a DynamoDB resource
        dynamodb = boto3.resource("dynamodb", region_name="us-east-1")

        # Specify your table
        table = dynamodb.Table(table_name)

        # Get the current time in epoch format
        current_time = int(datetime.now().timestamp())

        # Perform the query operation with a filter expression to exclude expired
 items
        # response = table.query(
        #
 KeyConditionExpression=boto3.dynamodb.conditions.Key('partitionKey').eq(partition_key),
        #
 FilterExpression=boto3.dynamodb.conditions.Attr('expireAt').gt(current_time)
        # )
        response = table.query(

 KeyConditionExpression=dynamodb.conditions.Key("partitionKey").eq(partition_key),

 FilterExpression=dynamodb.conditions.Attr("expireAt").gt(current_time),
        )
```

```
        # Print the items that are not expired
        for item in response["Items"]:
            print(item)

    except Exception as e:
        print(f"Error querying items: {e}")


# Call the function with your values
query_dynamodb_items("Music", "your-partition-key-value")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK로 날짜 및 시간 패턴을 사용하여 DynamoDB 테이블 쿼리

다음 코드 예제에서는 날짜 및 시간 패턴을 사용하여 테이블을 쿼리하는 방법을 보여줍니다.

- DynamoDB에 날짜/시간 값을 저장하고 쿼리합니다.
- 정렬 키를 사용하여 날짜 범위 쿼리를 구현합니다.
- 효과적인 쿼리를 위해 날짜 문자열의 형식을 지정합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x로 정렬 키에 날짜 범위를 사용하여 쿼리합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.LocalDate;
import java.util.HashMap;
```

```
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithDateRange(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDate startDate,
        final LocalDate endDate) {

        // Focus on query logic, assuming parameters are valid
        if (startDate == null || endDate == null) {
            throw new IllegalArgumentException("Start date and end date cannot be
 null");
        }

        if (endDate.isBefore(startDate)) {
            throw new IllegalArgumentException("End date must be after start
 date");
        }

        // Format dates as ISO strings for DynamoDB (using just the date part)
        final String formattedStartDate = startDate.toString();
        final String formattedEndDate = endDate.toString();

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_START_DATE,
            AttributeValue.builder().s(formattedStartDate).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_END_DATE,
```

```
            AttributeValue.builder().s(formattedEndDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query by date range successful. Found {0}
 items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying by date range: {0}",
 e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x로 날짜-시간 변수를 사용하여 쿼리합니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithDateTime(
        final String tableName,
```

```
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final String startDate,
        final String endDate) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateDateRangeParameters(dateKeyName, startDate,
endDate);
        CodeSampleUtils.validateDateFormat("Start date", startDate);
        CodeSampleUtils.validateDateFormat("End date", endDate);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put("#dateKey", dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            ":startDate", AttributeValue.builder().s(startDate).build());
        expressionAttributeValues.put(
            ":endDate", AttributeValue.builder().s(endDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query successful. Found " + response.count() + "
items");
            return response;
        } catch (ResourceNotFoundException e) {
```

```
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with date range: " +
 e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x로 Unix 에포크 타임스탬프의 날짜 범위 내에서 쿼리합니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithDateTimeEpoch(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final long startEpoch,
        final long endEpoch) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Date key name", dateKeyName);
        CodeSampleUtils.validateEpochTimestamp("Start epoch", startEpoch);
        CodeSampleUtils.validateEpochTimestamp("End epoch", endEpoch);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
```

```
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put("#dateKey", dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            ":startDate",
AttributeValue.builder().n(String.valueOf(startEpoch)).build());
        expressionAttributeValues.put(
            ":endDate",
AttributeValue.builder().n(String.valueOf(endEpoch)).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query successful. Found " + response.count() + "
items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with epoch timestamps: " +
e.getMessage());
            throw e;
        }
    }
```

AWS SDK for Java 2.x로 LocalDateTime 객체를 사용하여 날짜 범위 내에서 쿼리합니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithDateTimeLocalDateTime(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDateTime startDateTime,
        final LocalDateTime endDateTime) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Date key name", dateKeyName);
        if (startDateTime == null || endDateTime == null) {
            throw new IllegalArgumentException("Start and end LocalDateTime must
 not be null");
        }

        // Convert LocalDateTime to ISO-8601 strings in UTC with the correct
 format
        final String startDate =
startDateTime.atZone(ZoneOffset.UTC).format(DATE_TIME_FORMATTER);
        final String endDate =
endDateTime.atZone(ZoneOffset.UTC).format(DATE_TIME_FORMATTER);

        return queryWithDateTime(tableName, partitionKeyName, partitionKeyValue,
dateKeyName, startDate, endDate);
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 Query를 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript로 정렬 키에 날짜 범위를 사용하여 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort
 key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
```

```
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

AWS SDK for JavaScript로 날짜-시간 변수를 사용하여 쿼리합니다.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} dateKeyName - The name of the date attribute to filter on
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRange(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  dateKeyName,
```

```
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: `#pk = :pkValue AND #dateAttr BETWEEN :startDate
 AND :endDate`,
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#dateAttr": dateKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range: ${error}`);
    throw error;
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 Query를 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)로 정렬 키에 날짜 범위를 사용하여 쿼리합니다.

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_date_range(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        start_date (datetime): The start date for the query range.
        end_date (datetime): The end date for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date values as ISO 8601 strings
    # DynamoDB works well with ISO format for date values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key using BETWEEN operator
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
```

```
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response


def query_with_date_range_by_month(
    table_name, partition_key_name, partition_key_value, sort_key_name, year,
 month
):
    """
    Query a DynamoDB table for a specific month's data.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        year (int): The year to query.
        month (int): The month to query (1-12).

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Calculate the start and end dates for the specified month
    if month == 12:
        next_year = year + 1
        next_month = 1
    else:
        next_year = year
        next_month = month + 1
```

```
    start_date = datetime(year, month, 1)
    end_date = datetime(next_year, next_month, 1) - timedelta(microseconds=1)

    # Format the date values as ISO 8601 strings
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(KeyConditionExpression=key_condition)

    return response
```

AWS SDK for Python (Boto3)로 날짜-시간 변수를 사용하여 쿼리합니다.

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_datetime(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range filter on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date/
time values).
        start_date (datetime): The start date/time for the query range.
```

```
        end_date (datetime): The end date/time for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date/time values as ISO 8601 strings
    # DynamoDB works well with ISO format for date/time values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response



def example_usage():
    """Example of how to use the query_with_datetime function."""
    # Example parameters
    table_name = "Events"
    partition_key_name = "EventType"
    partition_key_value = "UserLogin"
    sort_key_name = "Timestamp"

    # Create date/time variables for the query
    end_date = datetime.now()
```

```
    start_date = end_date - timedelta(days=7)  # Query events from the last 7
days

    print(f"Querying events from {start_date.isoformat()} to
{end_date.isoformat()}")

    # Execute the query
    response = query_with_datetime(
        table_name, partition_key_name, partition_key_value, sort_key_name,
start_date, end_date
    )

    # Process the results
    items = response.get("Items", [])
    print(f"Found {len(items)} items")

    for item in items:
        print(f"Event: {item}")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 Query를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 EXIF 및 기타 이미지 정보 저장

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- JPG, JPEG 또는 PNG 파일에서 EXIF 정보를 가져옵니다.
- Amazon S3 버킷에 이미지 파일을 업로드합니다.
- Amazon Rekognition을 사용하여 파일에서 3가지 주요 속성(레이블)을 파악합니다.
- EXIF 및 레이블 정보를 리전의 Amazon DynamoDB 테이블에 추가합니다.

Rust

SDK for Rust

JPG, JPEG 또는 PNG 파일에서 EXIF 정보를 가져오고, 이미지 파일을 Amazon S3 버킷에 업로드하며, Amazon Rekognition을 사용하여 파일에서 3가지 주요 속성(Amazon Rekognition의 레이블)을 파악한 후 EXIF 및 레이블 정보를 리전의 Amazon DynamoDB 테이블에 추가합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Amazon Rekognition
- Amazon S3

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS Command Line Interface v2를 사용하여 DynamoDB에 대한 속성 기반 액세스 제어 설정

다음 코드 예제에서는 DynamoDB에 대한 속성 기반 액세스 제어(ABAC)를 구현하는 방법을 보여줍니다.

- ABAC를 위한 IAM 정책을 생성합니다.
- 부서별 태그가 있는 테이블을 생성합니다.
- 태그를 기반으로 테이블을 나열하고 필터링합니다.

Bash

Bash 스크립트와 함께 AWS CLI사용

ABAC를 위한 IAM 정책을 생성합니다.

```
# Step 1: Create a policy document for ABAC
cat > abac-policy.json << 'EOF'
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}",
          "aws:ResourceTag/Environment": "Development"
        }
      }
    }
  ]
}
EOF

# Step 2: Create the IAM policy
aws iam create-policy \
    --policy-name DynamoDBDepartmentBasedAccess \
    --policy-document file://abac-policy.json
```

부서별 태그가 있는 테이블을 생성합니다.

```
# Create a DynamoDB table with tags for ABAC
aws dynamodb create-table \
    --table-name FinanceData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Department,Value=Finance \
        Key=Environment,Value=Development

# Create another table with different tags
aws dynamodb create-table \
    --table-name MarketingData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Department,Value=Marketing \
        Key=Environment,Value=Production
```

태그를 기반으로 테이블을 나열하고 필터링합니다.

```
# List all DynamoDB tables
echo "Listing all tables:"
aws dynamodb list-tables

# Get ARNs for all tables
echo -e "\nGetting ARNs for all tables:"
TABLE_ARNS=$(aws dynamodb list-tables --query "TableNames[*]" --output text |
 xargs -I {} aws dynamodb describe-table --table-name {} --query "Table.TableArn"
 --output text)

# For each table ARN, list its tags
echo -e "\nListing tags for each table:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    echo -e "\nTags for table: $TABLE_NAME"
    aws dynamodb list-tags-of-resource --resource-arn $ARN
```

```
done

# Example: Find tables with a specific tag
echo -e "\nFinding tables with Environment=Production tag:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    TAGS=$(aws dynamodb list-tags-of-resource --resource-arn $ARN --query "Tags[?
Key=='Environment' && Value=='Production']" --output text)
    if [ ! -z "$TAGS" ]; then
        echo "Table with Production tag: $TABLE_NAME"
    fi
done
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - CreatePolicy
  - CreateTable
  - ListTables

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 DynamoDB에서 업데이트 표현식 순서 이해

다음 코드 예제에서는 업데이트 표현식 순서를 이해하는 방법을 보여줍니다.

- DynamoDB가 업데이트 표현식을 처리하는 방법을 알아봅니다.
- 업데이트 표현식의 연산 순서를 이해합니다.
- 표현식 평가를 이해하여 예상치 못한 결과를 방지합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용한 업데이트 표현식 순서를 보여줍니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Demonstrates the effect of update expression order.
     *
     * <p>This method shows how the order of operations in an update expression
     * affects the result of the update.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return Map containing the results of different update orders
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, Object> demonstrateUpdateOrder(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
 AttributeValue> key) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Initialize the item with a counter
            UpdateItemRequest initRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :zero, OldCounter = :zero")
                .expressionAttributeValues(
                    Map.of(":zero", AttributeValue.builder().n("0").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            dynamoDbClient.updateItem(initRequest);

            // Example 1: SET first, then ADD
            UpdateItemRequest setFirstRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
```

```
                .updateExpression("SET Counter = :value ADD
OldCounter :increment")
                .expressionAttributeValues(Map.of(
                    ":value", AttributeValue.builder().n("10").build(),
                    ":increment", AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse setFirstResponse =
dynamoDbClient.updateItem(setFirstRequest);
            results.put("setFirstResponse", setFirstResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 2: ADD first, then SET
            UpdateItemRequest addFirstRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("ADD Counter :increment SET OldCounter
= :value")
                .expressionAttributeValues(Map.of(
                    ":value", AttributeValue.builder().n("10").build(),
                    ":increment", AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse addFirstResponse =
dynamoDbClient.updateItem(addFirstRequest);
            results.put("addFirstResponse", addFirstResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 3: SET with multiple attributes
            UpdateItemRequest multiSetRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :value, OldCounter = Counter")
                .expressionAttributeValues(
                    Map.of(":value", AttributeValue.builder().n("10").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();
```

```
            UpdateItemResponse multiSetResponse =
dynamoDbClient.updateItem(multiSetRequest);
            results.put("multiSetResponse", multiSetResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 4: SET with expression using the same attribute
            UpdateItemRequest selfReferenceRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = Counter + :increment, OldCounter
= Counter")
                .expressionAttributeValues(
                    Map.of(":increment",
AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse selfReferenceResponse =
dynamoDbClient.updateItem(selfReferenceRequest);
            results.put("selfReferenceResponse", selfReferenceResponse);

            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Updates an item with SET first, then REMOVE.
     *
     * <p>This method demonstrates updating an item with SET operation first,
     * followed by a REMOVE operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributeToSet The attribute to set
     * @param setValue The value to set
```

```
     * @param attributeToRemove The attribute to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithSetFirst(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String attributeToSet,
        AttributeValue setValue,
        String attributeToRemove) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #setAttr = :setValue REMOVE #removeAttr")
            .expressionAttributeNames(Map.of(
                "#setAttr", attributeToSet,
                "#removeAttr", attributeToRemove))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with SET first: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Updates an item with REMOVE first, then SET.
     *
     * <p>This method demonstrates updating an item with REMOVE operation first,
     * followed by a SET operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
```

```
     * @param key The key of the item to update
     * @param attributeToSet The attribute to set
     * @param setValue The value to set
     * @param attributeToRemove The attribute to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithRemoveFirst(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String attributeToSet,
        AttributeValue setValue,
        String attributeToRemove) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #removeAttr SET #setAttr = :setValue")
            .expressionAttributeNames(Map.of(
                "#setAttr", attributeToSet,
                "#removeAttr", attributeToRemove))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with REMOVE first: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Updates an item with all operation types in a specific order.
     *
     * <p>This method demonstrates using all operation types (SET, REMOVE, ADD,
DELETE)
```

```
     * in a specific order in a single update expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithAllOperationTypes(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #stringAttr = :stringVal, #mapAttr.#nestedAttr
= :nestedVal " + "REMOVE #oldAttr "
                + "ADD #counterAttr :increment "
                + "DELETE #stringSetAttr :stringSetVal")
            .expressionAttributeNames(Map.of(
                "#stringAttr", "StringAttribute",
                "#mapAttr", "MapAttribute",
                "#nestedAttr", "NestedAttribute",
                "#oldAttr", "OldAttribute",
                "#counterAttr", "CounterAttribute",
                "#stringSetAttr", "StringSetAttribute"))
            .expressionAttributeValues(Map.of(
                ":stringVal", AttributeValue.builder().s("New Value").build(),
                ":nestedVal", AttributeValue.builder().s("Nested Value").build(),
                ":increment", AttributeValue.builder().n("1").build(),
                ":stringSetVal", AttributeValue.builder().ss("Value1").build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with all operation types: " +
e.getMessage())
                .cause(e)
                .build();
```

```
        }
    }

    /**
     * Gets the current state of an item.
     *
     * <p>Helper method to retrieve the current state of an item.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @return The item or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, AttributeValue> getItem(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        // Define the get parameters
        GetItemRequest request =
            GetItemRequest.builder().tableName(tableName).key(key).build();

        // Perform the get operation
        try {
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the item if it exists, otherwise null
            return response.item();
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get item: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

AWS SDK for Java 2.x를 사용한 업데이트 표현식 순서의 사용 예입니다.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
```

```
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating update expression order in DynamoDB");

        try {
            // Example 1: Demonstrate update order effects
            System.out.println("\nExample 1: Demonstrating update order
effects");
            Map<String, Object> orderResults =
demonstrateUpdateOrder(dynamoDbClient, tableName, key);

            if ((boolean) orderResults.get("success")) {
                System.out.println("SET first, then ADD:");
                System.out.println("  " + orderResults.get("setFirstResponse"));

                System.out.println("ADD first, then SET:");
                System.out.println("  " + orderResults.get("addFirstResponse"));

                System.out.println("SET with multiple attributes:");
                System.out.println("  " + orderResults.get("multiSetResponse"));

                System.out.println("SET with self-reference:");
                System.out.println("  " +
orderResults.get("selfReferenceResponse"));
            } else {
                System.out.println("Error: " + orderResults.get("error"));
            }

            // Example 2: Update with SET first, then REMOVE
            System.out.println("\nExample 2: Update with SET first, then
REMOVE");
            UpdateItemResponse setFirstResponse = updateWithSetFirst(
                dynamoDbClient,
                tableName,
                key,
                "Status",
                AttributeValue.builder().s("Active").build(),
                "OldStatus");

            System.out.println("Updated attributes: " +
setFirstResponse.attributes());

            // Example 3: Update with REMOVE first, then SET
```

```
            System.out.println("\nExample 3: Update with REMOVE first, then
SET");
            UpdateItemResponse removeFirstResponse = updateWithRemoveFirst(
                dynamoDbClient,
                tableName,
                key,
                "Status",
                AttributeValue.builder().s("Inactive").build(),
                "OldStatus");

            System.out.println("Updated attributes: " +
removeFirstResponse.attributes());

            // Example 4: Update with all operation types
            System.out.println("\nExample 4: Update with all operation types");
            UpdateItemResponse allOpsResponse =
updateWithAllOperationTypes(dynamoDbClient, tableName, key);

            System.out.println("Updated attributes: " +
allOpsResponse.attributes());

            // Example 5: Get the current state of the item
            System.out.println("\nExample 5: Current state of the item");
            Map<String, AttributeValue> item = getItem(dynamoDbClient, tableName,
key);

            if (item != null) {
                System.out.println("Item: " + item);
            } else {
                System.out.println("Item not found");
            }

            // Explain update expression order
            System.out.println("\nKey points about update expression order in
DynamoDB:");
            System.out.println("1. Update expressions are processed in this
order: SET, REMOVE, ADD, DELETE");
            System.out.println("2. Within each clause, operations are processed
from left to right");
            System.out.println("3. SET operations use the item state before any
updates in the expression");
            System.out.println("4. When an attribute is referenced multiple
times, the first operation wins");
```

```
            System.out.println("5. To reference a new value, split the update
 into multiple operations");
            System.out.println("6. The order of clauses in the expression doesn't
 change the evaluation order");
            System.out.println("7. For complex updates, consider using multiple
 separate update operations");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용한 업데이트 표현식 순서를 보여줍니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an item with multiple actions in a single update expression.
 *
 * This function demonstrates how to use multiple actions in a single update
 expression
 * and how DynamoDB processes these actions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} updateExpression - The update expression with multiple actions
 * @param {Object} [expressionAttributeNames] - Expression attribute name
 placeholders
```

```
 * @param {Object} [expressionAttributeValues] - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}

/**
 * Demonstrate that variables hold copies of existing values before
 modifications.
 *
```

```
 * This function creates an item with initial values, then updates it with an
 expression
 * that uses the values of attributes before they are modified in the same
 expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateValueCopying(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = { ...key, a: 1, b: 2, c: 3 };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with an expression that uses values before they are
 modified
  // This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to
 the value of 'b'
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE a SET b = a, c = b",
```

```
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}

/**
 * Demonstrate the order in which different action types are processed.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the
 order
 * in which they are processed.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateActionOrder(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
```

```
  const initialItem = {
    ...key,
    counter: 10,
    set_attr: new Set(["A", "B", "C"]),
    to_remove: "This will be removed",
    to_modify: "Original value"
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple action types
  // The actions will be processed in this order: REMOVE, SET, ADD, DELETE
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE to_remove SET to_modify = :new_value ADD
counter :increment DELETE set_attr :elements",
    ExpressionAttributeValues: {
      ":new_value": "Updated value",
      ":increment": 5,
      ":elements": new Set(["B"])
    },
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};
```

```
  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}

/**
 * Update multiple attributes with a single SET action.
 *
 * This function demonstrates how to update multiple attributes in a single SET
 action,
 * which is more efficient than using multiple separate update operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {Object} attributes - The attributes to update and their new values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleSetActions(
  config,
  tableName,
  key,
  attributes
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET ";
  const expressionAttributeValues = {};

  // Add each attribute to the update expression
  Object.entries(attributes).forEach(([attrName, attrValue], index) => {
    const valuePlaceholder = `:val${index}`;

    if (index > 0) {
      updateExpression += ", ";
    }
    updateExpression += `${attrName} = ${valuePlaceholder}`;
```

```
    expressionAttributeValues[valuePlaceholder] = attrValue;
  });

  // Execute the update
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Update an attribute with a value from another attribute or a default value.
 *
 * This function demonstrates how to use if_not_exists to conditionally copy a
 value
 * from one attribute to another, or use a default value if the source doesn't
 exist.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} sourceAttribute - The attribute to copy the value from
 * @param {string} targetAttribute - The attribute to update
 * @param {any} defaultValue - The default value to use if the source attribute
 doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithConditionalValueCopying(
  config,
  tableName,
  key,
  sourceAttribute,
  targetAttribute,
  defaultValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Use if_not_exists to conditionally copy the value
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${targetAttribute} =
 if_not_exists(${sourceAttribute}, :default)`,
    ExpressionAttributeValues: {
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Demonstrate complex update expressions with multiple operations on the same
 attribute.
 *
 * This function shows how DynamoDB processes multiple operations on the same
 attribute
 * in a single update expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateMultipleOperationsOnSameAttribute(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    list_attr: [1, 2, 3],
    map_attr: {
```

```
    nested1: "value1",
    nested2: "value2"
  }
};

await docClient.send(new PutCommand({
  TableName: tableName,
  Item: initialItem
}));

// Step 2: Get the item to verify initial state
const responseBefore = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
}));

const itemBefore = responseBefore.Item || {};

// Step 3: Update the item with multiple operations on the same attributes
const updateResponse = await docClient.send(new UpdateCommand({
  TableName: tableName,
  Key: key,
  UpdateExpression: `
    SET counter = counter + :inc1,
        counter = counter + :inc2,
        map_attr.nested1 = :new_val1,
        map_attr.nested3 = :new_val3,
        list_attr[0] = list_attr[1],
        list_attr[1] = list_attr[2]
  `,
  ExpressionAttributeValues: {
    ":inc1": 5,
    ":inc2": 3,
    ":new_val1": "updated_value1",
    ":new_val3": "new_value3"
  },
  ReturnValues: "UPDATED_NEW"
}));

// Step 4: Get the item to verify final state
const responseAfter = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
}));
```

```
  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}
```

AWS SDK for JavaScript를 사용한 업데이트 표현식 순서의 사용 예입니다.

```
/**
 * Example of how to use update expression order of operations in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "OrderProcessing";

  console.log("Demonstrating update expression order of operations in DynamoDB");

  try {
    // Example 1: Demonstrating value copying in update expressions
    console.log("\nExample 1: Demonstrating value copying in update
 expressions");
    const results1 = await demonstrateValueCopying(
      config,
      tableName,
      { OrderId: "order123" }
    );

    console.log("Initial state:", JSON.stringify(results1.initialState, null,
 2));
    console.log("Update response:", JSON.stringify(results1.updateResponse, null,
 2));
    console.log("Final state:", JSON.stringify(results1.finalState, null, 2));

    console.log("\nExplanation:");
    console.log("1. The initial state had a=1, b=2, c=3");
```

```
   console.log("2. The update expression 'REMOVE a SET b = a, c = b' did the
following:");
   console.log("   - Copied the value of 'a' (which was 1) to be used for 'b'");
   console.log("   - Copied the value of 'b' (which was 2) to be used for 'c'");
   console.log("   - Removed the attribute 'a'");
   console.log("3. The final state has b=1, c=2, and 'a' is removed");
   console.log("4. This demonstrates that DynamoDB uses the values of attributes
as they were BEFORE any modifications");

   // Example 2: Demonstrating the order of different action types
   console.log("\nExample 2: Demonstrating the order of different action
types");
   const results2 = await demonstrateActionOrder(
     config,
     tableName,
     { OrderId: "order456" }
   );

   console.log("Initial state:", JSON.stringify(results2.initialState, null,
2));
   console.log("Update response:", JSON.stringify(results2.updateResponse, null,
2));
   console.log("Final state:", JSON.stringify(results2.finalState, null, 2));

   console.log("\nExplanation:");
   console.log("1. The update expression contained multiple action types:
REMOVE, SET, ADD, DELETE");
   console.log("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE");
   console.log("3. First, 'to_remove' was removed");
   console.log("4. Then, 'to_modify' was set to a new value");
   console.log("5. Next, 'counter' was incremented by 5");
   console.log("6. Finally, 'B' was removed from the set attribute");

   // Example 3: Updating multiple attributes in a single SET action
   console.log("\nExample 3: Updating multiple attributes in a single SET
action");
   const response3 = await updateWithMultipleSetActions(
     config,
     tableName,
     { OrderId: "order789" },
     {
       Status: "Shipped",
       ShippingDate: "2025-05-28",
```

```
        TrackingNumber: "1Z999AA10123456784"
      }
    );

    console.log("Multiple attributes updated successfully:",
JSON.stringify(response3.Attributes, null, 2));

    // Example 4: Conditional value copying with if_not_exists
    console.log("\nExample 4: Conditional value copying with if_not_exists");
    const response4 = await updateWithConditionalValueCopying(
      config,
      tableName,
      { OrderId: "order101" },
      "PreferredShippingMethod",
      "ShippingMethod",
      "Standard"
    );

    console.log("Conditional value copying result:",
JSON.stringify(response4.Attributes, null, 2));

    // Example 5: Multiple operations on the same attribute
    console.log("\nExample 5: Multiple operations on the same attribute");
    const results5 = await demonstrateMultipleOperationsOnSameAttribute(
      config,
      tableName,
      { OrderId: "order202" }
    );

    console.log("Initial state:", JSON.stringify(results5.initialState, null,
2));
    console.log("Update response:", JSON.stringify(results5.updateResponse, null,
2));
    console.log("Final state:", JSON.stringify(results5.finalState, null, 2));

    console.log("\nExplanation:");
    console.log("1. The counter was incremented twice (first by 5, then by 3) for
a total of +8");
    console.log("2. The map attribute had one value updated and a new nested
attribute added");
    console.log("3. The list attribute had values shifted (value at index 1 moved
to index 0, value at index 2 moved to index 1)");
    console.log("4. All operations within the SET action are processed from left
to right");
```

```
    // Key points about update expression order of operations
    console.log("\nKey Points About Update Expression Order of Operations:");
    console.log("1. Variables in expressions hold copies of attribute values as
 they existed BEFORE any modifications");
    console.log("2. Multiple actions in an update expression are processed in
 this order: REMOVE, SET, ADD, DELETE");
    console.log("3. Within each action type, operations are processed from left
 to right");
    console.log("4. You can reference the same attribute multiple times in an
 expression");
    console.log("5. You can use if_not_exists() to conditionally set values based
 on attribute existence");
    console.log("6. Using a single update expression with multiple actions is
 more efficient than multiple separate updates");
    console.log("7. The update expression is atomic - either all actions succeed
 or none do");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용한 업데이트 표현식 순서를 보여줍니다.

```
import boto3
import json
from typing import Any, Dict, Optional


def update_with_multiple_actions(
    table_name: str,
    key: Dict[str, Any],
    update_expression: str,
    expression_attribute_names: Optional[Dict[str, str]] = None,
    expression_attribute_values: Optional[Dict[str, Any]] = None,
```

```
) -> Dict[str, Any]:
    """
    Update an item with multiple actions in a single update expression.

    This function demonstrates how to use multiple actions in a single update
 expression
    and how DynamoDB processes these actions.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        update_expression (str): The update expression with multiple actions.
        expression_attribute_names (Optional[Dict[str, str]]): Expression
 attribute name placeholders.
        expression_attribute_values (Optional[Dict[str, Any]]): Expression
 attribute value placeholders.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Prepare the update parameters
    update_params = {
        "Key": key,
        "UpdateExpression": update_expression,
        "ReturnValues": "UPDATED_NEW",
    }

    # Add expression attribute names if provided
    if expression_attribute_names:
        update_params["ExpressionAttributeNames"] = expression_attribute_names

    # Add expression attribute values if provided
    if expression_attribute_values:
        update_params["ExpressionAttributeValues"] = expression_attribute_values

    # Execute the update
    response = table.update_item(**update_params)

    return response
```

```
def demonstrate_value_copying(table_name: str, key: Dict[str, Any]) -> Dict[str,
 Any]:
    """
    Demonstrate that variables hold copies of existing values before
 modifications.

    This function creates an item with initial values, then updates it with an
 expression
    that uses the values of attributes before they are modified in the same
 expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to create and update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Step 1: Create an item with initial values
    initial_item = key.copy()
    initial_item.update({"a": 1, "b": 2, "c": 3})

    table.put_item(Item=initial_item)

    # Step 2: Get the item to verify initial state
    response_before = table.get_item(Key=key)
    item_before = response_before.get("Item", {})

    # Step 3: Update the item with an expression that uses values before they are
 modified
    # This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to
 the value of 'b'
    update_response = table.update_item(
        Key=key, UpdateExpression="REMOVE a SET b = a, c = b",
 ReturnValues="UPDATED_NEW"
    )

    # Step 4: Get the item to verify final state
```

```
    response_after = table.get_item(Key=key)
    item_after = response_after.get("Item", {})

    # Return the results
    return {
        "initial_state": item_before,
        "update_response": update_response,
        "final_state": item_after,
    }


def demonstrate_action_order(table_name: str, key: Dict[str, Any]) -> Dict[str,
 Any]:
    """
    Demonstrate the order in which different action types are processed.

    This function creates an item with initial values, then updates it with an
 expression
    that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the
 order
    in which they are processed.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to create and update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Step 1: Create an item with initial values
    initial_item = key.copy()
    initial_item.update(
        {
            "counter": 10,
            "set_attr": set(["A", "B", "C"]),
            "to_remove": "This will be removed",
            "to_modify": "Original value",
        }
    )
```

```
    table.put_item(Item=initial_item)

    # Step 2: Get the item to verify initial state
    response_before = table.get_item(Key=key)
    item_before = response_before.get("Item", {})

    # Step 3: Update the item with multiple action types
    # The actions will be processed in this order: REMOVE, SET, ADD, DELETE
    update_response = table.update_item(
        Key=key,
        UpdateExpression="REMOVE to_remove SET to_modify = :new_value ADD
 counter :increment DELETE set_attr :elements",
        ExpressionAttributeValues={
            ":new_value": "Updated value",
            ":increment": 5,
            ":elements": set(["B"]),
        },
        ReturnValues="UPDATED_NEW",
    )

    # Step 4: Get the item to verify final state
    response_after = table.get_item(Key=key)
    item_after = response_after.get("Item", {})

    # Return the results
    return {
        "initial_state": item_before,
        "update_response": update_response,
        "final_state": item_after,
    }


def update_with_multiple_set_actions(
    table_name: str, key: Dict[str, Any], attributes: Dict[str, Any]
) -> Dict[str, Any]:
    """
    Update multiple attributes with a single SET action.

    This function demonstrates how to update multiple attributes in a single SET
 action,
    which is more efficient than using multiple separate update operations.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        key (Dict[str, Any]): The primary key of the item to update.
        attributes (Dict[str, Any]): The attributes to update and their new
 values.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the update expression and expression attribute values
    update_expression = "SET "
    expression_attribute_values = {}

    # Add each attribute to the update expression
    for i, (attr_name, attr_value) in enumerate(attributes.items()):
        value_placeholder = f":val{i}"

        if i > 0:
            update_expression += ", "
        update_expression += f"{attr_name} = {value_placeholder}"

        expression_attribute_values[value_placeholder] = attr_value

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeValues=expression_attribute_values,
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_with_conditional_value_copying(
    table_name: str,
    key: Dict[str, Any],
    source_attribute: str,
    target_attribute: str,
    default_value: Any,
) -> Dict[str, Any]:
```

```
    """
    Update an attribute with a value from another attribute or a default value.

    This function demonstrates how to use if_not_exists to conditionally copy a
value
    from one attribute to another, or use a default value if the source doesn't
exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        source_attribute (str): The attribute to copy the value from.
        target_attribute (str): The attribute to update.
        default_value (Any): The default value to use if the source attribute
doesn't exist.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use if_not_exists to conditionally copy the value
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {target_attribute} =
if_not_exists({source_attribute}, :default)",
        ExpressionAttributeValues={":default": default_value},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

AWS SDK for Python (Boto3)를 사용한 업데이트 표현식 순서의 사용 예입니다.

```
def example_usage():
    """Example of how to use update expression order of operations in
 DynamoDB."""
```

```
    # Example parameters
    table_name = "OrderProcessing"
    key = {"OrderId": "order123"}

    print("Example 1: Demonstrating value copying in update expressions")
    try:
        results = demonstrate_value_copying(table_name=table_name, key=key)

        print(f"Initial state: {json.dumps(results['initial_state'],
default=str)}")
        print(f"Update response: {json.dumps(results['update_response'],
default=str)}")
        print(f"Final state: {json.dumps(results['final_state'], default=str)}")

        print("\nExplanation:")
        print("1. The initial state had a=1, b=2, c=3")
        print("2. The update expression 'REMOVE a SET b = a, c = b' did the
following:")
        print("   - Copied the value of 'a' (which was 1) to be used for 'b'")
        print("   - Copied the value of 'b' (which was 2) to be used for 'c'")
        print("   - Removed the attribute 'a'")
        print("3. The final state has b=1, c=2, and 'a' is removed")
        print(
            "4. This demonstrates that DynamoDB uses the values of attributes as
they were BEFORE any modifications"
        )
    except Exception as e:
        print(f"Error demonstrating value copying: {e}")

    print("\nExample 2: Demonstrating the order of different action types")
    try:
        results = demonstrate_action_order(table_name=table_name, key={"OrderId":
"order456"})

        print(f"Initial state: {json.dumps(results['initial_state'],
default=str)}")
        print(f"Update response: {json.dumps(results['update_response'],
default=str)}")
        print(f"Final state: {json.dumps(results['final_state'], default=str)}")

        print("\nExplanation:")
        print("1. The update expression contained multiple action types: REMOVE,
SET, ADD, DELETE")
```

```
        print("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE")
        print("3. First, 'to_remove' was removed")
        print("4. Then, 'to_modify' was set to a new value")
        print("5. Next, 'counter' was incremented by 5")
        print("6. Finally, 'B' was removed from the set attribute")
    except Exception as e:
        print(f"Error demonstrating action order: {e}")

    print("\nExample 3: Updating multiple attributes in a single SET action")
    try:
        response = update_with_multiple_set_actions(
            table_name=table_name,
            key={"OrderId": "order789"},
            attributes={
                "Status": "Shipped",
                "ShippingDate": "2025-05-14",
                "TrackingNumber": "1Z999AA10123456784",
            },
        )

        print(
            f"Multiple attributes updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error updating multiple attributes: {e}")

    print("\nExample 4: Conditional value copying with if_not_exists")
    try:
        response = update_with_conditional_value_copying(
            table_name=table_name,
            key={"OrderId": "order101"},
            source_attribute="PreferredShippingMethod",
            target_attribute="ShippingMethod",
            default_value="Standard",
        )

        print(
            f"Conditional value copying result:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error with conditional value copying: {e}")
```

```
    print("\nKey Points About Update Expression Order of Operations:")
    print(
        "1. Variables in expressions hold copies of attribute values as they
existed BEFORE any modifications"
    )
    print(
        "2. Multiple actions in an update expression are processed in this order:
REMOVE, SET, ADD, DELETE"
    )
    print("3. Within each action type, operations are processed from left to
right")
    print("4. You can reference the same attribute multiple times in an
expression")
    print("5. You can use if_not_exists() to conditionally set values based on
attribute existence")
    print(
        "6. Using a single update expression with multiple actions is more
efficient than multiple separate updates"
    )
    print("7. The update expression is atomic - either all actions succeed or
none do")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 웜 처리량으로 DynamoDB 테이블 설정 업데이트

다음 코드 예제에서는 테이블의 웜 처리량 설정을 업데이트하는 방법을 보여줍니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용하여 기존 DynamoDB 테이블에서 웜 처리량 설정을 업데이트합니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndexUpdate;
import
 software.amazon.awssdk.services.dynamodb.model.UpdateGlobalSecondaryIndexAction;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

    public static WarmThroughput buildWarmThroughput(final Long
 readUnitsPerSecond, final Long writeUnitsPerSecond) {
        return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
    }

    /**
     * Updates a DynamoDB table with warm throughput settings for both the table
 and a global secondary index.
     *
     * @param ddb The DynamoDB client
     * @param tableName The name of the table to update
     * @param tableReadUnitsPerSecond Read units per second for the table
     * @param tableWriteUnitsPerSecond Write units per second for the table
     * @param globalSecondaryIndexName The name of the global secondary index to
 update
     * @param globalSecondaryIndexReadUnitsPerSecond Read units per second for
 the GSI
     * @param globalSecondaryIndexWriteUnitsPerSecond Write units per second for
 the GSI
     */
    public static void updateDynamoDBTable(
        final DynamoDbClient ddb,
        final String tableName,
        final Long tableReadUnitsPerSecond,
        final Long tableWriteUnitsPerSecond,
        final String globalSecondaryIndexName,
        final Long globalSecondaryIndexReadUnitsPerSecond,
        final Long globalSecondaryIndexWriteUnitsPerSecond) {

        final WarmThroughput tableWarmThroughput =
            buildWarmThroughput(tableReadUnitsPerSecond,
 tableWriteUnitsPerSecond);
```

```
        final WarmThroughput gsiWarmThroughput =
            buildWarmThroughput(globalSecondaryIndexReadUnitsPerSecond,
 globalSecondaryIndexWriteUnitsPerSecond);

        final GlobalSecondaryIndexUpdate globalSecondaryIndexUpdate =
 GlobalSecondaryIndexUpdate.builder()
            .update(UpdateGlobalSecondaryIndexAction.builder()
                .indexName(globalSecondaryIndexName)
                .warmThroughput(gsiWarmThroughput)
                .build())
            .build();

        final UpdateTableRequest request = UpdateTableRequest.builder()
            .tableName(tableName)
            .globalSecondaryIndexUpdates(globalSecondaryIndexUpdate)
            .warmThroughput(tableWarmThroughput)
            .build();

        try {
            ddb.updateTable(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }

        System.out.println(SUCCESS_MESSAGE);
    }
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateTable을 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용하여 기존 DynamoDB 테이블에서 웜 처리량 설정을 업데이트합니다.

```
import { DynamoDBClient, UpdateTableCommand } from "@aws-sdk/client-dynamodb";

export async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
```

```
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
            Update: {
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response:
 ${JSON.stringify(response)}`);
    return response;
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
updateDynamoDBTableWarmThroughput(
  'example-table',
```

```
  5, 5,
  'example-index',
  2, 2
);
*/
```

- API에 대한 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateTable을 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용하여 기존 DynamoDB 테이블에서 웜 처리량 설정을 업데이트합니다.

```
from boto3 import client
from botocore.exceptions import ClientError


def update_dynamodb_table_warm_throughput(
    table_name,
    table_read_units,
    table_write_units,
    gsi_name,
    gsi_read_units,
    gsi_write_units,
    region_name="us-east-1",
):
    """
    Updates the warm throughput of a DynamoDB table and a global secondary index.

    :param table_name: The name of the table to update.
    :param table_read_units: The new read units per second for the table's warm
 throughput.
    :param table_write_units: The new write units per second for the table's warm
 throughput.
    :param gsi_name: The name of the global secondary index to update.
    :param gsi_read_units: The new read units per second for the GSI's warm
 throughput.
    :param gsi_write_units: The new write units per second for the GSI's warm
 throughput.
```

```
    :param region_name: The AWS Region name to target. defaults to us-east-1
    :return: The response from the update_table operation
    """
    try:
        ddb = client("dynamodb", region_name=region_name)

        # Update the table's warm throughput
        table_warm_throughput = {
            "ReadUnitsPerSecond": table_read_units,
            "WriteUnitsPerSecond": table_write_units,
        }

        # Update the global secondary index's warm throughput
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_read_units,
            "WriteUnitsPerSecond": gsi_write_units,
        }

        # Construct the global secondary index update
        global_secondary_index_update = [
            {"Update": {"IndexName": gsi_name, "WarmThroughput":
gsi_warm_throughput}}
        ]

        # Construct the update table request
        update_table_request = {
            "TableName": table_name,
            "GlobalSecondaryIndexUpdates": global_secondary_index_update,
            "WarmThroughput": table_warm_throughput,
        }

        # Update the table
        response = ddb.update_table(**update_table_request)
        print("Table updated successfully!")
        return response  # Make sure to return the response
    except ClientError as e:
        print(f"Error updating table: {e}")
        raise e
```

- API에 대한 세부 정보는 Python(Boto3)용 AWS SDK API 참조의 UpdateTable을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 TTL이 포함된 DynamoDB 항목 업데이트

다음 코드 예제에서는 항목의 TTL을 업데이트하는 방법을 보여줍니다.

Java

SDK for Java 2.x

테이블의 기존 DynamoDB 항목에서 TTL을 업데이트합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

    public UpdateItemResponse updateItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
 sortKeyValue) {
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = new HashMap<>();
        keyMap.put(PRIMARY_KEY_ATTR,
 AttributeValue.builder().s(primaryKeyValue).build());
        keyMap.put(SORT_KEY_ATTR,
 AttributeValue.builder().s(sortKeyValue).build());

        // Create the expression attribute values
```

```
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            ":c",
 AttributeValue.builder().n(String.valueOf(currentTime)).build());
        expressionAttributeValues.put(
            ":e",
 AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final UpdateItemResponse response =
 dynamoDbClient.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

## JavaScript

### SDK for JavaScript (v3)

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    const client = new DynamoDBClient({
```

```
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-
value');
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

### Python

#### SDK for Python(Boto3)

```
from datetime import datetime, timedelta

import boto3


def update_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Update an existing DynamoDB item with a TTL.
    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        # Create the DynamoDB resource.
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expireAt time (90 days from now) in epoch second format
        expire_at = int((datetime.now() + timedelta(days=90)).timestamp())

        table.update_item(
            Key={"partitionKey": primary_key, "sortKey": sort_key},
            UpdateExpression="set updatedAt=:c, expireAt=:e",
            ExpressionAttributeValues={":c": current_time, ":e": expire_at},
        )

        print("Item updated successfully.")
    except Exception as e:
        print(f"Error updating item: {e}")


# Replace with your own values
update_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
```

```
)
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK에서 PartiQL UPDATE 문을 사용하여 DynamoDB 데이터 업데이트

다음 코드 예제에서는 PartiQL UPDATE 문을 사용하여 데이터를 업데이트하는 방법을 보여줍니다.

JavaScript

JavaScript용 SDK(v3)

AWS SDK for JavaScript와 함께 PartiQL UPDATE 문을 사용하여 DynamoDB 테이블의 항목을 업데이트합니다.

```
/**
 * This example demonstrates how to update items in a DynamoDB table using
 PartiQL.
 * It shows different ways to update documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Update a single attribute of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
```

```
 */
export const updateSingleAttribute = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update multiple attributes of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeUpdates - Object containing attribute names and their new
 values
 * @returns The response from the ExecuteStatementCommand
 */
export const updateMultipleAttributes = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeUpdates: Record<string, any>
) => {
  const client = new DynamoDBClient({});
```

```
  const docClient = DynamoDBDocumentClient.from(client);

  // Create SET clause for each attribute
  const setClause = Object.keys(attributeUpdates)
    .map((attr, index) => `${attr} = ?`)
    .join(", ");

  // Create parameters array with attribute values followed by the partition key
 value
  const parameters = [...Object.values(attributeUpdates), partitionKeyValue];

  const params = {
    Statement: `UPDATE "${tableName}" SET ${setClause} WHERE ${partitionKeyName}
 = ?`,
    Parameters: parameters,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item identified by a composite key (partition key + sort key) using
 PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
```

```
  sortKeyName: string,
  sortKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${sortKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item with a condition to ensure the update only happens if a
 condition is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any,
  conditionAttribute: string,
```

```
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${conditionAttribute} = ?`,
    Parameters: [attributeValue, partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated with condition successfully");
    return data;
  } catch (err) {
    console.error("Error updating item with condition:", err);
    throw err;
  }
};

/**
 * Batch update multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param updates - Array of objects containing key and update information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchUpdateItems = async (
  tableName: string,
  updates: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    attributeName: string;
    attributeValue: any;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each update
  const statements = updates.map((update) => {
    return {
```

```
      Statement: `UPDATE "${tableName}" SET ${update.attributeName} = ? WHERE
 ${update.partitionKeyName} = ?`,
      Parameters: [update.attributeValue, update.partitionKeyValue],
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch updated successfully");
    return data;
  } catch (err) {
    console.error("Error batch updating items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to update items with different index types
 */
export const updateExamples = async () => {
  // Update a single attribute using a simple primary key
  await updateSingleAttribute("UsersTable", "userId", "user123", "email",
 "newemail@example.com");

  // Update multiple attributes at once
  await updateMultipleAttributes("UsersTable", "userId", "user123", {
    email: "newemail@example.com",
    name: "John Smith",
    lastLogin: new Date().toISOString(),
  });

  // Update an item with a composite key (partition key + sort key)
  await updateItemWithCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789",
    "quantity",
    5
```

```
  );

  // Update with a condition
  await updateItemWithCondition(
    "UsersTable",
    "userId",
    "user123",
    "userStatus",
    "active",
    "userType",
    "premium"
  );

  // Batch update multiple items
  await batchUpdateItems("UsersTable", [
    {
      partitionKeyName: "userId",
      partitionKeyValue: "user123",
      attributeName: "lastLogin",
      attributeValue: new Date().toISOString(),
    },
    {
      partitionKeyName: "userId",
      partitionKeyValue: "user456",
      attributeName: "lastLogin",
      attributeValue: new Date().toISOString(),
    },
  ]);
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - BatchExecuteStatement
  - ExecuteStatement

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

# API Gateway를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon API Gateway에서 호출하여 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

Java

SDK for Java 2.x

Lambda Java 런타임 API를 사용하여 AWS Lambda 함수를 생성하는 방법을 보여줍니다. 이 예제에서는 특정 사용 사례를 수행하는 서로 다른 AWS 서비스를 호출합니다. 이 예제에서는 Amazon API Gateway에서 간접 호출한 Lambda 함수를 생성하여 작업 기념일에 대한 Amazon DynamoDB 테이블을 스캔하고 Amazon Simple Notification Service(Amazon SNS)를 사용하여 직원에게 1주년 기념일을 축하하는 문자 메시지를 전송하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Lambda JavaScript 런타임 API를 사용하여 AWS Lambda 함수를 생성하는 방법을 보여줍니다. 이 예제에서는 특정 사용 사례를 수행하는 서로 다른 AWS 서비스를 호출합니다. 이 예제에서는 Amazon API Gateway에서 간접 호출한 Lambda 함수를 생성하여 작업 기념일에 대한 Amazon DynamoDB 테이블을 스캔하고 Amazon Simple Notification Service(Amazon SNS)를 사용하여 직원에게 1주년 기념일을 축하하는 문자 메시지를 전송하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예시는 AWS SDK for JavaScript v3 개발자 안내서에서도 확인할 수 있습니다.

이 예제에서 사용되는 서비스

- API Gateway

- DynamoDB
- Lambda
- Amazon SNS

Python

SDK for Python (Boto3)

이 예제에서는 AWS Lambda 함수를 대상으로 하는 Amazon API Gateway REST API를 생성하고 사용하는 방법을 보여줍니다. Lambda 핸들러는 HTTP 메서드를 기반으로 라우팅하는 방법, 쿼리 문자열, 헤더 및 본문에서 데이터를 가져오는 방법, JSON 응답을 반환하는 방법을 보여줍니다.

- Lambda 함수를 배포합니다.
- API Gateway REST API를 생성합니다.
- Lambda 함수를 대상으로 하는 REST 리소스를 생성합니다.
- API Gateway가 Lambda 함수를 간접 호출할 수 있는 권한을 부여합니다.
- 요청 패키지를 사용하여 REST API에 요청을 보냅니다.
- 데모 중에 생성된 모든 리소스를 정리합니다.

이 예제는 GitHub에서 가장 잘 볼 수 있습니다. 전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## Step Functions를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 AWS Lambda 함수 시퀀스를 호출하는 AWS Step Functions 상태 머신을 만드는 방법을 보여줍니다.

Java

SDK for Java 2.x

AWS Step Functions 및 AWS SDK for Java 2.x을 사용하여 AWS 서버리스 워크플로를 만드는 방법을 보여줍니다. 각 워크플로 단계는 AWS Lambda 함수를 사용하여 구현됩니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 DynamoDB용 문서 모델 사용

다음 코드 예제는 DynamoDB 및 AWS SDK용 문서 모델을 사용하여 생성, 읽기, 업데이트 및 삭제(CRUD) 및 배치 작업을 수행하는 방법을 보여 줍니다.

자세한 내용은 문서 모델을 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

문서 모델을 사용하여 CRUD 작업을 수행합니다.

```
/// <summary>
/// Performs CRUD operations on an Amazon DynamoDB table.
```

```
    /// </summary>
    public class MidlevelItemCRUD
    {
        public static async Task Main()
        {
            var tableName = "ProductCatalog";
            var sampleBookId = 555;

            var client = new AmazonDynamoDBClient();
            var productCatalog = LoadTable(client, tableName);

            await CreateBookItem(productCatalog, sampleBookId);
            RetrieveBook(productCatalog, sampleBookId);

            // Couple of sample updates.
            UpdateMultipleAttributes(productCatalog, sampleBookId);
            UpdateBookPriceConditionally(productCatalog, sampleBookId);

            // Delete.
            await DeleteBook(productCatalog, sampleBookId);
        }

        /// <summary>
        /// Loads the contents of a DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB client object.</param>
        /// <param name="tableName">The name of the table to load.</param>
        /// <returns>A DynamoDB table object.</returns>
        public static Table LoadTable(IAmazonDynamoDB client, string tableName)
        {
            Table productCatalog = Table.LoadTable(client, tableName);
            return productCatalog;
        }

        /// <summary>
        /// Creates an example book item and adds it to the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task CreateBookItem(Table productCatalog, int
sampleBookId)
        {
```

```
            Console.WriteLine("\n*** Executing CreateBookItem() ***");
            var book = new Document
            {
                ["Id"] = sampleBookId,
                ["Title"] = "Book " + sampleBookId,
                ["Price"] = 19.99,
                ["ISBN"] = "111-1111111111",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["PageCount"] = 500,
                ["Dimensions"] = "8.5x11x.5",
                ["InPublication"] = new DynamoDBBool(true),
                ["InStock"] = new DynamoDBBool(false),
                ["QuantityOnHand"] = 0,
            };

            // Adds the book to the ProductCatalog table.
            await productCatalog.PutItemAsync(book);
        }

        /// <summary>
        /// Retrieves an item, a book, from the DynamoDB ProductCatalog table.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void RetrieveBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing RetrieveBook() ***");

            // Optional configuration.
            var config = new GetItemOperationConfig
            {
                AttributesToGet = new List<string> { "Id", "ISBN", "Title",
"Authors", "Price" },
                ConsistentRead = true,
            };

            Document document = await productCatalog.GetItemAsync(sampleBookId,
config);
            Console.WriteLine("RetrieveBook: Printing book retrieved...");
            PrintDocument(document);
```

```
        }

        /// <summary>
        /// Updates multiple attributes for a book and writes the changes to the
        /// DynamoDB table ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateMultipleAttributes(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\nUpdating multiple attributes....");
            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,

                // List of attribute updates.
                // The following replaces the existing authors list.
                ["Authors"] = new List<string> { "Author x", "Author y" },
                ["newAttribute"] = "New Value",
                ["ISBN"] = null, // Remove it.
            };

            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
                // Gets updated item in response.
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateMultipleAttributes: Printing item after
updates ...");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Updates a book item if it meets the specified criteria.
        /// </summary>
```

```
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateBookPriceConditionally(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing UpdateBookPriceConditionally()
***");

            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,
                ["Price"] = 29.99,
            };

            // For conditional price update, creating a condition expression.
            var expr = new Expression
            {
                ExpressionStatement = "Price = :val",
            };
            expr.ExpressionAttributeValues[":val"] = 19.00;

            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
                ConditionalExpression = expr,
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateBookPriceConditionally: Printing item whose
price was conditionally updated");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Deletes the book with the supplied Id value from the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
```

```
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task DeleteBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing DeleteBook() ***");

            // Optional configuration.
            var config = new DeleteItemOperationConfig
            {
                // Returns the deleted item.
                ReturnValues = ReturnValues.AllOldAttributes,
            };
            Document document = await
productCatalog.DeleteItemAsync(sampleBookId, config);
            Console.WriteLine("DeleteBook: Printing deleted just deleted...");

            PrintDocument(document);
        }

        /// <summary>
        /// Prints the information for the supplied DynamoDB document.
        /// </summary>
        /// <param name="updatedDocument">A DynamoDB document object.</param>
        public static void PrintDocument(Document updatedDocument)
        {
            if (updatedDocument is null)
            {
                return;
            }

            foreach (var attribute in updatedDocument.GetAttributeNames())
            {
                string stringValue = null;
                var value = updatedDocument[attribute];

                if (value is null)
                {
                    continue;
                }

                if (value is Primitive)
                {
```

```
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}", attribute,
stringValue);
            }
        }
    }
```

문서 모델을 사용하여 배치 쓰기 작업을 수행합니다.

```
    /// <summary>
    /// Shows how to use mid-level Amazon DynamoDB API calls to perform batch
    /// operations.
    /// </summary>
    public class MidLevelBatchWriteItem
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            await SingleTableBatchWrite(client);
            await MultiTableBatchWrite(client);
        }

        /// <summary>
        /// Perform a batch operation on a single DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB object.</param>
        public static async Task SingleTableBatchWrite(IAmazonDynamoDB client)
        {
            Table productCatalog = Table.LoadTable(client, "ProductCatalog");
            var batchWrite = productCatalog.CreateBatchWrite();
```

```
            var book1 = new Document
            {
                ["Id"] = 902,
                ["Title"] = "My book1 in batch write using .NET helper classes",
                ["ISBN"] = "902-11-11-1111",
                ["Price"] = 10,
                ["ProductCategory"] = "Book",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["Dimensions"] = "8.5x11x.5",
                ["InStock"] = new DynamoDBBool(true),
                ["QuantityOnHand"] = new DynamoDBNull(), // Quantity is unknown
at this time.
            };

            batchWrite.AddDocumentToPut(book1);

            // Specify delete item using overload that takes PK.
            batchWrite.AddKeyToDelete(12345);
            Console.WriteLine("Performing batch write in
SingleTableBatchWrite()");
            await batchWrite.ExecuteAsync();
        }

        /// <summary>
        /// Perform a batch operation involving multiple DynamoDB tables.
        /// </summary>
        /// <param name="client">An initialized DynamoDB client object.</param>
        public static async Task MultiTableBatchWrite(IAmazonDynamoDB client)
        {
            // Specify item to add in the Forum table.
            Table forum = Table.LoadTable(client, "Forum");
            var forumBatchWrite = forum.CreateBatchWrite();

            var forum1 = new Document
            {
                ["Name"] = "Test BatchWrite Forum",
                ["Threads"] = 0,
            };
            forumBatchWrite.AddDocumentToPut(forum1);

            // Specify item to add in the Thread table.
            Table thread = Table.LoadTable(client, "Thread");
```

```
            var threadBatchWrite = thread.CreateBatchWrite();

            var thread1 = new Document
            {
                ["ForumName"] = "S3 forum",
                ["Subject"] = "My sample question",
                ["Message"] = "Message text",
                ["KeywordTags"] = new List<string> { "S3", "Bucket" },
            };
            threadBatchWrite.AddDocumentToPut(thread1);

            // Specify item to delete from the Thread table.
            threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

            // Create multi-table batch.
            var superBatch = new MultiTableDocumentBatchWrite();
            superBatch.AddBatch(forumBatchWrite);
            superBatch.AddBatch(threadBatchWrite);
            Console.WriteLine("Performing batch write in
MultiTableBatchWrite()");

            // Execute the batch.
            await superBatch.ExecuteAsync();
        }
    }
```

문서 모델을 사용하여 테이블을 스캔합니다.

```
    /// <summary>
    /// Shows how to use mid-level Amazon DynamoDB API calls to scan a DynamoDB
    /// table for values.
    /// </summary>
    public class MidLevelScanOnly
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            Table productCatalogTable = Table.LoadTable(client,
"ProductCatalog");
```

```
            await FindProductsWithNegativePrice(productCatalogTable);
            await FindProductsWithNegativePriceWithConfig(productCatalogTable);
        }

        /// <summary>
        /// Retrieves any products that have a negative price in a DynamoDB
table.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePrice(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced <
0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            Search search = productCatalogTable.Scan(scanFilter);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePrice:
printing .....");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Finds any items in the ProductCatalog table using a DynamoDB
        /// configuration object.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePriceWithConfig(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced <
0.
```

```
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            var config = new ScanOperationConfig()
            {
                Filter = scanFilter,
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string> { "Title", "Id" },
            };

            Search search = productCatalogTable.Scan(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePriceWithConfig:
printing .............");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the details of the passed DynamoDB document object on the
        /// console.
        /// </summary>
        /// <param name="document">A DynamoDB document object.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];
                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
```

```
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

문서 모델을 사용하여 테이블을 쿼리하고 스캔합니다.

```
    /// <summary>
    /// Shows how to perform mid-level query procedures on an Amazon DynamoDB
    /// table.
    /// </summary>
    public class MidLevelQueryAndScan
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            // Query examples.
            Table replyTable = Table.LoadTable(client, "Reply");
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 2";

            await FindRepliesInLast15Days(replyTable);
            await FindRepliesInLast15DaysWithConfig(replyTable, forumName,
threadSubject);
            await FindRepliesPostedWithinTimePeriod(replyTable, forumName,
threadSubject);

            // Get Example.
            Table productCatalogTable = Table.LoadTable(client,
"ProductCatalog");
            int productId = 101;

            await GetProduct(productCatalogTable, productId);
```

```
        }

        /// <summary>
        /// Retrieves information about a product from the DynamoDB table
        /// ProductCatalog based on the product ID and displays the information
        /// on the console.
        /// </summary>
        /// <param name="tableName">The name of the table from which to retrieve
        /// product information.</param>
        /// <param name="productId">The ID of the product to retrieve.</param>
        public static async Task GetProduct(Table tableName, int productId)
        {
            Console.WriteLine("*** Executing GetProduct() ***");
            Document productDocument = await tableName.GetItemAsync(productId);
            if (productDocument != null)
            {
                PrintDocument(productDocument);
            }
            else
            {
                Console.WriteLine("Error: product " + productId + " does not
exist");
            }
        }

        /// <summary>
        /// Retrieves replies from the passed DynamoDB table object.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        public static async Task FindRepliesInLast15Days(
          Table table)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, "Id");
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            // Use Query overloads that take the minimum required query
parameters.
            Search search = table.Query(filter);

            do
            {
                var documentSet = await search.GetNextSetAsync();
```

```
                Console.WriteLine("\nFindRepliesInLast15Days:
printing .....");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Retrieve replies made during a specific time period.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">The subject of the thread, which we are
        /// searching for replies.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          Table table,
          string forumName,
          string threadSubject)
        {
            DateTime startDate = DateTime.UtcNow.Subtract(new TimeSpan(21, 0, 0,
0));
            DateTime endDate = DateTime.UtcNow.Subtract(new TimeSpan(1, 0, 0,
0));

            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadSubject);
            filter.AddCondition("ReplyDateTime", QueryOperator.Between,
startDate, endDate);

            var config = new QueryOperationConfig()
            {
                Limit = 2, // 2 items/page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
        {
          "Message",
          "ReplyDateTime",
          "PostedBy",
        },
```

```
                ConsistentRead = true,
                Filter = filter,
            };

            Search search = table.Query(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing
replies posted within dates: {0} and {1} .....", startDate, endDate);

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Perform a query for replies made in the last 15 days using a DynamoDB
        /// QueryOperationConfig object.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadName">The bane of the thread that we are searching
        /// for replies.</param>
        public static async Task FindRepliesInLast15DaysWithConfig(
          Table table,
          string forumName,
          string threadName)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadName);
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            var config = new QueryOperationConfig()
            {
                Filter = filter,
```

```
                // Optional parameters.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                  "Message",
                  "ReplyDateTime",
                  "PostedBy",
                },
                ConsistentRead = true,
            };

            Search search = table.Query(config);

            do
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
printing .....");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the contents of the passed DynamoDB document on the console.
        /// </summary>
        /// <param name="document">A DynamoDB document to display.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
```

```
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                        select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 DynamoDB용 상위 수준 객체 지속성 모델 사용

다음 코드 예제는 DynamoDB 및 AWS SDK용 객체 지속성 모델을 사용하여 생성, 읽기, 업데이트 및 삭제(CRUD) 및 배치 작업을 수행하는 방법을 보여 줍니다.

자세한 내용은 객체 지속성 모델을 참조하세요.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

상위 수준 객체 지속성 모델을 사용하여 CRUD 작업을 수행합니다.

```
    /// <summary>
    /// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
    /// table.
    /// </summary>
```

```
    public class HighLevelItemCrud
    {
        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await PerformCRUDOperations(context);
        }

        public static async Task PerformCRUDOperations(IDynamoDBContext context)
        {
            int bookId = 1001; // Some unique value.
            Book myBook = new Book
            {
                Id = bookId,
                Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
                Isbn = "111-1111111001",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
            };

            // Save the book to the ProductCatalog table.
            await context.SaveAsync(myBook);

            // Retrieve the book from the ProductCatalog table.
            Book bookRetrieved = await context.LoadAsync<Book>(bookId);

            // Update some properties.
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);
```

```
            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
            }
        }
    }
```

상위 수준 객체 지속성 모델을 사용하여 배치 쓰기 작업을 수행합니다.

```
    /// <summary>
    /// Performs high-level batch write operations to an Amazon DynamoDB table.
    /// This example was written using the AWS SDK for .NET version 3.7 and .NET
    /// Core 5.0.
    /// </summary>
    public class HighLevelBatchWriteItem
    {
        public static async Task SingleTableBatchWrite(IDynamoDBContext context)
        {
            Book book1 = new Book
            {
                Id = 902,
                InPublication = true,
                Isbn = "902-11-11-1111",
                PageCount = "100",
                Price = 10,
                ProductCategory = "Book",
                Title = "My book3 in batch write",
            };

            Book book2 = new Book
            {
                Id = 903,
                InPublication = true,
```

```
                Isbn = "903-11-11-1111",
                PageCount = "200",
                Price = 10,
                ProductCategory = "Book",
                Title = "My book4 in batch write",
            };

            var bookBatch = context.CreateBatchWrite<Book>();
            bookBatch.AddPutItems(new List<Book> { book1, book2 });

            Console.WriteLine("Adding two books to ProductCatalog table.");
            await bookBatch.ExecuteAsync();
        }

        public static async Task MultiTableBatchWrite(IDynamoDBContext context)
        {
            // New Forum item.
            Forum newForum = new Forum
            {
                Name = "Test BatchWrite Forum",
                Threads = 0,
            };
            var forumBatch = context.CreateBatchWrite<Forum>();
            forumBatch.AddPutItem(newForum);

            // New Thread item.
            Thread newThread = new Thread
            {
                ForumName = "S3 forum",
                Subject = "My sample question",
                KeywordTags = new List<string> { "S3", "Bucket" },
                Message = "Message text",
            };

            DynamoDBOperationConfig config = new DynamoDBOperationConfig();
            config.SkipVersionCheck = true;
            var threadBatch = context.CreateBatchWrite<Thread>(config);
            threadBatch.AddPutItem(newThread);
            threadBatch.AddDeleteKey("some partition key value", "some sort key
value");

            var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);
```

```
            Console.WriteLine("Performing batch write in
MultiTableBatchWrite().");
            await superBatch.ExecuteAsync();
        }

        public static async Task Main()
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);

            await SingleTableBatchWrite(context);
            await MultiTableBatchWrite(context);
        }
    }
```

상위 수준 객체 지속성 모델을 사용하여 임의 데이터를 테이블에 매핑합니다.

```
    /// <summary>
    /// Shows how to map arbitrary data to an Amazon DynamoDB table.
    /// </summary>
    public class HighLevelMappingArbitraryData
    {
        /// <summary>
        /// Creates a book, adds it to the DynamoDB ProductCatalog table,
retrieves
        /// the new book from the table, updates the dimensions and writes the
        /// changed item back to the table.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to write and
        /// read data from the table.</param>
        public static async Task AddRetrieveUpdateBook(IDynamoDBContext context)
        {
            // Create a book.
            DimensionType myBookDimensions = new DimensionType()
            {
                Length = 8M,
                Height = 11M,
                Thickness = 0.5M,
            };
```

```
            Book myBook = new Book
            {
                Id = 501,
                Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
                Isbn = "999-9999999999",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
                Dimensions = myBookDimensions,
            };

            // Add the book to the DynamoDB table ProductCatalog.
            await context.SaveAsync(myBook);

            // Retrieve the book.
            Book bookRetrieved = await context.LoadAsync<Book>(501);

            // Update the book dimensions property.
            bookRetrieved.Dimensions.Height += 1;
            bookRetrieved.Dimensions.Length += 1;
            bookRetrieved.Dimensions.Thickness += 0.2M;

            // Write the changed item to the table.
            await context.SaveAsync(bookRetrieved);
        }

        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await AddRetrieveUpdateBook(context);
        }
    }
```

상위 수준 객체 지속성 모델을 사용하여 테이블을 쿼리하고 스캔합니다.

```
    /// <summary>
    /// Shows how to perform high-level query and scan operations to Amazon
    /// DynamoDB tables.
    /// </summary>
    public class HighLevelQueryAndScan
```

```
    {
        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();

            DynamoDBContext context = new DynamoDBContext(client);

            // Get an item.
            await GetBook(context, 101);

            // Sample forum and thread to test queries.
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 1";

            // Sample queries.
            await FindRepliesInLast15Days(context, forumName, threadSubject);
            await FindRepliesPostedWithinTimePeriod(context, forumName,
threadSubject);

            // Scan table.
            await FindProductsPricedLessThanZero(context);
        }

        public static async Task GetBook(IDynamoDBContext context, int productId)
        {
            Book bookItem = await context.LoadAsync<Book>(productId);

            Console.WriteLine("\nGetBook: Printing result.....");
            Console.WriteLine($"Title: {bookItem.Title} \n ISBN:{bookItem.Isbn}
\n No. of pages: {bookItem.PageCount}");
        }

        /// <summary>
        /// Queries a DynamoDB table to find replies posted within the last 15
days.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the
query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">The thread object containing the query
parameters.</param>
        public static async Task FindRepliesInLast15Days(
          IDynamoDBContext context,
```

```
          string forumName,
          string threadSubject)
       {
           string replyId = $"{forumName} #{threadSubject}";
           DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);

           List<object> times = new List<object>();
           times.Add(twoWeeksAgoDate);

           List<ScanCondition> scs = new List<ScanCondition>();
           var sc = new ScanCondition("PostedBy", ScanOperator.GreaterThan,
times.ToArray());
           scs.Add(sc);

           var cfg = new DynamoDBOperationConfig
           {
               QueryFilter = scs,
           };

           AsyncSearch<Reply> response = context.QueryAsync<Reply>(replyId,
cfg);
           IEnumerable<Reply> latestReplies = await
response.GetRemainingAsync();

           Console.WriteLine("\nReplies in last 15 days:");

           foreach (Reply r in latestReplies)
           {

Console.WriteLine($"{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
           }
       }

       /// <summary>
       /// Queries for replies posted within a specific time period.
       /// </summary>
       /// <param name="context">The DynamoDB context used to perform the
query.</param>
       /// <param name="forumName">The name of the forum that we're interested
in.</param>
       /// <param name="threadSubject">Information about the subject that we're
       /// interested in.</param>
       public static async Task FindRepliesPostedWithinTimePeriod(
         IDynamoDBContext context,
```

```
          string forumName,
          string threadSubject)
       {
           string forumId = forumName + "#" + threadSubject;
           Console.WriteLine("\nReplies posted within time period:");

           DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
           DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);

           List<object> times = new List<object>();
           times.Add(startDate);
           times.Add(endDate);

           List<ScanCondition> scs = new List<ScanCondition>();
           var sc = new ScanCondition("LastPostedBy", ScanOperator.Between,
times.ToArray());
           scs.Add(sc);

           var cfg = new DynamoDBOperationConfig
           {
               QueryFilter = scs,
           };

           AsyncSearch<Reply> response = context.QueryAsync<Reply>(forumId,
cfg);
           IEnumerable<Reply> repliesInAPeriod = await
response.GetRemainingAsync();

           foreach (Reply r in repliesInAPeriod)
           {

Console.WriteLine("{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
           }
       }

       /// <summary>
       /// Queries the DynamoDB ProductCatalog table for products costing less
       /// than zero.
       /// </summary>
       /// <param name="context">The DynamoDB context object used to perform the
       /// query.</param>
       public static async Task FindProductsPricedLessThanZero(IDynamoDBContext
context)
       {
```

```
            int price = 0;

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc1 = new ScanCondition("Price", ScanOperator.LessThan, price);
            var sc2 = new ScanCondition("ProductCategory", ScanOperator.Equal,
"Book");
            scs.Add(sc1);
            scs.Add(sc2);

            AsyncSearch<Book> response = context.ScanAsync<Book>(scs);

            IEnumerable<Book> itemsWithWrongPrice = await
response.GetRemainingAsync();

            Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
result.....");

            foreach (Book r in itemsWithWrongPrice)
            {
                Console.WriteLine($"{r.Id}\t{r.Title}\t{r.Price}\t{r.Isbn}");
            }
        }
    }
```

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 DynamoDB에서 원자성 카운터 작업 사용

다음 코드 예제에서는 DynamoDB에서 원자성 카운터 작업을 사용하는 방법을 보여줍니다.

- ADD 및 SET 작업을 사용하여 원자적으로 카운터를 증가시킵니다.
- 존재하지 않을 수 있는 카운터를 안전하게 증가시킵니다.
- 카운터 작업에 대한 낙관적 잠금을 구현합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용한 원자성 카운터 작업을 보여줍니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Increments a counter using the ADD operation.
     *
     * <p>This method demonstrates how to use the ADD operation to atomically
     * increment a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterWithAdd(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #counterName :increment")
```

```
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Increments a counter using the SET operation.
     *
     * <p>This method demonstrates how to use the SET operation with an
expression
     * to increment a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterWithSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName = #counterName + :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build()))
```

```
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Increments a counter safely, handling the case where the counter doesn't
exist yet.
     *
     * <p>This method demonstrates how to use if_not_exists to safely increment a
counter
     * that may not exist yet.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterSafely(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName =
if_not_exists(#counterName, :zero) + :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build(),
                ":zero", AttributeValue.builder().n("0").build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();
```

```
        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Decrements a counter safely, ensuring it doesn't go below zero.
     *
     * <p>This method demonstrates how to use a condition expression to safely
     * decrement a counter without going below zero.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param decrementValue The value to decrement by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation or if
the counter would go below zero
     */
    public static UpdateItemResponse decrementCounterSafely(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int decrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName = #counterName - :decrement")
            .conditionExpression("#counterName >= :decrement")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":decrement",

AttributeValue.builder().n(String.valueOf(decrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }
```

```
   /**
    * Compares the ADD and SET approaches for incrementing counters.
    *
    * <p>This method demonstrates the differences between using ADD and SET
    * for incrementing counters in DynamoDB.
    *
    * @param dynamoDbClient The DynamoDB client
    * @param tableName The name of the DynamoDB table
    * @param key The key of the item to update
    * @return Map containing the comparison results
    */
   public static Map<String, Object> compareAddVsSet(
       DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

       Map<String, Object> results = new HashMap<>();

       try {
           // Reset counters to ensure a fair comparison
           UpdateItemRequest resetRequest = UpdateItemRequest.builder()
               .tableName(tableName)
               .key(key)
               .updateExpression("SET AddCounter = :zero, SetCounter = :zero")
               .expressionAttributeValues(
                   Map.of(":zero", AttributeValue.builder().n("0").build()))
               .build();

           dynamoDbClient.updateItem(resetRequest);

           // Increment with ADD
           long addStartTime = System.nanoTime();
           UpdateItemResponse addResponse =
incrementCounterWithAdd(dynamoDbClient, tableName, key, "AddCounter", 1);
           long addEndTime = System.nanoTime();
           long addDuration = addEndTime - addStartTime;

           // Increment with SET
           long setStartTime = System.nanoTime();
           UpdateItemResponse setResponse =
incrementCounterWithSet(dynamoDbClient, tableName, key, "SetCounter", 1);
           long setEndTime = System.nanoTime();
           long setDuration = setEndTime - setStartTime;
```

```
            // Record results
            results.put("addResponse", addResponse);
            results.put("setResponse", setResponse);
            results.put("addDuration", addDuration);
            results.put("setDuration", setDuration);
            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Gets the current value of a counter attribute.
     *
     * <p>Helper method to retrieve the current value of a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param counterName The name of the counter attribute
     * @return The counter value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Integer getCounterValue(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String counterName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(counterName)
            .build();

        // Perform the get operation
        GetItemResponse response = dynamoDbClient.getItem(request);

        // Return the counter value if it exists, otherwise null
        if (response.item() != null && response.item().containsKey(counterName))
{
```

```
            return Integer.parseInt(response.item().get(counterName).n());
        }

        return null;
    }
```

AWS SDK for Java 2.x를 사용한 원자성 카운터 작업의 사용 예입니다.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating atomic counter operations in
DynamoDB");

        try {
            // Example 1: Increment a counter using ADD
            System.out.println("\nExample 1: Incrementing a counter using ADD");
            UpdateItemResponse addResponse =
incrementCounterWithAdd(dynamoDbClient, tableName, key, "ViewCount", 1);

            System.out.println("Updated counter: " + addResponse.attributes());

            // Example 2: Increment a counter using SET
            System.out.println("\nExample 2: Incrementing a counter using SET");
            UpdateItemResponse setResponse =
incrementCounterWithSet(dynamoDbClient, tableName, key, "LikeCount", 1);

            System.out.println("Updated counter: " + setResponse.attributes());

            // Example 3: Increment a counter safely
            System.out.println("\nExample 3: Incrementing a counter safely");
            UpdateItemResponse safeResponse =
incrementCounterSafely(dynamoDbClient, tableName, key, "ShareCount", 1);

            System.out.println("Updated counter: " + safeResponse.attributes());

            // Example 4: Decrement a counter safely
            System.out.println("\nExample 4: Decrementing a counter safely");
            try {
```

```
                UpdateItemResponse decrementResponse =
                    decrementCounterSafely(dynamoDbClient, tableName, key,
"InventoryCount", 1);

                System.out.println("Updated counter: " +
decrementResponse.attributes());
            } catch (DynamoDbException e) {
                if (e.getMessage().contains("ConditionalCheckFailed")) {
                    System.out.println("Cannot decrement counter below zero");
                } else {
                    throw e;
                }
            }

            // Example 5: Compare ADD vs SET
            System.out.println("\nExample 5: Comparing ADD vs SET");
            Map<String, Object> comparison = compareAddVsSet(dynamoDbClient,
tableName, key);

            if ((boolean) comparison.get("success")) {
                System.out.println("ADD duration: " +
comparison.get("addDuration") + " ns");
                System.out.println("SET duration: " +
comparison.get("setDuration") + " ns");
                System.out.println("ADD response: " +
comparison.get("addResponse"));
                System.out.println("SET response: " +
comparison.get("setResponse"));
            } else {
                System.out.println("Comparison failed: " +
comparison.get("error"));
            }

            // Explain atomic counter operations
            System.out.println("\nKey points about DynamoDB atomic counter
operations:");
            System.out.println("1. Both ADD and SET can be used for atomic
counters");
            System.out.println("2. ADD is more concise for simple increments");
            System.out.println("3. SET with an expression is more flexible for
complex operations");
            System.out.println("4. Use if_not_exists to handle the case where the
counter doesn't exist yet");
```

```
            System.out.println("5. Use condition expressions to prevent counters
 from going below zero");
            System.out.println("6. Atomic operations are guaranteed to be
 isolated from other writes");
            System.out.println("7. ADD can only be used with number and set data
 types");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 UpdateItem을 참조하세요.

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용한 원자성 카운터 작업을 보여줍니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Increment a counter using the ADD operation.
 *
 * This function demonstrates using the ADD operation for atomic increments.
 * The ADD operation is atomic and is the recommended way to increment counters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithAdd(
```

```
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${counterName} :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter using the SET operation with an expression.
 *
 * This function demonstrates using the SET operation with an expression for
 increments.
 * While this approach works, it's less idiomatic for simple increments than
 using ADD.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithSet(
  config,
```

```
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with an expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter safely, handling the case where the counter might not
 exist.
 *
 * This function demonstrates using the if_not_exists function with SET to safely
 * increment a counter that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} defaultValue - The default value if the counter doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterSafely(
  config,
  tableName,
```

```
  key,
  counterName,
  incrementValue,
  defaultValue = 0
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} =
 if_not_exists(${counterName}, :default) + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter with optimistic locking to prevent race conditions.
 *
 * This function demonstrates using a condition expression to implement
 optimistic
 * locking, which prevents race conditions when multiple processes try to update
 * the same counter.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} expectedValue - The expected current value of the counter
 * @returns {Promise<Object>} - The response from DynamoDB
 */
```

```
async function incrementCounterWithLocking(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  expectedValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ConditionExpression: `${counterName} = :expected`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":expected": expectedValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      data: response
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        error: "Optimistic locking failed: the counter value has changed"
      };
    }
    // Re-throw other errors
    throw error;
  }
}
```

```
/**
 * Get the current value of a counter.
 *
 * Helper function to retrieve the current value of a counter attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @param {string} counterName - The name of the counter attribute
 * @returns {Promise<number|null>} - The current counter value or null if not
 found
 */
async function getCounterValue(
  config,
  tableName,
  key,
  counterName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the counter value if it exists, otherwise null
  return response.Item && counterName in response.Item
    ? response.Item[counterName]
    : null;
}
```

AWS SDK for JavaScript를 사용한 원자성 카운터 작업의 사용 예입니다.

```
/**
 * Example of how to use the atomic counter operations.
```

```
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };
  const counterName = "ViewCount";
  const incrementValue = 1;

  console.log("Demonstrating different approaches to increment counters in
 DynamoDB");

  try {
    // Example 1: Using ADD operation (recommended for simple increments)
    console.log("\nExample 1: Incrementing counter with ADD operation");
    const response1 = await incrementCounterWithAdd(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response1.Attributes[counterName]}`);

    // Example 2: Using SET operation with an expression
    console.log("\nExample 2: Incrementing counter with SET operation");
    const response2 = await incrementCounterWithSet(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response2.Attributes[counterName]}`);

    // Example 3: Safely incrementing a counter that might not exist
    console.log("\nExample 3: Safely incrementing counter that might not exist");
    const newKey = { ProductId: "P67890" };
    const response3 = await incrementCounterSafely(
      config,
      tableName,
      newKey,
```

```
      counterName,
      incrementValue,
      0
    );

    console.log(`Counter initialized and incremented to:
${response3.Attributes[counterName]}`);

    // Example 4: Incrementing with optimistic locking
    console.log("\nExample 4: Incrementing with optimistic locking");

    // First, get the current counter value
    const currentValue = await getCounterValue(config, tableName, key,
counterName);
    console.log(`Current counter value: ${currentValue}`);

    // Then, try to increment with optimistic locking
    const response4 = await incrementCounterWithLocking(
      config,
      tableName,
      key,
      counterName,
      incrementValue,
      currentValue
    );

    if (response4.success) {
      console.log(`Counter successfully incremented to:
${response4.data.Attributes[counterName]}`);
    } else {
      console.log(response4.error);
    }

    // Explain the differences between ADD and SET
    console.log("\nKey differences between ADD and SET for counter operations:");
    console.log("1. ADD is more concise and idiomatic for simple increments");
    console.log("2. SET with expressions is more flexible for complex
operations");
    console.log("3. Both operations are atomic and safe for concurrent updates");
    console.log("4. SET with if_not_exists is required when the attribute might
not exist");
    console.log("5. Optimistic locking can be added to either approach for
additional safety");
```

```
  } catch (error) {
    console.error("Error:", error);
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 UpdateItem을 참조하세요.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용한 원자성 카운터 작업을 보여줍니다.

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, Union


def increment_counter_with_add(
    table_name: str, key: Dict[str, Any], counter_name: str, increment_value: int
 = 1
) -> Dict[str, Any]:
    """
    Increment a counter attribute using the ADD operation.

    This function demonstrates the atomic ADD operation, which is ideal for
    incrementing counters without the risk of race conditions.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Use the ADD operation to atomically increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #counter :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def increment_counter_with_set(
    table_name: str, key: Dict[str, Any], counter_name: str, increment_value: int
 = 1
) -> Dict[str, Any]:
    """
    Increment a counter attribute using the SET operation with an expression.

    This function demonstrates using SET with an expression to increment a
 counter.
    While this works, it's generally recommended to use ADD for simple
 increments.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the SET operation with an expression to increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #counter = #counter + :increment",
```

```
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def increment_counter_safely(
    table_name: str,
    key: Dict[str, Any],
    counter_name: str,
    increment_value: int = 1,
    initial_value: int = 0,
) -> Dict[str, Any]:
    """
    Increment a counter attribute safely, handling the case where it might not
 exist.

    This function demonstrates a best practice for incrementing counters by using
    the if_not_exists function to handle the case where the counter doesn't exist
 yet.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.
        initial_value (int, optional): The initial value if the counter doesn't
 exist. Defaults to 0.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use SET with if_not_exists to safely increment the counter
    response = table.update_item(
        Key=key,
```

```
        UpdateExpression="SET #counter = if_not_exists(#counter, :initial)
 + :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value, ":initial":
 initial_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def atomic_conditional_increment(
    table_name: str,
    key: Dict[str, Any],
    counter_name: str,
    condition_attribute: str,
    condition_value: Any,
    increment_value: int = 1,
) -> Union[Dict[str, Any], None]:
    """
    Atomically increment a counter only if a condition is met.

    This function demonstrates combining atomic counter operations with
    conditional expressions for more complex update scenarios.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Optional[Dict[str, Any]]: The response from DynamoDB if successful, None
 if condition failed.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Use ADD with a condition expression
```

```
        response = table.update_item(
            Key=key,
            UpdateExpression="ADD #counter :increment",
            ConditionExpression="#condition = :value",
            ExpressionAttributeNames={"#counter": counter_name, "#condition":
 condition_attribute},
            ExpressionAttributeValues={":increment": increment_value, ":value":
 condition_value},
            ReturnValues="UPDATED_NEW",
        )
        return response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return None
        else:
            # Other error occurred
            raise
```

AWS SDK for Python (Boto3)를 사용한 원자성 카운터 작업의 사용 예입니다.

```
def example_usage():
    """Example of how to use the atomic counter operations functions."""
    # Example parameters
    table_name = "GameScores"
    key = {"UserId": "user123", "GameId": "game456"}
    counter_name = "Score"

    print("Example 1: Incrementing a counter with ADD operation")
    try:
        response = increment_counter_with_add(
            table_name=table_name, key=key, counter_name=counter_name,
 increment_value=10
        )
        print(
            f"Counter incremented successfully. New value:
 {response.get('Attributes', {}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error incrementing counter with ADD: {e}")
```

```
    print("\nExample 2: Incrementing a counter with SET operation")
    try:
        response = increment_counter_with_set(
            table_name=table_name, key=key, counter_name=counter_name,
increment_value=5
        )
        print(
            f"Counter incremented successfully. New value:
{response.get('Attributes', {}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error incrementing counter with SET: {e}")

    print("\nExample 3: Safely incrementing a counter that might not exist")
    try:
        new_key = {"UserId": "newuser789", "GameId": "game456"}
        response = increment_counter_safely(
            table_name=table_name,
            key=new_key,
            counter_name=counter_name,
            increment_value=15,
            initial_value=100,
        )
        print(
            f"Counter safely incremented. New value: {response.get('Attributes',
{}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error safely incrementing counter: {e}")

    print("\nExample 4: Conditional counter increment")
    try:
        # Fix for mypy: Handle the case where response might be None
        result = atomic_conditional_increment(
            table_name=table_name,
            key=key,
            counter_name="Achievements",
            condition_attribute="Level",
            condition_value=5,
            increment_value=1,
        )

        if result is not None:
```

```
            print(
                f"Conditional increment succeeded. New value:
{result.get('Attributes', {}).get('Achievements')}"
            )
        else:
            print("Conditional increment failed because condition was not met.")
        if response:
            print(
                f"Conditional increment succeeded. New value:
{response.get('Attributes', {}).get('Achievements')}"
            )
        else:
            print("Conditional increment failed because condition was not met.")
    except Exception as e:
        print(f"Error with conditional increment: {e}")

    print("\nComparison of ADD vs SET for counter operations:")
    print("1. ADD is specifically designed for atomic numeric increments and set
operations")
    print("2. SET with an expression can be used for more complex calculations")
    print("3. Both operations are atomic, preventing race conditions")
    print("4. ADD is more concise for simple increments")
    print("5. SET with if_not_exists() is recommended when the attribute might
not exist")
    print("6. For counters, ADD is generally preferred for clarity and
simplicity")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 UpdateItem를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 DynamoDB에서 조건부 작업 사용

다음 코드 예제에서는 DynamoDB에서 조건부 작업을 사용하는 방법을 보여줍니다.

- 데이터 덮어쓰기를 방지하기 위해 조건부 쓰기를 구현합니다.
- 조건 표현식을 사용하여 비즈니스 규칙을 적용합니다.
- 조건부 검사 실패를 원활하게 처리합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용한 조건부 작업을 보여줍니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Performs a conditional update on an item.
     *
     * <p>This method demonstrates how to use a condition expression to update an
 item
     * only if a specific condition is met.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param conditionAttribute The attribute to check in the condition
     * @param conditionValue The value to compare against
     * @param updateAttribute The attribute to update
     * @param updateValue The new value to set
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalUpdate(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String conditionAttribute,
        AttributeValue conditionValue,
```

```
        String updateAttribute,
        AttributeValue updateValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #updateAttr = :updateVal")
                .conditionExpression("#condAttr = :condVal")
                .expressionAttributeNames(Map.of(
                    "#condAttr", conditionAttribute,
                    "#updateAttr", updateAttribute))
                .expressionAttributeValues(Map.of(
                    ":condVal", conditionValue,
                    ":updateVal", updateValue))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Perform the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "Condition was met and update was performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "Condition was not met, update was not
performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
```

```
    }

    /**
     * Performs a conditional delete on an item.
     *
     * <p>This method demonstrates how to use a condition expression to delete an
item
     * only if a specific condition is met.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to delete
     * @param conditionAttribute The attribute to check in the condition
     * @param conditionValue The value to compare against
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalDelete(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String conditionAttribute,
        AttributeValue conditionValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the delete parameters
            DeleteItemRequest request = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .conditionExpression("#condAttr = :condVal")
                .expressionAttributeNames(Map.of("#condAttr",
conditionAttribute))
                .expressionAttributeValues(Map.of(":condVal", conditionValue))
                .returnValues(ReturnValue.ALL_OLD)
                .build();

            // Perform the delete operation
            DeleteItemResponse response = dynamoDbClient.deleteItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "Condition was met and delete was performed");
            result.put("attributes", response.attributes());
```

```
        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "Condition was not met, delete was not
performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }

    /**
     * Demonstrates optimistic locking using a version attribute.
     *
     * <p>This method shows how to implement optimistic locking by using a
version
     * attribute that is incremented with each update.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param versionAttribute The name of the version attribute
     * @return Map containing the operation result
     */
    public static Map<String, Object> optimisticLockingExample(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String versionAttribute) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Get the current version of the item
            GetItemRequest getRequest = GetItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .projectionExpression(versionAttribute)
                .build();
```

```
            GetItemResponse getResponse = dynamoDbClient.getItem(getRequest);

            // Check if the item exists
            if (getResponse.item() == null || !
getResponse.item().containsKey(versionAttribute)) {
                // Item doesn't exist or doesn't have a version attribute
                // Initialize with version 1
                UpdateItemRequest initRequest = UpdateItemRequest.builder()
                    .tableName(tableName)
                    .key(key)
                    .updateExpression("SET #verAttr = :newVer, #dataAttr
 = :data")
                    .expressionAttributeNames(Map.of("#verAttr",
 versionAttribute, "#dataAttr", "Data"))
                    .expressionAttributeValues(Map.of(
                        ":newVer", AttributeValue.builder().n("1").build(),
                        ":data", AttributeValue.builder().s("Initial
 data").build()))
                    .returnValues(ReturnValue.UPDATED_NEW)
                    .build();

                UpdateItemResponse initResponse =
 dynamoDbClient.updateItem(initRequest);

                result.put("operation", "initialize");
                result.put("success", true);
                result.put("attributes", initResponse.attributes());

                return result;
            }

            // Get the current version number
            int currentVersion =
                Integer.parseInt(getResponse.item().get(versionAttribute).n());
            int newVersion = currentVersion + 1;

            // Update the item with a condition on the version
            UpdateItemRequest updateRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #verAttr = :newVer, #dataAttr = :newData")
                .conditionExpression("#verAttr = :curVer")
```

```
                .expressionAttributeNames(Map.of("#verAttr", versionAttribute,
"#dataAttr", "Data"))
                .expressionAttributeValues(Map.of(
                    ":curVer",
                        AttributeValue.builder()
                            .n(String.valueOf(currentVersion))
                            .build(),
                    ":newVer",

AttributeValue.builder().n(String.valueOf(newVersion)).build(),
                    ":newData",
                        AttributeValue.builder()
                            .s("Updated data at version " + newVersion)
                            .build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse updateResponse =
dynamoDbClient.updateItem(updateRequest);

            // Record success result
            result.put("operation", "update");
            result.put("success", true);
            result.put("oldVersion", currentVersion);
            result.put("newVersion", newVersion);
            result.put("attributes", updateResponse.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to version mismatch
            result.put("operation", "update");
            result.put("success", false);
            result.put("message", "Version mismatch, another process may have
updated the item");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("operation", "update");
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
```

```
    }

    /**
     * Performs a conditional update with multiple conditions.
     *
     * <p>This method demonstrates how to use multiple conditions in a condition
expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param conditions Map of attribute names to values for conditions
     * @param updateAttribute The attribute to update
     * @param updateValue The new value to set
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalUpdateWithMultipleConditions(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        Map<String, AttributeValue> conditions,
        String updateAttribute,
        AttributeValue updateValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Build the condition expression and attribute names/values
            StringBuilder conditionExpression = new StringBuilder();
            Map<String, String> expressionAttributeNames = new HashMap<>();
            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();

            // Add update attribute
            expressionAttributeNames.put("#updateAttr", updateAttribute);
            expressionAttributeValues.put(":updateVal", updateValue);

            // Add conditions
            int i = 0;
            for (Map.Entry<String, AttributeValue> condition :
conditions.entrySet()) {
                String attrName = condition.getKey();
                AttributeValue attrValue = condition.getValue();
```

```
                String nameKey = "#cond" + i;
                String valueKey = ":val" + i;

                expressionAttributeNames.put(nameKey, attrName);
                expressionAttributeValues.put(valueKey, attrValue);

                // Add AND between conditions (except for the first one)
                if (i > 0) {
                    conditionExpression.append(" AND ");
                }

                conditionExpression.append(nameKey).append(" =
").append(valueKey);
                i++;
            }

            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #updateAttr = :updateVal")
                .conditionExpression(conditionExpression.toString())
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Perform the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "All conditions were met and update was
performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "One or more conditions were not met, update
was not performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
```

```
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }
```

AWS SDK for Java 2.x를 사용한 조건부 작업의 사용 예입니다.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating conditional operations in DynamoDB");

        try {
            // Example 1: Conditional update
            System.out.println("\nExample 1: Conditional update");
            Map<String, Object> updateResult = conditionalUpdate(
                dynamoDbClient,
                tableName,
                key,
                "InStock",
                AttributeValue.builder().bool(true).build(),
                "Status",
                AttributeValue.builder().s("Available").build());

            System.out.println("Update result: " + updateResult.get("message"));
            if ((boolean) updateResult.get("success")) {
                System.out.println("Updated attributes: " +
updateResult.get("attributes"));
            }

            // Example 2: Conditional delete
            System.out.println("\nExample 2: Conditional delete");
            Map<String, Object> deleteResult = conditionalDelete(
                dynamoDbClient,
                tableName,
```

```
                key,
                "Status",
                AttributeValue.builder().s("Discontinued").build());

            System.out.println("Delete result: " + deleteResult.get("message"));
            if ((boolean) deleteResult.get("success")) {
                System.out.println("Deleted item: " +
deleteResult.get("attributes"));
            }

            // Example 3: Optimistic locking
            System.out.println("\nExample 3: Optimistic locking");
            Map<String, Object> lockingResult =
optimisticLockingExample(dynamoDbClient, tableName, key, "Version");

            System.out.println("Optimistic locking result:");
            System.out.println("  Operation: " + lockingResult.get("operation"));
            System.out.println("  Success: " + lockingResult.get("success"));
            if (lockingResult.get("operation").equals("update") && (boolean)
lockingResult.get("success")) {
                System.out.println("  Old version: " +
lockingResult.get("oldVersion"));
                System.out.println("  New version: " +
lockingResult.get("newVersion"));
            }
            System.out.println("  Attributes: " +
lockingResult.get("attributes"));

            // Example 4: Multiple conditions
            System.out.println("\nExample 4: Multiple conditions");
            Map<String, AttributeValue> conditions = new HashMap<>();
            conditions.put("Price",
AttributeValue.builder().n("199.99").build());
            conditions.put("Category",
AttributeValue.builder().s("Electronics").build());

            Map<String, Object> multiConditionResult =
conditionalUpdateWithMultipleConditions(
                dynamoDbClient,
                tableName,
                key,
                conditions,
                "OnSale",
                AttributeValue.builder().bool(true).build());
```

```
            System.out.println("Multiple conditions result: " +
multiConditionResult.get("message"));
            if ((boolean) multiConditionResult.get("success")) {
                System.out.println("Updated attributes: " +
multiConditionResult.get("attributes"));
            }

            // Explain conditional operations
            System.out.println("\nKey points about DynamoDB conditional
operations:");
            System.out.println("1. Conditional operations only succeed if the
condition is met");
            System.out.println("2. ConditionalCheckFailedException is thrown when
the condition fails");
            System.out.println("3. No changes are made to the item if the
condition fails");
            System.out.println("4. Conditions can be used with update, delete,
and put operations");
            System.out.println("5. Multiple conditions can be combined with AND
and OR");
            System.out.println("6. Optimistic locking can be implemented using a
version attribute");
            System.out.println(
                "7. Conditional operations consume the same amount of write
capacity whether they succeed or fail");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - DeleteItem
  - PutItem
  - UpdateItem

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용한 조건부 작업을 보여줍니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  DeleteCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Perform a conditional update operation.
 *
 * This function demonstrates how to update an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @param {string} updateAttribute - The attribute to update
 * @param {any} updateValue - The new value to set
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalUpdate(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue,
  updateAttribute,
  updateValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
```

```
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateAttribute} = :value`,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":value": updateValue,
      ":condition": conditionValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Condition was met and update was performed",
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, update was not performed",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Perform a conditional delete operation.
 *
 * This function demonstrates how to delete an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to delete
 * @param {string} conditionAttribute - The attribute to check in the condition
```

```
 * @param {any} conditionValue - The value to compare against
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalDelete(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the delete parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":condition": conditionValue
    },
    ReturnValues: "ALL_OLD"
  };

  try {
    // Perform the delete operation
    const response = await docClient.send(new DeleteCommand(params));

    return {
      success: true,
      message: "Condition was met and item was deleted",
      deletedItem: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, item was not deleted",
        error: "ConditionalCheckFailedException"
      };
    }
```

```
    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement optimistic locking with a version number.
 *
 * This function demonstrates how to use a version number for optimistic locking
 * to prevent race conditions when multiple processes update the same item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {Object} updates - The attributes to update
 * @param {number} expectedVersion - The expected current version number
 * @returns {Promise<Object>} - Result of the operation
 */
async function updateWithOptimisticLocking(
  config,
  tableName,
  key,
  updates,
  expectedVersion
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression
  const updateExpressions = [];
  const expressionAttributeValues = {
    ":expectedVersion": expectedVersion,
    ":newVersion": expectedVersion + 1
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([attribute, value], index) => {
    updateExpressions.push(`${attribute} = :val${index}`);
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Add the version update
  updateExpressions.push("version = :newVersion");
```

```
  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateExpressions.join(", ")}`,
    ConditionExpression: "version = :expectedVersion",
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Update succeeded with optimistic locking",
      newVersion: expectedVersion + 1,
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Optimistic locking failed: the item was modified by another
 process",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement a conditional write that creates an item only if it doesn't exist.
 *
 * This function demonstrates how to use attribute_not_exists to create an item
 * only if it doesn't already exist (similar to an "INSERT IF NOT EXISTS"
 operation).
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} item - The item to create
 * @returns {Promise<Object>} - Result of the operation
 */
async function createIfNotExists(
  config,
  tableName,
  item
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Extract the primary key attributes
  const keyAttributes = Object.keys(item).filter(attr =>
    attr === "id" || attr === "ID" || attr === "Id" ||
    attr.endsWith("Id") || attr.endsWith("ID") ||
    attr.endsWith("Key")
  );

  if (keyAttributes.length === 0) {
    throw new Error("Could not determine primary key attributes");
  }

  // Create a condition expression that checks if the item doesn't exist
  const conditionExpression = `attribute_not_exists(${keyAttributes[0]})`;

  // Define the put parameters with a condition expression
  const params = {
    TableName: tableName,
    Item: item,
    ConditionExpression: conditionExpression
  };

  try {
    // Perform the put operation
    await docClient.send(new PutCommand(params));

    return {
      success: true,
      message: "Item was created because it didn't exist",
      item
    };
```

```
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Item already exists, creation was skipped",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
```

```
  return response.Item || null;
}
```

AWS SDK for JavaScript를 사용한 조건부 작업의 사용 예입니다.

```
/**
 * Example of how to use conditional operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating conditional operations in DynamoDB");

  try {
    // Example 1: Conditional update based on attribute value
    console.log("\nExample 1: Conditional update based on attribute value");
    const updateResult = await conditionalUpdate(
      config,
      tableName,
      key,
      "Category",
      "Electronics",
      "Price",
      299.99
    );

    console.log(`Result: ${updateResult.message}`);
    if (updateResult.success) {
      console.log("Updated attributes:", updateResult.updatedAttributes);
    }

    // Example 2: Conditional delete based on attribute value
    console.log("\nExample 2: Conditional delete based on attribute value");
    const deleteResult = await conditionalDelete(
      config,
      tableName,
      key,
      "InStock",
      false
```

```
    );

    console.log(`Result: ${deleteResult.message}`);
    if (deleteResult.success) {
      console.log("Deleted item:", deleteResult.deletedItem);
    }

    // Example 3: Optimistic locking with version number
    console.log("\nExample 3: Optimistic locking with version number");

    // First, get the current item to check its version
    const currentItem = await getItem(config, tableName, { ProductId:
"P67890" });
    const currentVersion = currentItem ? (currentItem.version || 0) : 0;

    console.log(`Current version: ${currentVersion}`);

    // Then, update with optimistic locking
    const lockingResult = await updateWithOptimisticLocking(
      config,
      tableName,
      { ProductId: "P67890" },
      {
        Name: "Updated Product Name",
        Description: "This is an updated description"
      },
      currentVersion
    );

    console.log(`Result: ${lockingResult.message}`);
    if (lockingResult.success) {
      console.log(`New version: ${lockingResult.newVersion}`);
      console.log("Updated attributes:", lockingResult.updatedAttributes);
    }

    // Example 4: Create item only if it doesn't exist
    console.log("\nExample 4: Create item only if it doesn't exist");
    const createResult = await createIfNotExists(
      config,
      tableName,
      {
        ProductId: "P99999",
        Name: "New Product",
        Category: "Accessories",
```

```
        Price: 19.99,
        InStock: true
      }
    );

    console.log(`Result: ${createResult.message}`);
    if (createResult.success) {
      console.log("Created item:", createResult.item);
    }

    // Explain conditional operations
    console.log("\nKey points about conditional operations:");
    console.log("1. Conditional operations only succeed if the condition is
 met");
    console.log("2. ConditionalCheckFailedException indicates the condition
 wasn't met");
    console.log("3. Optimistic locking prevents race conditions in concurrent
 updates");
    console.log("4. attribute_exists and attribute_not_exists are useful for
 checking if attributes are present");
    console.log("5. Conditional operations are atomic - they either succeed
 completely or fail completely");
    console.log("6. You can use any valid comparison operators and functions in
 condition expressions");
    console.log("7. Conditional operations don't consume write capacity if the
 condition fails");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - DeleteItem
  - PutItem
  - UpdateItem

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용한 조건부 작업을 보여줍니다.

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, Optional, Tuple, Union


def conditional_update(
    table_name: str,
    key: Dict[str, Any],
    condition_attribute: str,
    condition_value: Any,
    update_attribute: str,
    update_value: Any,
) -> Tuple[bool, Optional[Dict[str, Any]]]:
    """
    Update an item only if a condition is met.

    This function demonstrates how to perform a conditional update operation
    and determine if the condition was met.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.

    Returns:
        Tuple[bool, Optional[Dict[str, Any]]]: A tuple containing:
            - A boolean indicating if the update succeeded
            - The response from DynamoDB if successful, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Perform the conditional update
```

```
        response = table.update_item(
            Key=key,
            UpdateExpression="SET #update_attr = :update_val",
            ConditionExpression="#cond_attr = :cond_val",
            ExpressionAttributeNames={
                "#update_attr": update_attribute,
                "#cond_attr": condition_attribute,
            },
            ExpressionAttributeValues={":update_val": update_value, ":cond_val":
 condition_value},
            ReturnValues="UPDATED_NEW",
        )
        # Update succeeded, condition was met
        return True, response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return False, None
        else:
            # Other error occurred
            raise


def conditional_delete(
    table_name: str, key: Dict[str, Any], condition_attribute: str,
 condition_value: Any
) -> bool:
    """
    Delete an item only if a condition is met.

    This function demonstrates how to perform a conditional delete operation
    and determine if the condition was met.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to delete.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.

    Returns:
        bool: True if the delete succeeded (condition was met), False otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
```

```
    table = dynamodb.Table(table_name)

    try:
        # Perform the conditional delete
        table.delete_item(
            Key=key,
            ConditionExpression="#attr = :val",
            ExpressionAttributeNames={"#attr": condition_attribute},
            ExpressionAttributeValues={":val": condition_value},
        )
        # Delete succeeded, condition was met
        return True
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return False
        else:
            # Other error occurred
            raise


def optimistic_locking_update(
    table_name: str,
    key: Dict[str, Any],
    version_attribute: str,
    update_attribute: str,
    update_value: Any,
) -> Tuple[bool, Optional[Dict[str, Any]]]:
    """
    Update an item using optimistic locking with a version attribute.

    This function demonstrates how to implement optimistic locking using
    a version attribute that is incremented with each update.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        version_attribute (str): The name of the version attribute.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.

    Returns:
        Tuple[bool, Optional[Dict[str, Any]]]: A tuple containing:
            - A boolean indicating if the update succeeded
```

```
            - The response from DynamoDB if successful, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # First, get the current version
    try:
        response = table.get_item(
            Key=key,
            ProjectionExpression=f"#{version_attribute}",
            ExpressionAttributeNames={f"#{version_attribute}":
version_attribute},
        )

        item = response.get("Item", {})
        current_version = item.get(version_attribute, 0)

        # Now, try to update with a condition on the version
        try:
            update_response = table.update_item(
                Key=key,
                UpdateExpression=f"SET #{update_attribute} = :update_val,
#{version_attribute} = :new_version",
                ConditionExpression=f"#{version_attribute} = :current_version",
                ExpressionAttributeNames={
                    f"#{update_attribute}": update_attribute,
                    f"#{version_attribute}": version_attribute,
                },
                ExpressionAttributeValues={
                    ":update_val": update_value,
                    ":current_version": current_version,
                    ":new_version": current_version + 1,
                },
                ReturnValues="UPDATED_NEW",
            )
            # Update succeeded
            return True, update_response
        except ClientError as e:
            if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
                # Version has changed, optimistic locking failed
                return False, None
            else:
                # Other error occurred
```

```
                raise
    except ClientError:
        # Error getting the item
        raise


def conditional_check_and_update(
    table_name: str,
    key: Dict[str, Any],
    check_attribute: str,
    check_value: Any,
    update_attribute: str,
    update_value: Any,
    create_if_not_exists: bool = False,
) -> Union[Dict[str, Any], None]:
    """
    Check if an attribute has a specific value and update another attribute if it
 does.

    This function demonstrates a more complex conditional update that can also
    create the item if it doesn't exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        check_attribute (str): The attribute to check in the condition.
        check_value (Any): The value to compare against.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.
        create_if_not_exists (bool, optional): Whether to create the item if it
 doesn't exist.

    Returns:
        Union[Dict[str, Any], None]: The response from DynamoDB if successful,
 None otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        if create_if_not_exists:
            # Use attribute_not_exists to create the item if it doesn't exist
```

```
            condition_expression = "attribute_not_exists(#pk) OR #check_attr
= :check_val"
            update_expression = "SET #update_attr = :update_val, #check_attr =
if_not_exists(#check_attr, :check_val)"

            # Get the partition key name from the key dictionary
            pk_name = next(iter(key))

            expression_attribute_names = {
                "#pk": pk_name,
                "#check_attr": check_attribute,
                "#update_attr": update_attribute,
            }
        else:
            # Only update if the check attribute has the expected value
            condition_expression = "#check_attr = :check_val"
            update_expression = "SET #update_attr = :update_val"

            expression_attribute_names = {
                "#check_attr": check_attribute,
                "#update_attr": update_attribute,
            }

        # Perform the conditional update
        response = table.update_item(
            Key=key,
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeNames=expression_attribute_names,
            ExpressionAttributeValues={":check_val": check_value, ":update_val":
update_value},
            ReturnValues="UPDATED_NEW",
        )
        return response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return None
        else:
            # Other error occurred
            raise
```

AWS SDK for Python (Boto3)를 사용한 조건부 작업의 사용 예입니다.

```
def example_usage():
    """Example of how to use the conditional operations functions."""
    # Example parameters
    table_name = "Products"
    key = {"ProductId": "prod123"}

    print("Example 1: Conditional Update")
    try:
        # Update the price only if the current stock is greater than 10
        success, response = conditional_update(
            table_name=table_name,
            key=key,
            condition_attribute="Stock",
            condition_value=10,
            update_attribute="Price",
            update_value=99.99,
        )

        if success:
            # Fix for mypy: Handle the case where response might be None
            attributes = {} if response is None else response.get("Attributes",
 {})
            print(f"Update succeeded! New values: {attributes}")
        else:
            print("Update failed because the condition was not met.")
    except Exception as e:
        print(f"Error during conditional update: {e}")

    print("\nExample 2: Conditional Delete")
    try:
        # Delete the product only if it's discontinued
        success = conditional_delete(
            table_name=table_name,
            key=key,
            condition_attribute="Status",
            condition_value="Discontinued",
        )

        if success:
```

```
            print("Delete succeeded! The item was deleted.")
        else:
            print("Delete failed because the condition was not met.")
    except Exception as e:
        print(f"Error during conditional delete: {e}")

    print("\nExample 3: Optimistic Locking")
    try:
        # Update with optimistic locking using a version attribute
        success, response = optimistic_locking_update(
            table_name=table_name,
            key=key,
            version_attribute="Version",
            update_attribute="Description",
            update_value="Updated product description",
        )

        if success:
            # Fix for mypy: Handle the case where response might be None
            attributes = {} if response is None else response.get("Attributes",
{})
            print(f"Optimistic locking update succeeded! New values:
{attributes}")
        else:
            print("Optimistic locking update failed because the version has
changed.")
    except Exception as e:
        print(f"Error during optimistic locking update: {e}")

    print("\nExample 4: Conditional Check and Update")
    try:
        # Update the featured status if the product is in stock
        response = conditional_check_and_update(
            table_name=table_name,
            key=key,
            check_attribute="InStock",
            check_value=True,
            update_attribute="Featured",
            update_value=True,
            create_if_not_exists=True,
        )

        if response:
            print(
```

```
                f"Conditional check and update succeeded! New values:
{response.get('Attributes', {})}"
            )
        else:
            print("Conditional check and update failed because the condition was
not met.")
    except Exception as e:
        print(f"Error during conditional check and update: {e}")

    print("\nUnderstanding Conditional Operations in DynamoDB:")
    print("1. Conditional operations help maintain data integrity")
    print("2. They prevent race conditions in concurrent environments")
    print("3. Failed conditions result in ConditionalCheckFailedException")
    print("4. No DynamoDB capacity is consumed when conditions fail")
    print("5. Optimistic locking is a common pattern using version attributes")
    print("6. Conditions can be combined with logical operators (AND, OR, NOT)")
    print("7. Conditions can use comparison operators (=, <>, <, <=, >, >=)")
    print(
        "8. attribute_exists() and attribute_not_exists() are useful for checking
attribute presence"
    )
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하세요.
  - DeleteItem
  - PutItem
  - UpdateItem

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS SDK를 사용하여 DynamoDB에서 표현식 속성 이름 사용

다음 코드 예제에서는 DynamoDB에서 표현식 속성 이름을 사용하는 방법을 보여줍니다.

- DynamoDB 표현식에서 예약어로 작업합니다.
- 표현식 속성 이름 자리 표시자를 사용합니다.
- 속성 이름의 특수 문자를 처리합니다.

Java

SDK for Java 2.x

AWS SDK for Java 2.x를 사용한 표현식 속성 이름을 보여줍니다.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Updates an attribute that is a reserved word in DynamoDB.
     *
     * <p>This method demonstrates how to use expression attribute names to
 update
     * attributes that are reserved words in DynamoDB.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param reservedWordAttribute The reserved word attribute to update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String reservedWordAttribute,
        AttributeValue value) {

        // Define the update parameters using expression attribute names
```

```
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attr = :value")
            .expressionAttributeNames(Map.of("#attr", reservedWordAttribute))
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Updates an attribute that contains special characters.
     *
     * <p>This method demonstrates how to use expression attribute names to
update
     * attributes that contain special characters.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param specialCharAttribute The attribute with special characters to
update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateSpecialCharacterAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String specialCharAttribute,
        AttributeValue value) {

        // Define the update parameters using expression attribute names
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attr = :value")
            .expressionAttributeNames(Map.of("#attr", specialCharAttribute))
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
```

```
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Queries items using an attribute that is a reserved word.
     *
     * <p>This method demonstrates how to use expression attribute names in a
query
     * when the attribute is a reserved word.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key
     * @param reservedWordAttribute The reserved word attribute to filter on
     * @param value The value to compare against
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse queryWithReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String reservedWordAttribute,
        AttributeValue value) {

        // Define the query parameters using expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attr", reservedWordAttribute);

        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);
        expressionAttributeValues.put(":value", value);

        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression("#attr = :value")
            .expressionAttributeNames(expressionAttributeNames)
```

```
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Updates a nested attribute with a path that contains reserved words.
     *
     * <p>This method demonstrates how to use expression attribute names to
update
     * nested attributes where the path contains reserved words.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributePath The path to the nested attribute as an array
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateNestedReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        List<String> attributePath,
        AttributeValue value) {

        // Create expression attribute names for each part of the path
        Map<String, String> expressionAttributeNames = new HashMap<>();
        for (int i = 0; i < attributePath.size(); i++) {
            expressionAttributeNames.put("#attr" + i, attributePath.get(i));
        }

        // Build the attribute path using the expression attribute names
        StringBuilder attributePathExpression = new StringBuilder();
        for (int i = 0; i < attributePath.size(); i++) {
            if (i > 0) {
                attributePathExpression.append(".");
            }
            attributePathExpression.append("#attr").append(i);
        }
```

```
        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET " + attributePathExpression.toString() + "
= :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Scans a table with multiple attribute name placeholders.
     *
     * <p>This method demonstrates how to use multiple expression attribute names
     * in a complex filter expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param filters Object mapping attribute names to filter values
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static ScanResponse scanWithMultipleAttributeNames(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> filters) {

        // Create expression attribute names and values
        Map<String, String> expressionAttributeNames = new HashMap<>();
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        StringBuilder filterExpression = new StringBuilder();

        // Build the filter expression
        int index = 0;
        for (Map.Entry<String, AttributeValue> entry : filters.entrySet()) {
            String attrName = entry.getKey();
            AttributeValue attrValue = entry.getValue();

            String nameKey = "#attr" + index;
            String valueKey = ":val" + index;
```

```
            expressionAttributeNames.put(nameKey, attrName);
            expressionAttributeValues.put(valueKey, attrValue);

            // Add AND between conditions (except for the first one)
            if (index > 0) {
                filterExpression.append(" AND ");
            }

            filterExpression.append(nameKey).append(" = ").append(valueKey);
            index++;
        }

        // Define the scan parameters
        ScanRequest request = ScanRequest.builder()
            .tableName(tableName)
            .filterExpression(filterExpression.toString())
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the scan operation
        return dynamoDbClient.scan(request);
    }
```

AWS SDK for Java 2.x를 사용한 표현식 속성 이름의 사용 예시입니다.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating expression attribute names in
DynamoDB");

        try {
            // Example 1: Update an attribute that is a reserved word
            System.out.println("\nExample 1: Updating an attribute that is a
reserved word");
            UpdateItemResponse response1 = updateReservedWordAttribute(
                dynamoDbClient,
```

```
                tableName,
                key,
                "Size", // "SIZE" is a reserved word in DynamoDB
                AttributeValue.builder().s("Large").build());

            System.out.println("Updated attribute: " + response1.attributes());

            // Example 2: Update an attribute with special characters
            System.out.println("\nExample 2: Updating an attribute with special
characters");
            UpdateItemResponse response2 = updateSpecialCharacterAttribute(
                dynamoDbClient,
                tableName,
                key,
                "Product-Type", // Contains a hyphen, which is a special
character
                AttributeValue.builder().s("Electronics").build());

            System.out.println("Updated attribute: " + response2.attributes());

            // Example 3: Query with a reserved word attribute
            System.out.println("\nExample 3: Querying with a reserved word
attribute");
            QueryResponse response3 = queryWithReservedWordAttribute(
                dynamoDbClient,
                tableName,
                "Category",
                AttributeValue.builder().s("Electronics").build(),
                "Count", // "COUNT" is a reserved word in DynamoDB
                AttributeValue.builder().n("10").build());

            System.out.println("Found " + response3.count() + " items");

            // Example 4: Update a nested attribute with reserved words in the
path
            System.out.println("\nExample 4: Updating a nested attribute with
reserved words in the path");
            UpdateItemResponse response4 = updateNestedReservedWordAttribute(
                dynamoDbClient,
                tableName,
                key,
                Arrays.asList("Dimensions", "Size", "Height"), // "SIZE" is a
reserved word
                AttributeValue.builder().n("30").build());
```

```
            System.out.println("Updated nested attribute: " +
 response4.attributes());

            // Example 5: Scan with multiple attribute name placeholders
            System.out.println("\nExample 5: Scanning with multiple attribute
 name placeholders");
            Map<String, AttributeValue> filters = new HashMap<>();
            filters.put("Size", AttributeValue.builder().s("Large").build());
            filters.put("Count", AttributeValue.builder().n("10").build());
            filters.put(
                "Product-Type",
 AttributeValue.builder().s("Electronics").build());

            ScanResponse response5 =
 scanWithMultipleAttributeNames(dynamoDbClient, tableName, filters);

            System.out.println("Found " + response5.count() + " items");

            // Show some common reserved words
            System.out.println("\nSome common DynamoDB reserved words:");
            List<String> commonReservedWords = getDynamoDBReservedWords();
            System.out.println(String.join(", ", commonReservedWords));

            // Explain expression attribute names
            System.out.println("\nKey points about expression attribute names:");
            System.out.println("1. Use expression attribute names (#name) for
 reserved words");
            System.out.println("2. Use expression attribute names for attributes
 with special characters");
            System.out.println(
                "3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
            System.out.println("4. Expression attribute names are required for
 nested attributes with reserved words");
            System.out.println("5. You can use multiple expression attribute
 names in a single expression");
            System.out.println("6. Expression attribute names are case-
sensitive");
            System.out.println("7. Expression attribute names are only used in
 expressions, not in the actual data");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
```

```
            e.printStackTrace();
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - Query
  - UpdateItem

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript를 사용한 표현식 속성 이름을 보여줍니다.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  QueryCommand,
  ScanCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an attribute that is a reserved word in DynamoDB.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that are reserved words in DynamoDB.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} reservedWordAttribute - The reserved word attribute to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateReservedWordAttribute(
  config,
  tableName,
  key,
  reservedWordAttribute,
```

```
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an attribute that contains special characters.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that contain special characters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} specialCharAttribute - The attribute with special characters
 to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateSpecialCharacterAttribute(
  config,
  tableName,
  key,
```

```
  specialCharAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": specialCharAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Query items using an attribute that is a reserved word.
 *
 * This function demonstrates how to use expression attribute names in a query
 * when the attribute is a reserved word.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key attribute
 * @param {any} partitionKeyValue - The value of the partition key
 * @param {string} reservedWordAttribute - The reserved word attribute to filter
 on
 * @param {any} value - The value to compare against
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function queryWithReservedWordAttribute(
  config,
```

```
  tableName,
  partitionKeyName,
  partitionKeyValue,
  reservedWordAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the query parameters using expression attribute names
  const params = {
    TableName: tableName,
    KeyConditionExpression: "#pkName = :pkValue",
    FilterExpression: "#attr = :value",
    ExpressionAttributeNames: {
      "#pkName": partitionKeyName,
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":pkValue": partitionKeyValue,
      ":value": value
    }
  };

  // Perform the query operation
  const response = await docClient.send(new QueryCommand(params));

  return response;
}

/**
 * Update a nested attribute with a path that contains reserved words.
 *
 * This function demonstrates how to use expression attribute names to update
 * nested attributes where the path contains reserved words.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} attributePath - The path to the nested attribute as an array
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
```

```
async function updateNestedReservedWordAttribute(
  config,
  tableName,
  key,
  attributePath,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names for each part of the path
  const expressionAttributeNames = {};
  for (let i = 0; i < attributePath.length; i++) {
    expressionAttributeNames[`#attr${i}`] = attributePath[i];
  }

  // Build the attribute path using the expression attribute names
  const attributePathExpression = attributePath
    .map((_, i) => `#attr${i}`)
    .join(".");

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${attributePathExpression} = :value`,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Scan a table with multiple attribute name placeholders.
 *
 * This function demonstrates how to use multiple expression attribute names
```

```
 * in a complex filter expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} filters - Object mapping attribute names to filter values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function scanWithMultipleAttributeNames(
  config,
  tableName,
  filters
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names and values
  const expressionAttributeNames = {};
  const expressionAttributeValues = {};
  const filterConditions = [];

  // Build the filter expression
  Object.entries(filters).forEach(([attrName, value], index) => {
    const nameKey = `#attr${index}`;
    const valueKey = `:val${index}`;

    expressionAttributeNames[nameKey] = attrName;
    expressionAttributeValues[valueKey] = value;
    filterConditions.push(`${nameKey} = ${valueKey}`);
  });

  // Join the filter conditions with AND
  const filterExpression = filterConditions.join(" AND ");

  // Define the scan parameters
  const params = {
    TableName: tableName,
    FilterExpression: filterExpression,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: expressionAttributeValues
  };

  // Perform the scan operation
  const response = await docClient.send(new ScanCommand(params));
```

```
  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

AWS SDK for JavaScript를 사용한 표현식 속성 이름의 사용 예시입니다.

```
/**
 * Example of how to use expression attribute names.
 */
async function exampleUsage() {
```

```
// Example parameters
const config = { region: "us-west-2" };
const tableName = "Products";
const key = { ProductId: "P12345" };

console.log("Demonstrating expression attribute names in DynamoDB");

try {
  // Example 1: Update an attribute that is a reserved word
  console.log("\nExample 1: Updating an attribute that is a reserved word");
  const response1 = await updateReservedWordAttribute(
    config,
    tableName,
    key,
    "Size", // "SIZE" is a reserved word in DynamoDB
    "Large"
  );

  console.log("Updated attribute:", response1.Attributes);

  // Example 2: Update an attribute with special characters
  console.log("\nExample 2: Updating an attribute with special characters");
  const response2 = await updateSpecialCharacterAttribute(
    config,
    tableName,
    key,
    "Product-Type", // Contains a hyphen, which is a special character
    "Electronics"
  );

  console.log("Updated attribute:", response2.Attributes);

  // Example 3: Query with a reserved word attribute
  console.log("\nExample 3: Querying with a reserved word attribute");
  const response3 = await queryWithReservedWordAttribute(
    config,
    tableName,
    "Category",
    "Electronics",
    "Count", // "COUNT" is a reserved word in DynamoDB
    10
  );

  console.log(`Found ${response3.Items.length} items`);
```

```
   // Example 4: Update a nested attribute with reserved words in the path
   console.log("\nExample 4: Updating a nested attribute with reserved words in
the path");
   const response4 = await updateNestedReservedWordAttribute(
     config,
     tableName,
     key,
     ["Dimensions", "Size", "Height"], // "SIZE" is a reserved word
     30
   );

   console.log("Updated nested attribute:", response4.Attributes);

   // Example 5: Scan with multiple attribute name placeholders
   console.log("\nExample 5: Scanning with multiple attribute name
placeholders");
   const response5 = await scanWithMultipleAttributeNames(
     config,
     tableName,
     {
       "Size": "Large",
       "Count": 10,
       "Product-Type": "Electronics"
     }
   );

   console.log(`Found ${response5.Items.length} items`);

   // Get the final state of the item
   console.log("\nFinal state of the item:");
   const item = await getItem(config, tableName, key);
   console.log(JSON.stringify(item, null, 2));

   // Show some common reserved words
   console.log("\nSome common DynamoDB reserved words:");
   const commonReservedWords = [
     "ABORT", "ABSOLUTE", "ACTION", "ADD", "ALL", "ALTER", "AND", "ANY", "AS",
     "ASC", "BETWEEN", "BY", "CASE", "CAST", "COLUMN", "CONNECT", "COUNT",
     "CREATE", "CURRENT", "DATE", "DELETE", "DESC", "DROP", "ELSE", "EXISTS",
     "FOR", "FROM", "GRANT", "GROUP", "HAVING", "IN", "INDEX", "INSERT", "INTO",
     "IS", "JOIN", "KEY", "LEVEL", "LIKE", "LIMIT", "LOCAL", "MAX", "MIN",
"NAME",
     "NOT", "NULL", "OF", "ON", "OR", "ORDER", "OUTER", "REPLACE", "RETURN",
```

```
      "SELECT", "SET", "SIZE", "TABLE", "THEN", "TO", "UPDATE", "USER", "VALUES",
      "VIEW", "WHERE"
    ];
    console.log(commonReservedWords.join(", "));

    // Explain expression attribute names
    console.log("\nKey points about expression attribute names:");
    console.log("1. Use expression attribute names (#name) for reserved words");
    console.log("2. Use expression attribute names for attributes with special
 characters");
    console.log("3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
    console.log("4. Expression attribute names are required for nested attributes
 with reserved words");
    console.log("5. You can use multiple expression attribute names in a single
 expression");
    console.log("6. Expression attribute names are case-sensitive");
    console.log("7. Expression attribute names are only used in expressions, not
 in the actual data");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - Query
  - UpdateItem

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3)를 사용한 표현식 속성 이름을 보여줍니다.

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, List


def use_reserved_word_attribute(
```

```
    table_name: str, key: Dict[str, Any], reserved_word: str, value: Any
) -> Dict[str, Any]:
    """
    Update an attribute whose name is a DynamoDB reserved word.

    This function demonstrates how to use expression attribute names to work with
    attributes that have names that are DynamoDB reserved words.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        reserved_word (str): The reserved word to use as an attribute name.
        value (Any): The value to set for the attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names to handle the reserved word
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #reserved_attr = :value",
        ExpressionAttributeNames={"#reserved_attr": reserved_word},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def use_special_character_attribute(
    table_name: str, key: Dict[str, Any], attribute_with_special_chars: str,
 value: Any
) -> Dict[str, Any]:
    """
    Update an attribute whose name contains special characters.

    This function demonstrates how to use expression attribute names to work with
    attributes that have names containing special characters like spaces, dots,
 or hyphens.
```

```
    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        attribute_with_special_chars (str): The attribute name with special
 characters.
        value (Any): The value to set for the attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names to handle special characters
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #special_attr = :value",
        ExpressionAttributeNames={"#special_attr": attribute_with_special_chars},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def query_with_attribute_names(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    filter_attribute_name: str,
    filter_value: Any,
) -> Dict[str, Any]:
    """
    Query a table using expression attribute names for both key and filter
 attributes.

    This function demonstrates how to use expression attribute names in a query
 operation
    for both the key condition expression and filter expression.

    Args:
```

```
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute_name (str): The name of the attribute to filter on.
        filter_value (Any): The value to compare against in the filter.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names for both key condition and filter
    response = table.query(
        KeyConditionExpression="#pk = :pk_val",
        FilterExpression="#filter_attr = :filter_val",
        ExpressionAttributeNames={"#pk": partition_key_name, "#filter_attr":
 filter_attribute_name},
        ExpressionAttributeValues={":pk_val": partition_key_value, ":filter_val":
 filter_value},
    )

    return response


def update_nested_attribute_with_dots(
    table_name: str, key: Dict[str, Any], path_with_dots: str, value: Any
) -> Dict[str, Any]:
    """
    Update a nested attribute using a path with dot notation.

    This function demonstrates how to use expression attribute names to work with
    nested attributes specified using dot notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path_with_dots (str): The path to the nested attribute using dot notation
 (e.g., "a.b.c").
        value (Any): The value to set for the nested attribute.

    Returns:
```

```
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Split the path into components
    path_parts = path_with_dots.split(".")

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        name_placeholder = f"#attr{i}"
        expression_attribute_names[name_placeholder] = part

        if i == 0:
            path_expression = name_placeholder
        else:
            path_expression += f".{name_placeholder}"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def demonstrate_attribute_name_requirements(table_name: str, key: Dict[str, Any])
 -> Dict[str, Any]:
    """
    Demonstrate the requirements and allowed characters for attribute names.
```

```
    This function shows examples of valid and invalid attribute names and how to
    handle them using expression attribute names.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Examples of attribute names with different characteristics
    examples = {
        "valid_standard": "NormalAttribute",  # Standard attribute name (no
placeholder needed)
        "valid_with_underscore": "Normal_Attribute",  # Underscore is allowed
        "valid_with_number": "Attribute123",  # Numbers are allowed
        "reserved_word": "Timestamp",  # Reserved word (requires placeholder)
        "starts_with_number": "123Attribute",  # Starts with number (valid but
may need placeholder in some contexts)
        "with_space": "Attribute Name",  # Contains space (requires placeholder)
        "with_dot": "Attribute.Name",  # Contains dot (requires placeholder)
        "with_hyphen": "Attribute-Name",  # Contains hyphen (requires
placeholder)
        "with_special_chars": "Attribute#$%",  # Contains special characters
(requires placeholder)
    }

    results = {}

    # Try to update each attribute type
    for example_type, attr_name in examples.items():
        try:
            # For attributes that don't need placeholders, try direct reference
            if example_type in ["valid_standard", "valid_with_underscore",
"valid_with_number"]:
                try:
                    # Try without expression attribute names first
                    response = table.update_item(
                        Key=key,
```

```
                        UpdateExpression=f"SET {attr_name} = :value",
                        ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                        ReturnValues="UPDATED_NEW",
                    )
                    results[example_type] = {
                        "attribute_name": attr_name,
                        "success": True,
                        "needed_placeholder": False,
                        "response": response,
                    }
                except ClientError:
                    # If direct reference fails, try with placeholder
                    response = table.update_item(
                        Key=key,
                        UpdateExpression="SET #attr = :value",
                        ExpressionAttributeNames={"#attr": attr_name},
                        ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                        ReturnValues="UPDATED_NEW",
                    )
                    results[example_type] = {
                        "attribute_name": attr_name,
                        "success": True,
                        "needed_placeholder": True,
                        "response": response,
                    }
            else:
                # For attributes that definitely need placeholders
                response = table.update_item(
                    Key=key,
                    UpdateExpression="SET #attr = :value",
                    ExpressionAttributeNames={"#attr": attr_name},
                    ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                    ReturnValues="UPDATED_NEW",
                )
                results[example_type] = {
                    "attribute_name": attr_name,
                    "success": True,
                    "needed_placeholder": True,
                    "response": response,
                }
        except ClientError as e:
```

```
            results[example_type] = {"attribute_name": attr_name, "success":
 False, "error": str(e)}

    return results
```

AWS SDK for Python (Boto3)를 사용한 표현식 속성 이름의 사용 예시입니다.

```
def example_usage():
    """Example of how to use expression attribute names in DynamoDB."""
    # Example parameters
    table_name = "Products"
    key = {"ProductId": "prod123"}

    print("Example 1: Using a reserved word as an attribute name")
    try:
        response = use_reserved_word_attribute(
            table_name=table_name, key=key, reserved_word="Timestamp",
 value="2025-05-14T12:00:00Z"
        )
        print(f"Reserved word attribute updated successfully:
 {response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating reserved word attribute: {e}")

    print("\nExample 2: Using an attribute name with special characters")
    try:
        response = use_special_character_attribute(
            table_name=table_name,
            key=key,
            attribute_with_special_chars="Product Info",
            value="Special product information",
        )
        print(f"Special character attribute updated successfully:
 {response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating special character attribute: {e}")

    print("\nExample 3: Querying with expression attribute names")
    try:
        response = query_with_attribute_names(
```

```
            table_name=table_name,
            partition_key_name="Category",
            partition_key_value="Electronics",
            filter_attribute_name="Price",
            filter_value=500,
        )
        print(
            f"Query with expression attribute names returned
{len(response.get('Items', []))} items"
        )
    except Exception as e:
        print(f"Error querying with expression attribute names: {e}")

    print("\nExample 4: Updating a nested attribute with dot notation")
    try:
        response = update_nested_attribute_with_dots(
            table_name=table_name,
            key=key,
            path_with_dots="Product.Details.Specifications",
            value={"Weight": "2.5 kg", "Dimensions": "30x20x10 cm"},
        )
        print(f"Nested attribute updated successfully:
{response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating nested attribute: {e}")

    print("\nExample 5: Demonstrating attribute name requirements")
    try:
        results = demonstrate_attribute_name_requirements(table_name=table_name,
key=key)

        print("Attribute Name Requirements Results:")
        for example_type, result in results.items():
            if result.get("success", False):
                needed_placeholder = result.get("needed_placeholder", True)
                print(
                    f"  - {example_type}: '{result['attribute_name']}' -
{'Requires' if needed_placeholder else 'Does not require'} placeholder"
                )
            else:
                print(
                    f"  - {example_type}: '{result['attribute_name']}' - Failed:
{result.get('error', 'Unknown error')}"
                )
```

```
    except Exception as e:
        print(f"Error demonstrating attribute name requirements: {e}")

    print("\nCommon DynamoDB Reserved Words (sample):")
    reserved_words = get_common_reserved_words()
    print(", ".join(reserved_words[:20]) + "... (and many more)")

    print("\nWhen to Use Expression Attribute Names:")
    print("1. When the attribute name is a DynamoDB reserved word")
    print("2. When the attribute name contains special characters (spaces, dots,
hyphens)")
    print("3. When the attribute name begins with a number")
    print("4. When working with nested attributes using dot notation")
    print("5. When you need to reference the same attribute multiple times in an
expression")

    print("\nExpression Attribute Name Requirements:")
    print("1. Must begin with a pound sign (#)")
    print("2. After the pound sign, must contain at least one character")
    print("3. Can contain alphanumeric characters and underscore (_)")
    print("4. Are case-sensitive")
    print("5. Must be unique within a single expression")

    print("\nAttribute Name Requirements in DynamoDB:")
    print("1. Can begin with a-z, A-Z, or 0-9")
    print("2. Can contain a-z, A-Z, 0-9, underscore (_), dash (-), and dot (.)")
    print("3. Are case-sensitive")
    print("4. No length restrictions, but practical limits apply")
    print("5. Cannot be a DynamoDB reserved word if used directly in
expressions")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하세요.
  - Query
  - UpdateItem

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

# 예약된 이벤트를 사용하여 Lambda 함수 간접 호출

다음 코드 예제에서는 Amazon EventBridge의 예약된 이벤트에 의해 호출된 AWS Lambda 함수를 생성하는 방법을 보여줍니다.

Java

SDK for Java 2.x

AWS Lambda 함수를 간접 호출하는 Amazon EventBridge 예약된 이벤트를 생성하는 방법을 보여줍니다. Lambda 함수가 간접 호출될 때 cron 표현식을 사용하여 일정을 예약하도록 EventBridge를 구성합니다. 이 예제에서는 Lambda Java 런타임 API를 사용하여 Lambda 함수를 생성합니다. 이 예제에서는 특정 사용 사례를 수행하는 서로 다른 AWS 서비스를 호출합니다. 이 예제에서는 1주년 기념일에 직원에게 축하하는 모바일 문자 메시지를 전송하는 앱을 생성하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예시에서 사용되는 서비스

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

AWS Lambda 함수를 간접 호출하는 Amazon EventBridge 예약된 이벤트를 생성하는 방법을 보여줍니다. Lambda 함수가 간접 호출될 때 cron 표현식을 사용하여 일정을 예약하도록 EventBridge를 구성합니다. 이 예제에서는 Lambda JavaScript 런타임 API를 사용하여 Lambda 함수를 생성합니다. 이 예제에서는 특정 사용 사례를 수행하는 서로 다른 AWS 서비스를 호출합니다. 이 예제에서는 1주년 기념일에 직원에게 축하하는 모바일 문자 메시지를 전송하는 앱을 생성하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예시는 AWS SDK for JavaScript v3 개발자 안내서에서도 확인할 수 있습니다.

이 예시에서 사용되는 서비스

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK for Python (Boto3)

이 예제에서는 AWS Lambda 함수를 예약된 Amazon EventBridge 이벤트의 대상으로 등록하는 방법을 보여줍니다. Lambda 핸들러는 나중에 검색할 수 있도록 알기 쉬운 메시지와 전체 이벤트 데이터를 Amazon CloudWatch Logs에 기록합니다.

- Lambda 함수를 배포합니다.
- EventBridge 예약된 이벤트를 생성하고 Lambda 함수를 대상으로 만듭니다.
- EventBridge에 Lambda 함수를 간접 호출할 수 있는 권한을 부여합니다.
- CloudWatch Logs에서 최신 데이터를 인쇄하여 예약된 호출의 결과를 표시합니다.
- 데모 중에 생성된 모든 리소스를 정리합니다.

이 예제는 GitHub에서 가장 잘 볼 수 있습니다. 전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예시에서 사용되는 서비스

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

# AWS Command Line Interface v2를 사용한 DynamoDB 로컬 보조 인덱스 작업

다음 코드 예제에서는 로컬 보조 인덱스가 있는 테이블을 생성하고 쿼리하는 방법을 보여줍니다.

- 로컬 보조 인덱스(LSI)가 있는 테이블을 생성합니다.
- 다양한 프로젝션 유형의 여러 LSI가 있는 테이블을 생성합니다.
- LSI를 사용하여 데이터를 쿼리합니다.

Bash

Bash 스크립트와 함께 AWS CLI사용

로컬 보조 인덱스가 있는 테이블을 생성합니다.

```
# Create a table with a Local Secondary Index
aws dynamodb create-table \
    --table-name CustomerOrders \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
        AttributeName=OrderID,AttributeType=S \
        AttributeName=OrderDate,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
        AttributeName=OrderID,KeyType=RANGE \
    --local-secondary-indexes \
        "IndexName=OrderDateIndex,\
        KeySchema=[{AttributeName=CustomerID,KeyType=HASH},
{AttributeName=OrderDate,KeyType=RANGE}],\
        Projection={ProjectionType=ALL}" \
    --billing-mode PAY_PER_REQUEST
```

여러 LSI가 있는 테이블을 생성합니다.

```
# Create a table with multiple Local Secondary Indexes
aws dynamodb create-table \
    --table-name CustomerDetails \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
```

```
        AttributeName=Name,AttributeType=S \
        AttributeName=Email,AttributeType=S \
        AttributeName=RegistrationDate,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
        AttributeName=Name,KeyType=RANGE \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"EmailIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"CustomerID\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"Email\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {\"ProjectionType\":\"INCLUDE\",
\"NonKeyAttributes\":[\"Address\",\"Phone\"]}
            },
            {
                \"IndexName\": \"RegistrationIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"CustomerID\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"RegistrationDate\",\"KeyType\":\"RANGE
\"}
                ],
                \"Projection\": {\"ProjectionType\":\"KEYS_ONLY\"}
            }
        ]" \
    --billing-mode PAY_PER_REQUEST
```

LSI를 사용하여 데이터를 쿼리합니다.

```
# Query the OrderDateIndex LSI
aws dynamodb query \
    --table-name CustomerOrders \
    --index-name OrderDateIndex \
    --key-condition-expression "CustomerID = :custId AND OrderDate BETWEEN :date1
 AND :date2" \
    --expression-attribute-values '{
        ":custId": {"S": "C1"},
        ":date1": {"S": "2023-01-01"},
        ":date2": {"S": "2023-02-01"}
    }'
```

```
# Query with a filter expression
aws dynamodb query \
    --table-name CustomerOrders \
    --index-name OrderDateIndex \
    --key-condition-expression "CustomerID = :custId" \
    --filter-expression "Amount > :amount" \
    --expression-attribute-values '{
        ":custId": {"S": "C1"},
        ":amount": {"N": "150"}
    }'
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - CreateTable
  - Query

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS Command Line Interface v2를 사용한 DynamoDB Streams 및 Time-to-Live 작업

다음 코드 예제에서는 DynamoDB Streams 및 Time-to-Live 기능을 관리하는 방법을 보여줍니다.

- Streams를 활성화하여 테이블을 생성합니다.
- Streams를 설명합니다.
- Streams 처리를 위한 Lambda 함수를 생성합니다.
- 테이블에서 TTL을 활성화합니다.
- TTL 속성이 있는 항목을 추가합니다.
- TTL 설정을 설명합니다.

Bash

Bash 스크립트와 함께 AWS CLI사용

Streams를 활성화하여 테이블을 생성합니다.

```
# Create a table with DynamoDB Streams enabled
aws dynamodb create-table \
    --table-name StreamsDemo \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

Streams를 설명합니다.

```
# Get information about the stream
aws dynamodb describe-table \
    --table-name StreamsDemo \
    --query "Table.StreamSpecification"

# Get the stream ARN
STREAM_ARN=$(aws dynamodb describe-table \
    --table-name StreamsDemo \
    --query "Table.LatestStreamArn" \
    --output text)

echo "Stream ARN: $STREAM_ARN"

# Describe the stream
aws dynamodbstreams describe-stream \
    --stream-arn $STREAM_ARN
```

Streams 관련 Lambda 함수를 생성합니다.

```
# Step 1: Create an IAM role for the Lambda function
cat > trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

aws iam create-role \
    --role-name DynamoDBStreamsLambdaRole \
    --assume-role-policy-document file://trust-policy.json

# Step 2: Attach permissions to the role
aws iam attach-role-policy \
    --role-name DynamoDBStreamsLambdaRole \
    --policy-arn arn:aws:iam::aws:policy/service-role/
AWSLambdaDynamoDBExecutionRole

# Step 3: Create a Lambda function (code would be in a separate file)
echo "Lambda function creation would be done separately with appropriate code"

# Step 4: Create an event source mapping
echo "Example command to create event source mapping:"
echo "aws lambda create-event-source-mapping \\"
echo "    --function-name ProcessDynamoDBRecords \\"
echo "    --event-source $STREAM_ARN \\"
echo "    --batch-size 100 \\"
echo "    --starting-position LATEST"
```

테이블에서 TTL을 활성화합니다.

```
# Create a table for TTL demonstration
aws dynamodb create-table \
    --table-name TTLDemo \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST

# Wait for table to become active
aws dynamodb wait table-exists --table-name TTLDemo
```

```
# Enable TTL on the table
aws dynamodb update-time-to-live \
    --table-name TTLDemo \
    --time-to-live-specification "Enabled=true, AttributeName=ExpirationTime"
```

TTL 속성이 있는 항목을 추가합니다.

```
# Calculate expiration time (current time + 1 day in seconds)
EXPIRATION_TIME=$(date -d "+1 day" +%s)

# Add an item with TTL attribute
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item1"},
        "Data": {"S": "This item will expire in 1 day"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME'"}
    }'

# Add an item that expires in 1 hour
EXPIRATION_TIME_HOUR=$(date -d "+1 hour" +%s)
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item2"},
        "Data": {"S": "This item will expire in 1 hour"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME_HOUR'"}
    }'
```

TTL 설정을 설명합니다.

```
# Describe TTL settings for a table
aws dynamodb describe-time-to-live \
    --table-name TTLDemo
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - AttachRolePolicy
  - CreateRole

- CreateTable
- DescribeTable
- DescribeTimeToLive
- PutItem
- UpdateTimeToLive

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS Command Line Interface v2를 사용한 DynamoDB 글로벌 테이블 및 최종 일관성(MREC)을 갖춘 다중 리전 복제 작업

다음 코드 예제에서는 최종 일관성이 있는 다중 리전 복제(MREC)를 사용하여 DynamoDB 글로벌 테이블을 관리하는 방법을 보여줍니다.

- 다중 리전 복제(MREC)를 사용하여 테이블을 만듭니다.
- 복제본 테이블에서 항목을 넣고 가져옵니다.
- 복제본을 하나씩 제거합니다.
- 테이블을 삭제하여 정리합니다.

Bash

Bash 스크립트와 함께 AWS CLI사용

다중 리전 복제를 사용하여 테이블을 생성합니다.

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
```

```
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2
```

다중 리전 테이블을 설명합니다.

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

복제본 테이블에 항목을 넣습니다.

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

복제본 테이블에서 항목을 가져옵니다.

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

복제본을 제거합니다.

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
```

```
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

테이블을 삭제하여 정리합니다.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateTable

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS Command Line Interface v2를 사용한 DynamoDB 리소스 태그 지정 작업

다음 코드 예제에서는 DynamoDB 리소스에 대한 태그를 관리하는 방법을 보여줍니다.

- 태그가 있는 테이블을 생성합니다.

- 리소스에 대한 태그를 나열합니다.
- 리소스에 태그를 추가합니다.
- 리소스에서 태그를 제거합니다.
- 태그를 기준으로 테이블을 필터링합니다.

Bash

Bash 스크립트와 함께 AWS CLI사용

태그가 있는 테이블을 생성합니다.

```
# Create a table with tags
aws dynamodb create-table \
    --table-name TaggedTable \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Environment,Value=Production \
        Key=Project,Value=Analytics \
        Key=Owner,Value=DataTeam
```

리소스에 대한 태그를 나열합니다.

```
# Get the table ARN
TABLE_ARN=$(aws dynamodb describe-table \
    --table-name TaggedTable \
    --query "Table.TableArn" \
    --output text)

# List tags for the table
aws dynamodb list-tags-of-resource \
    --resource-arn $TABLE_ARN
```

리소스에 태그를 추가합니다.

```
# Add tags to an existing table
```

```
aws dynamodb tag-resource \
    --resource-arn $TABLE_ARN \
    --tags \
        Key=CostCenter,Value=12345 \
        Key=BackupSchedule,Value=Daily
```

리소스에서 태그를 제거합니다.

```
# Remove tags from a table
aws dynamodb untag-resource \
    --resource-arn $TABLE_ARN \
    --tag-keys Owner BackupSchedule
```

태그를 기준으로 테이블을 필터링합니다.

```
# Create another table with different tags
aws dynamodb create-table \
    --table-name AnotherTaggedTable \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Environment,Value=Development \
        Key=Project,Value=Testing

# Wait for table to become active
aws dynamodb wait table-exists --table-name AnotherTaggedTable

# List all tables
echo "All tables:"
aws dynamodb list-tables

# Get ARNs for all tables
echo -e "\nFiltering tables by Environment=Production tag:"
TABLE_ARNS=$(aws dynamodb list-tables --query "TableNames[*]" --output text |
 xargs -I {} aws dynamodb describe-table --table-name {} --query "Table.TableArn"
 --output text)

# Find tables with specific tag
```

```
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    TAGS=$(aws dynamodb list-tags-of-resource --resource-arn $ARN --query "Tags[?
Key=='Environment' && Value=='Production']" --output text)
    if [ ! -z "$TAGS" ]; then
        echo "Table with Production tag: $TABLE_NAME"
    fi
done
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - CreateTable
  - ListTagsOfResource
  - TagResource
  - UntagResource

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## AWS Command Line Interface v2를 사용한 DynamoDB 테이블 암호화 작업

다음 코드 예제에서는 DynamoDB 테이블의 암호화 옵션을 관리하는 방법을 보여줍니다.

- 기본 암호화를 사용하여 테이블을 생성합니다.
- 고객 관리형 CMK를 사용하여 테이블을 생성합니다.
- 테이블 암호화 설정을 업데이트합니다.
- 테이블 암호화를 설명합니다.

Bash

Bash 스크립트와 함께 AWS CLI사용

기본 암호화를 사용하여 테이블을 생성합니다.

```
# Create a table with default encryption (AWS owned key)
aws dynamodb create-table \
    --table-name CustomerData \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
```

```
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --sse-specification Enabled=true,SSEType=KMS
```

고객 관리형 CMK를 사용하여 테이블을 생성합니다.

```
# Step 1: Create a customer managed key in KMS
aws kms create-key \
    --description "Key for DynamoDB table encryption" \
    --key-usage ENCRYPT_DECRYPT \
    --customer-master-key-spec SYMMETRIC_DEFAULT

# Store the key ID for later use
KEY_ID=$(aws kms list-keys --query "Keys[?contains(KeyArn, 'Key for
 DynamoDB')].KeyId" --output text)

# Step 2: Create a table with the customer managed key
aws dynamodb create-table \
    --table-name SensitiveData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=$KEY_ID
```

테이블 암호화를 업데이트합니다.

```
# Update a table to use a different KMS key
aws dynamodb update-table \
    --table-name CustomerData \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=$KEY_ID
```

테이블 암호화를 설명합니다.

```
# Describe the table to see encryption settings
aws dynamodb describe-table \
    --table-name CustomerData \
```

```
    --query "Table.SSEDescription"
```

- API 세부 정보는 AWS CLI 명령 참조의 다음 토픽을 참조하세요.
  - CreateKey
  - CreateTable
  - DescribeTable
  - UpdateTable

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

# DynamoDB의 서버리스 예시

다음 코드 예제에서는 DynamoDB를 AWS SDK와 함께 사용하는 방법을 보여줍니다.

예시

- DynamoDB 트리거에서 간접적으로 Lambda 함수 간접 호출
- DynamoDB 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

## DynamoDB 트리거에서 간접적으로 Lambda 함수 간접 호출

다음 코드 예시에서는 DynamoDB 스트림에서 레코드를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 이 함수는 DynamoDB 페이로드를 검색하고 레코드 콘텐츠를 로깅합니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

.NET을 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
 context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Go를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
 error) {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
```

```
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Java를 사용하여 Lambda로 DynamoDB 이벤트 소비

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
 com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
 GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
 GSON.toJson(record.getDynamodb()));
    }
```

```
}
```

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

TypeScript를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
```

```
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

## PHP

### SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

PHP를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
```

```
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Python을 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
import json
```

```
def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Ruby를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
def lambda_handler(event:, context:)
    return 'received empty event' if event['Records'].empty?

    event['Records'].each do |record|
      log_dynamodb_record(record)
    end

    "Records processed: #{event['Records'].length}"
  end

  def log_dynamodb_record(record)
    puts record['eventID']
    puts record['eventName']
    puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
  end
```

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 DynamoDB 이벤트 사용.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");
```

```
    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())

}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

## DynamoDB 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

다음 코드 예시에서는 DynamoDB 스트림에서 이벤트를 수신하는 Lambda 함수에 대한 부분 배치 응답을 구현하는 방법을 보여줍니다. 이 함수는 응답으로 배치 항목 실패를 보고하고 나중에 해당 메시지를 다시 시도하도록 Lambda에 신호를 보냅니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

.NET을 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
```

```
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Go를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)
```

```
type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
 (*BatchResult, error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }

 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Java를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
 input.getRecords()) {
          try {
                //Process your record
                StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
```

```
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse();
    }
}
```

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
 curRecordSequenceNumber }] };
    }
  }
```

```
  return { batchItemFailures: [] };
};
```

TypeScript를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

PHP를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
```

```
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Python을 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
```

```
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Ruby를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```
def lambda_handler(event:, context:)
    records = event["Records"]
    cur_record_sequence_number = ""

    records.each do |record|
      begin
        # Process your record
        cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
        # Return failed record's sequence number
        return {"batchItemFailures" => [{"itemIdentifier" =>
 cur_record_sequence_number}]}
      end
    end

    {"batchItemFailures" => []}
  end
```

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 DynamoDB 배치 항목 실패 보고.

```
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
 Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
```

```
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!("Successfully processed {} record(s)", records.len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

# DynamoDB에 대한 AWS 커뮤니티 기여

AWS 커뮤니티 기여는 AWS의 여러 팀에서 생성하고 유지 관리하는 예입니다. 피드백을 제공하려면 연결된 리포지토리에 제공된 메커니즘을 사용합니다.

예시

- 서버리스 애플리케이션 빌드 및 테스트

## 서버리스 애플리케이션 빌드 및 테스트

다음 코드 예제에서는 Lambda 및 DynamoDB와 함께 API 게이트웨이를 사용하여 서버리스 애플리케이션을 빌드하고 테스트하는 방법을 보여줍니다.

.NET

SDK for .NET

.NET SDK를 사용하여 Lambda 및 DynamoDB가 포함된 API 게이트웨이로 구성된 서버리스 애플리케이션을 빌드하고 테스트하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda

Go

SDK for Go V2

Go SDK를 사용하여 Lambda 및 DynamoDB가 포함된 API 게이트웨이로 구성된 서버리스 애플리케이션을 빌드하고 테스트하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda

Java

SDK for Java 2.x

Java SDK를 사용하여 Lambda 및 DynamoDB가 포함된 API 게이트웨이로 구성된 서버리스 애플리케이션을 빌드하고 테스트하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda

Rust

SDK for Rust

Rust SDK를 사용하여 Lambda 및 DynamoDB가 포함된 API 게이트웨이로 구성된 서버리스 애플리케이션을 빌드하고 테스트하는 방법을 보여줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- API Gateway
- DynamoDB
- Lambda

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 DynamoDB 사용 섹션을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

# Amazon DynamoDB의 보안 및 규정 준수

AWS는 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객으로서 여러분은 가장 높은 보안 요구 사항을 충족하기 위해 설계된 데이터 센터 및 네트워크 아키텍처의 혜택을 받게 됩니다.

보안은 AWS과 사용자의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드의 보안 - AWS는 AWS 클라우드 내에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. DynamoDB에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 제공 범위 내의 AWS 서비스를 참조하세요.
- 클라우드 내 보안 – 사용지의 책임은 사용자가 사용하는 AWS 서비스에 의해 결정됩니다. 또한 데이터의 민감도, 조직의 요건 및 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 DynamoDB 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 항목에서는 보안 및 규정 준수 목표를 충족하도록 DynamoDB를 구성하는 방법을 보여줍니다. 또한 DynamoDB 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법도 알아봅니다.

주제


- Amazon DynamoDB의 AWS 관리형 정책
- DynamoDB에서 리소스 기반 정책 사용
- DynamoDB의 속성 기반 액세스 제어 사용
- DynamoDB의 데이터 보호
- AWS Identity and Access Management(IAM) 및 DynamoDB
- DynamoDB에 대한 업종별 규정 준수 확인
- Amazon DynamoDB의 복원성 및 재해 복구
- Amazon DynamoDB의 인프라 보안
- AWS PrivateLink for DynamoDB
- Amazon DynamoDB의 구성 및 취약성 분석
- Amazon DynamoDB 보안 모범 사례

# Amazon DynamoDB의 AWS 관리형 정책

DynamoDB는 AWS 관리형 정책을 사용하여 서비스가 특정 작업을 수행하는 데 필요한 권한 집합을 정의합니다. DynamoDB는 AWS 관리형 정책을 유지 관리하고 업데이트합니다. AWS 관리형 정책에서는 권한을 변경할 수 없습니다. AWS 관리형 정책에 대한 자세한 정보는 IAM 사용 설명서에서 AWS 관리형 정책을 참조하세요.

DynamoDB는 때때로 추가 권한을 AWS 관리형 정책에 추가하여 새로운 기능을 지원합니다. 이 유형의 업데이트는 정책이 연결된 모든 ID(사용자, 그룹 및 역할)에 적용됩니다. 새로운 기능이 출시되거나 새 작업을 사용할 수 있게 되면 AWS 관리형 정책이 업데이트될 가능성이 높습니다. DynamoDB는 AWS 관리형 정책에서 권한을 제거하지 않기 때문에 정책 업데이트로 인해 기존 권한이 손상되지 않습니다. AWS 관리형 정책의 전체 목록은 AWS managed policies를 참조하세요.

## AWS 관리형 정책: DynamoDBReplicationServiceRolePolicy

`DynamoDBReplicationServiceRolePolicy` 정책을 IAM 엔터티에 연결할 수 없습니다. 이 정책은 DynamoDB가 사용자를 대신하여 작업을 수행할 수 있도록 서비스 연결 역할에 연결됩니다. 자세한 내용은 글로벌 테이블에 IAM 사용을 참조하세요.

이 정책은 서비스 연결 역할이 글로벌 테이블 복제본 간에 데이터 복제를 수행할 수 있도록 하는 권한을 부여합니다. 또한 사용자 대신 글로벌 테이블 복제본을 관리할 수 있는 관리 권한도 부여합니다.

권한 세부 정보

이 정책은 다음을 수행할 수 있는 권한을 부여합니다.

- `dynamodb` - 데이터 복제를 수행하고 테이블 복제본을 관리합니다.
- `application-autoscaling` - 테이블 Auto Scaling 설정을 검색하고 관리합니다.
- `account` - 복제본 접근성을 평가하기 위해 리전 상태를 검색합니다.
- `iam` - 서비스 연결 역할이 아직 없는 경우 애플리케이션 Auto Scaling을 위한 서비스 연결 역할을 생성합니다.

이 관리형 정책의 정의는 여기에서 확인할 수 있습니다.

## AWS 관리형 정책: AmazonDynamoDBFullAccess_v2

범위 축소 `AmazonDynamoDBFullAccess_v2` 정책은 사용자에게 특정 액세스 권한을 부여합니다. `AmazonDynamoDBFullAccess_v2` 정책을 IAM 보안 인증에 연결할 수 있습니다. 이 정책은 Amazon

DynamoDB 리소스에 대한 관리 액세스 권한을 부여하고, 모든 DynamoDB 기능을 사용하기 위해 DynamoDB와 통합되는 AWS 서비스에 대한 액세스 권한을 IAM ID(예: 사용자, 그룹 또는 역할)에 부여합니다. 이 정책을 사용하면 AWS Management Console에서 사용할 수 있는 모든 DynamoDB 기능에 액세스할 수 있습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `Amazon DynamoDB`
- `DynamoDB Accelerator`
- `AWS KMS`
- `AWS Resource Groups Tagging`
- `Lambda`
- `Application Auto Scaling`
- `CloudWatch`
- `Amazon Kinesis`
- `Amazon EC2`
- `IAM`

`JSON` 형식으로 정책을 검토하려면 AmazonDynamoDBFullAccess_v2를 참조하세요.

## AWS 관리형 정책: AmazonDynamoDBReadOnlyAccess

`AmazonDynamoDBReadOnlyAccess` 정책을 IAM 보안 인증에 연결할 수 있습니다.

이 정책은 Amazon DynamoDB에 대한 읽기 전용 액세스 권한을 부여합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `Amazon DynamoDB` - Amazon DynamoDB에 대한 읽기 전용 액세스 권한을 제공합니다.
- `Amazon DynamoDB Accelerator (DAX)` - Amazon DynamoDB Accelerator(DAX)에 대한 읽기 전용 액세스 권한을 제공합니다.
- `Application Auto Scaling` - 보안 주체가 Application Auto Scaling의 구성을 볼 수 있도록 허용합니다. 이는 사용자가 테이블에 첨부된 자동 조정 정책을 볼 수 있도록 하기 위해 필요합니다.

- `CloudWatch` – 보안 주체가 CloudWatch에서 구성된 지표 데이터와 경보를 볼 수 있도록 허용합니다. 이는 사용자가 청구 가능한 테이블 크기 및 테이블에 구성된 CloudWatch 경보를 볼 수 있도록 하기 위해 필요합니다.
- `AWS Data Pipeline` - 보안 주체가 AWS Data Pipeline 및 관련 객체를 보도록 허용합니다.
- `Amazon EC2` - 보안 주체가 Amazon EC2 VPC, 서브넷, 보안 그룹을 보도록 허용합니다.
- `IAM` - 보안 주체가 IAM 역할을 보도록 허용합니다.
- `AWS KMS` - 보안 주체가 AWS KMS에 구성된 키를 볼 수 있도록 허용합니다. 이는 사용자가 자신의 계정에서 생성하고 관리하는 AWS KMS keys를 볼 수 있도록 하기 위해 필요합니다.
- `Amazon SNS` - 보안 주체가 Amazon SNS 주제 및 주제별 구독을 나열하도록 허용합니다.
- `AWS Resource Groups` - 보안 주체가 리소스 그룹 및 해당 쿼리를 보도록 허용합니다.
- `AWS Resource Groups Tagging` - 보안 주체가 한 리전에서 태그가 지정되어 있거나 이전에 태그가 지정되었던 리소스를 모두 나열하도록 허용합니다.
- `Kinesis` - 보안 주체가 Kinesis 데이터 스트림 설명을 보도록 허용합니다.
- `Amazon CloudWatch Contributor Insights` - 보안 주체가 Contributor Insights 규칙이 수집한 시계열 데이터를 보도록 허용합니다.

정책을 `JSON` 형식으로 검토하려면 AmazonDynamoDBReadOnlyAccess를 참조하세요.

## AWS 관리형 정책으로 DynamoDB 업데이트

이 표는 DynamoDB의 AWS 액세스 관리 정책에 대한 업데이트를 보여 줍니다.

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| `AmazonDynamoDBFullAccess` - 사용 중단됨 | 이 정책은 범위가 축소된 `AmazonDynamoDBFullAccess_v2` 라는 정책으로 대체되었습니다.<br><br>2025년 4월 이후에는 새 사용자, 그룹 또는 역할에 `AmazonDynamoDBFullAccess` 정책을 연결할 수 없습니다. 자세한 내용은 AWS 관리형 정책: AmazonDynamoDBFullAccess_v2 섹션을 참조하세요. | 2025년 4월 28일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 기존 정책에 대한 AmazonDynamoDBReadOnlyAccess 업데이트 | AmazonDynamoDBReadOnlyAccess 가 dynamodb:GetAbacStatus 및 dynamodb:UpdateAbacStatus 권한을 추가했습니다. 이러한 권한을 통해 ABAC 상태를 보고 현재 리전의 AWS 계정에 ABAC를 활성화할 수 있습니다. | 2024년 11월 18일 |
| 기존 정책에 대한 AmazonDynamoDBReadOnlyAccess 업데이트 | AmazonDynamoDBReadOnlyAccess 가 dynamodb:GetResourcePolicy 권한을 추가했습니다. 이 권한은 DynamoDB 리소스에 연결된 리소스 기반 정책을 읽을 수 있는 액세스를 제공합니다. | 2024년 3월 20일 |
| 기존 정책에 대한 DynamoDBReplicationServiceRolePolicy 업데이트 | DynamoDBReplicationServiceRolePolicy 가 dynamodb:GetResourcePolicy 권한을 추가했습니다. 이 권한을 사용하면 서비스 연결 역할이 DynamoDB 리소스에 연결된 리소스 기반 정책을 읽을 수 있습니다. | 2023년 12월 15일 |
| 기존 정책에 대한 DynamoDBReplicationServiceRolePolicy 업데이트 | DynamoDBReplicationServiceRolePolicy 가 account:ListRegions 권한을 추가했습니다. 이 권한을 사용하면 서비스 연결 역할이 복제본 접근성을 평가할 수 있습니다. | 2023년 5월 10일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 관리형 정책 목록에 DynamoDBReplicationServiceRolePolicy 추가됨 | DynamoDB 글로벌 테이블 서비스 연결 역할에서 사용하는 관리형 정책 DynamoDBReplicationServiceRolePolicy 에 대한 정보가 추가되었습니다. | 2023년 5월 10일 |
| DynamoDB 글로벌 테이블 변경 사항 추적 시작 | DynamoDB 글로벌 테이블이 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다. | 2023년 5월 10일 |

# DynamoDB에서 리소스 기반 정책 사용

DynamoDB는 테이블, 인덱스 및 스트림에 리소스 기반 정책을 지원합니다. 리소스 기반 정책을 사용하면 각 리소스에 액세스할 수 있는 사람과 각 리소스에서 수행할 수 있는 작업을 지정하여 액세스 권한을 정의할 수 있습니다.

리소스 기반 정책을 테이블 또는 스트림과 같은 DynamoDB 리소스에 연결할 수 있습니다. 이 정책에서는 이러한 DynamoDB 리소스에서 특정 작업을 수행할 수 있는 Identity and Access Management(IAM) 보안 주체에 대한 권한을 지정합니다. 예를 들어, 테이블에 연결된 정책에는 테이블과 해당 인덱스에 대한 액세스 권한이 포함됩니다. 따라서 리소스 기반 정책은 리소스 수준에서 권한을 정의하여 DynamoDB 테이블, 인덱스 및 스트림에 대한 액세스 제어를 간소화하는 데 도움이 될 수 있습니다. DynamoDB 리소스에 연결할 수 있는 정책의 최대 크기는 20KB입니다.

리소스 기반 정책을 사용하면 얻을 수 있는 중요한 이점은 크로스 계정 액세스 제어를 간소화하여 서로 다른 AWS 계정의 IAM 보안 주체에 크로스 계정 액세스 권한을 제공할 수 있다는 것입니다. 자세한 내용은 크로스 게정 액세스에 대한 리소스 기반 정책 섹션을 참조하세요.

또한 리소스 기반 정책은 IAM Access Analyzer 외부 액세스 분석기 및 퍼블릭 액세스 차단(BPA) 기능과의 통합을 지원합니다. IAM Access Analyzer는 리소스 기반 정책에 지정된 외부 엔터티에 대한 크로스 계정 액세스를 보고합니다. 또한 권한을 세분화하고 최소 권한 원칙을 준수하는 데 도움이 되는 가시성을 제공합니다. BPA는 DynamoDB 테이블, 인덱스 및 스트림에 대한 퍼블릭 액세스를 방지하는 데 도움이 되며 리소스 기반 정책 생성 및 수정 워크플로에서 자동으로 활성화됩니다.

주제

- 리소스 기반 정책을 포함한 테이블 생성

- DynamoDB 기존 테이블에 정책 연결
- DynamoDB 스트림에 리소스 기반 정책 연결
- DynamoDB 테이블에서 리소스 기반 정책 제거
- DynamoDB에서 리소스 기반 정책을 사용하는 교차 계정 액세스
- DynamoDB의 리소스 기반 정책으로 퍼블릭 액세스 차단
- 리소스 기반 정책이 지원하는 DynamoDB API 작업
- IAM 자격 증명 기반 정책 및 DynamoDB 리소스 기반 정책을 사용한 권한 부여
- DynamoDB 리소스 기반 정책 예제
- DynamoDB 리소스 기반 정책 고려 사항
- DynamoDB 리소스 기반 정책 모범 사례

## 리소스 기반 정책을 포함한 테이블 생성

DynamoDB 콘솔, CreateTable APIAWS CLI, AWS SDK 또는 AWS CloudFormation 템플릿을 사용하여 테이블을 생성하는 동안 리소스 기반 정책을 추가할 수 있습니다.

AWS CLI

다음 예시에서는 `create-table` AWS CLI 명령을 사용하여 *`MusicCollection`*이라는 테이블을 생성합니다. 이 명령에는 테이블에 리소스 기반 정책을 추가하는 `resource-policy` 파라미터도 포함되어 있습니다. 이 정책을 통해 사용자 *`John`*은 테이블에서 RestoreTableToPointInTime, GetItem 및 PutItem API 작업을 수행할 수 있습니다.

*`####`* 텍스트를 리소스별 정보로 바꿔야 합니다.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --resource-policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
```

```
            {
                  \"Effect\": \"Allow\",
                  \"Principal\": {
                      \"AWS\": \"arn:aws:iam::123456789012:user/John\"
                  },
                  \"Action\": [
                      \"dynamodb:RestoreTableToPointInTime\",
                      \"dynamodb:GetItem\",
                      \"dynamodb:DescribeTable\"
                  ],
                  \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

AWS Management Console

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 대시보드에서 테이블 생성을 선택합니다.
3. 테이블 세부 정보에 테이블 이름, 파티션 키 및 정렬 키 세부 정보를 입력합니다.
4. 테이블 설정에서 설정 사용자 지정을 선택합니다.
5. (선택 사항) 테이블 클래스, 용량 계산기, 읽기/쓰기 용량 설정, 보조 인덱스, 저장 중 암호화 및 삭제 방지에 대한 옵션을 지정합니다.
6. 리소스 기반 정책에서 테이블 및 해당 인덱스에 대한 액세스 권한을 정의하는 정책을 추가합니다. 이 정책에서는 이러한 리소스에 액세스할 수 있는 사용자와 각 리소스에서 해당 사용자가 수행할 수 있는 작업을 지정합니다. 정책을 추가하려면 다음 중 하나를 수행합니다.

   - JSON 정책 문서를 입력하거나 붙여 넣습니다. IAM 정책 언어에 대한 자세한 내용은 IAM 사용 설명서의 JSON 편집기를 사용하여 정책 생성을 참조하세요.

     > **Tip**
     >
     > Amazon DynamoDB 개발자 안내서에서 리소스 기반 정책의 예를 보려면 정책 예시를 선택하세요.

   - 새 문을 추가하려면 새 문 추가를 선택하고 제공된 필드에 정보를 입력합니다. 문을 추가하려는 만큼 이 단계를 반복합니다.

> **Important**
>
> 정책을 저장하기 전에 보안 경고, 오류 및 제안 사항을 해결해야 합니다.

다음 IAM 정책 예시는 사용자 *John*이 *MusicCollection* 테이블에서 RestoreTableToPointInTime, GetItem 및 PutItem API 작업을 수행할 수 있도록 허용합니다.

*####* 텍스트를 리소스별 정보로 바꿔야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/username"
      },
      "Action": [
        "dynamodb:RestoreTableToPointInTime",
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

7. (선택 사항) 오른쪽 하단 외부 액세스 미리 보기를 선택하면 새 정책이 리소스의 퍼블릭 및 크로스 계정 액세스에 미치는 영향을 미리 확인할 수 있습니다. 정책을 저장하기 전에 새로운 IAM Access Analyzer 검색 결과가 나오는지, 기존 결과가 해결되는지 여부를 확인할 수 있습니다. 활성 분석기가 표시되지 않으면 Access Analyzer로 이동을 선택하여 IAM Access Analyzer에서 계정 분석기를 생성합니다. 자세한 내용은 액세스 미리 보기를 참조하세요
8. 테이블 생성을 선택합니다.

AWS CloudFormation 템플릿

Using the AWS::DynamoDB::Table resource

다음 CloudFormation 템플릿은 AWS::DynamoDB::Table 리소스를 사용하여 스트림이 포함된 테이블을 생성합니다. 이 템플릿에는 테이블과 스트림 모두에 연결된 리소스 기반 정책도 포함되어 있습니다.

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "MusicCollectionTable": {
            "Type": "AWS::DynamoDB::Table",
            "Properties": {
                "AttributeDefinitions": [
                    {
                        "AttributeName": "Artist",
                        "AttributeType": "S"
                    }
                ],
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    }
                ],
                "BillingMode": "PROVISIONED",
                "ProvisionedThroughput": {
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "StreamSpecification": {
                  "StreamViewType": "OLD_IMAGE",
                  "ResourcePolicy": {
                    "PolicyDocument": {
                      "Version": "2012-10-17",
                      "Statement": [
                        {
                            "Principal": {
                                "AWS": "arn:aws:iam::111122223333:user/John"
                            },
                            "Effect": "Allow",
                            "Action": [
```

```
                                "dynamodb:GetRecords",
                                "dynamodb:GetShardIterator",
                                "dynamodb:DescribeStream"
                            ],
                            "Resource": "*"
                        }
                    ]
                }
            }
        },
        "TableName": "MusicCollection",
        "ResourcePolicy": {
            "PolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Principal": {
                            "AWS": [
                                "arn:aws:iam::111122223333:user/John"
                            ]
                        },
                        "Effect": "Allow",
                        "Action": "dynamodb:GetItem",
                        "Resource": "*"
                    }
                ]
            }
        }
    }

    }
}
}
```

Using the AWS::DynamoDB::GlobalTable resource

다음 CloudFormation 템플릿은 AWS::DynamoDB::GlobalTable 리소스로 테이블을 생성하고 테이블과 해당 스트림에 리소스 기반 정책을 연결합니다.

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "GlobalMusicCollection": {
```

```
            "Type": "AWS::DynamoDB::GlobalTable",
            "Properties": {
                "TableName": "MusicCollection",
                "AttributeDefinitions": [{
                    "AttributeName": "Artist",
                    "AttributeType": "S"
                }],
                "KeySchema": [{
                    "AttributeName": "Artist",
                    "KeyType": "HASH"
                }],
                "BillingMode": "PAY_PER_REQUEST",
                "StreamSpecification": {
                    "StreamViewType": "NEW_AND_OLD_IMAGES"
                },
                "Replicas": [
                    {
                        "Region": "us-east-1",
                        "ResourcePolicy": {
                            "PolicyDocument": {
                                "Version": "2012-10-17",
                                "Statement": [{
                                    "Principal": {
                                        "AWS": [
                                            "arn:aws:iam::111122223333:user/John"
                                        ]
                                    },
                                    "Effect": "Allow",
                                    "Action": "dynamodb:GetItem",
                                    "Resource": "*"
                                }]
                            }
                        },
                        "ReplicaStreamSpecification": {
                            "ResourcePolicy": {
                                "PolicyDocument": {
                                    "Version": "2012-10-17",
                                    "Statement": [{
                                        "Principal": {
                                            "AWS":
"arn:aws:iam::111122223333:user/John"
                                        },
                                        "Effect": "Allow",
                                        "Action": [
```

```
                                        "dynamodb:GetRecords",
                                        "dynamodb:GetShardIterator",
                                        "dynamodb:DescribeStream"
                                    ],
                                    "Resource": "*"
                                }]
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

## DynamoDB 기존 테이블에 정책 연결

DynamoDB 콘솔, PutResourcePolicy API, AWS CLI, AWS SDK 또는 AWS CloudFormation 템플릿을 사용하여 기존 테이블에 리소스 기반 정책을 연결하거나 기존 정책을 수정할 수 있습니다.

새 정책을 연결하는 AWS CLI 예시

다음 IAM 정책 예시에서는 `put-resource-policy` AWS CLI 명령을 사용하여 기존 테이블에 리소스 기반 정책을 연결합니다. 이 예시를 사용하면 사용자 *John*이 *MusicCollection*이라는 기존 테이블에서 GetItem, PutItem, UpdateItem, UpdateTable API 작업을 수행할 수 있습니다.

*####* 텍스트를 리소스별 정보로 바꿔야 합니다.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetItem\",
```

```
                        \"dynamodb:PutItem\",
                        \"dynamodb:UpdateItem\",
                        \"dynamodb:UpdateTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

기존 정책을 조건부로 업데이트하는 AWS CLI 예시

테이블의 기존 리소스 기반 정책을 조건부로 업데이트하려면 선택적인 `expected-revision-id` 파라미터를 사용할 수 있습니다. 다음 예시는 정책이 DynamoDB에 존재하고 현재 수정 ID가 제공된 `expected-revision-id` 파라미터와 일치하는 경우에만 정책을 업데이트합니다.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --expected-revision-id 1709841168699 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetItem\",
                        \"dynamodb:UpdateItem\",
                        \"dynamodb:UpdateTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

AWS Management Console

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.

2. 대시보드에서 기존 테이블을 선택합니다.
3. 권한 탭으로 이동한 다음 테이블 정책 생성을 선택합니다.
4. 리소스 기반 정책 편집기에서 연결하려는 정책을 추가하고 정책 생성을 선택합니다.

   다음 IAM 정책 예시를 사용하면 사용자 *John*이 *MusicCollection*이라는 기존 테이블에서 GetItem, PutItem, UpdateItem, UpdateTable API 작업을 수행할 수 있습니다.

   #### 텍스트를 리소스별 정보로 바꿔야 합니다.

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:UpdateTable"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

AWS SDK for Java 2.x

다음 IAM 정책 예시에서는 putResourcePolicy 메서드를 사용하여 기존 테이블에 리소스 기반 정책을 연결합니다. 이 정책을 통해 사용자는 기존 테이블에서 GetItem API 작업을 수행할 수 있습니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutResourcePolicyRequest;
```

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * Get started with the AWS SDK for Java 2.x
 */
public class PutResourcePolicy {

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableArn> <allowedAWSPrincipal>

                Where:
                    tableArn - The Amazon DynamoDB table ARN to attach the policy to.
 For example, arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection.
                    allowedAWSPrincipal - Allowed AWS principal ARN that the example
 policy will give access to. For example, arn:aws:iam::123456789012:user/John.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableArn = args[0];
        String allowedAWSPrincipal = args[1];
        System.out.println("Attaching a resource-based policy to the Amazon DynamoDB
 table with ARN " +
                tableArn);
        Region region = Region.US_WEST_2;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        String result = putResourcePolicy(ddb, tableArn, allowedAWSPrincipal);
        System.out.println("Revision ID for the attached policy is " + result);
        ddb.close();
    }
```

```
    public static String putResourcePolicy(DynamoDbClient ddb, String tableArn, String
 allowedAWSPrincipal) {
        String policy = generatePolicy(tableArn, allowedAWSPrincipal);
        PutResourcePolicyRequest request = PutResourcePolicyRequest.builder()
                .policy(policy)
                .resourceArn(tableArn)
                .build();

        try {
            return ddb.putResourcePolicy(request).revisionId();
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

        return "";
    }

    private static String generatePolicy(String tableArn, String allowedAWSPrincipal) {
        return "{\n" +
                "    \"Version\": \"2012-10-17\",\n" +
                "    \"Statement\": [\n" +
                "        {\n" +
                "            \"Effect\": \"Allow\",\n" +
                "            \"Principal\": {\"AWS\":\"" + allowedAWSPrincipal + "\"},
\n" +
                "            \"Action\": [\n" +
                "                \"dynamodb:GetItem\"\n" +
                "            ],\n" +
                "            \"Resource\": \"" + tableArn + "\"\n" +
                "        }\n" +
                "    ]\n" +
                "}";
    }
}
```

## DynamoDB 스트림에 리소스 기반 정책 연결

DynamoDB 콘솔, PutResourcePolicy API, AWS CLI, AWS SDK 또는 AWS CloudFormation 템플릿을 사용하여 기존 테이블의 스트림에 리소스 기반 정책을 연결하거나 기존 정책을 수정할 수 있습니다.

> **Note**
>
> CreateTable 또는 UpdateTable API를 사용하여 스트림을 생성하는 동안에는 정책을 스트림에 연결할 수 없습니다. 하지만 테이블을 삭제한 후에는 정책을 수정하거나 삭제할 수 있습니다. 비활성화된 스트림의 정책을 수정하거나 삭제할 수도 있습니다.

AWS CLI

다음 IAM 정책 예시에서는 `put-resource-policy` AWS CLI 명령을 사용하여 *MusicCollection*이라는 테이블의 스트림에 리소스 기반 정책을 연결합니다. 이 예시를 통해 사용자 *John*은 스트림에서 GetRecords, GetShardIterator, DescribeStream API 작업을 수행할 수 있습니다.

*####* 텍스트를 리소스별 정보로 바꿔야 합니다.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetRecords\",
                        \"dynamodb:GetShardIterator\",
                        \"dynamodb:DescribeStream\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection/stream/2024-02-12T18:57:26.492\"
                }
            ]
        }"
```

AWS Management Console

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. DynamoDB 콘솔 대시보드에서 테이블을 선택한 다음, 기존 테이블을 선택합니다.

   선택한 테이블에 스트림이 켜져 있는지 확인하세요. 테이블의 스트림 활성화에 대한 정보는 스트림 활성화 섹션을 참조하세요.
3. 권한 탭을 선택합니다.
4. 활성 스트림용 리소스 기반 정책에서 스트림 정책 생성을 선택합니다.
5. 리소스 기반 정책 편집기에서 스트림에 대한 액세스 권한을 정의하는 정책을 추가합니다. 이 정책에서는 해당 스트림에 액세스할 수 있는 사용자와 스트림에서 해당 사용자가 수행할 수 있는 작업을 지정합니다. 정책을 추가하려면 다음 중 하나를 수행합니다.

   - JSON 정책 문서를 입력하거나 붙여 넣습니다. IAM 정책 언어에 대한 자세한 내용은 IAM 사용 설명서의 JSON 편집기를 사용하여 정책 생성을 참조하세요.

     > **Tip**
     >
     > Amazon DynamoDB 개발자 안내서에서 리소스 기반 정책의 예를 보려면 정책 예시를 선택하세요.

   - 새 문을 추가하려면 새 문 추가를 선택하고 제공된 필드에 정보를 입력합니다. 문을 추가하려는 만큼 이 단계를 반복합니다.

     > **Important**
     >
     > 정책을 저장하기 전에 보안 경고, 오류 및 제안 사항을 해결해야 합니다.
6. (선택 사항) 오른쪽 하단 외부 액세스 미리 보기를 선택하면 새 정책이 리소스의 퍼블릭 및 크로스 계정 액세스에 미치는 영향을 미리 확인할 수 있습니다. 정책을 저장하기 전에 새로운 IAM Access Analyzer 검색 결과가 나오는지, 기존 결과가 해결되는지 여부를 확인할 수 있습니다. 활성 분석기가 표시되지 않으면 Access Analyzer로 이동을 선택하여 IAM Access Analyzer에서 계정 분석기를 생성합니다. 자세한 내용은 액세스 미리 보기를 참조하세요
7. 정책 생성을 선택합니다.

다음 IAM 정책 예시에서는 리소스 기반 정책을 *MusicCollection*이라는 테이블의 스트림에 연결합니다. 이 예시를 통해 사용자 *John*은 스트림에서 GetRecords, GetShardIterator, DescribeStream API 작업을 수행할 수 있습니다.

*####* 텍스트를 리소스별 정보로 바꿔야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:DescribeStream"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

## DynamoDB 테이블에서 리소스 기반 정책 제거

DynamoDB 콘솔, DeleteResourcePolicy API, AWS CLI, AWS SDK 또는 AWS CloudFormation 템플릿을 사용하여 기존 테이블에서 리소스 기반 정책을 삭제할 수 있습니다.

AWS CLI

다음 예시에서는 `delete-resource-policy` AWS CLI 명령을 사용하여 *MusicCollection*이라는 테이블에서 리소스 기반 정책을 제거합니다.

*####* 텍스트를 리소스별 정보로 바꿔야 합니다.

```
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

AWS Management Console

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. DynamoDB 콘솔 대시보드에서 테이블을 선택한 다음, 기존 테이블을 선택합니다.
3. 권한을 선택합니다.
4. 정책 관리 드롭다운에서 정책 삭제를 선택합니다.
5. 테이블에 대한 리소스 기반 정책 삭제 대화 상자에서 **confirm**을 입력하여 삭제 작업을 확인합니다.
6. 삭제를 선택합니다.

## DynamoDB에서 리소스 기반 정책을 사용하는 교차 계정 액세스

리소스 기반 정책을 사용하여 서로 다른 AWS 계정에서 사용 가능한 리소스에 대한 크로스 계정 액세스 권한을 제공할 수 있습니다. 리소스 기반 정책에서 허용하는 모든 크로스 계정 액세스는 리소스와 동일한 AWS 리전에 분석기가 있는 경우 IAM Access Analyzer 외부 액세스 조사 결과를 통해 보고됩니다. IAM Access Analyzer는 정책 확인은 실행하여 IAM 정책 문법 및 모범 사례에 대해 정책을 검증합니다. 이러한 확인은 결과를 생성하고 보안 모범 사례를 준수하고 작동하는 정책을 작성하는 데 도움이 되는 실행 가능한 권장 사항을 제공합니다. DynamoDB 콘솔의 권한 탭에서 IAM Access Analyzer의 활성 조사 결과를 볼 수 있습니다.

IAM Access Analyzer를 사용한 정책 검증에 대한 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer 정책 검증을 참조하세요. IAM Access Analyzer에서 반환된 경고, 오류 및 제안 사항 목록을 보려면 IAM Access Analyzer 정책 확인 참조를 참조하십시오.

계정 A의 사용자 A에게 계정 B의 테이블 B에 액세스할 수 있는 GetItem 권한을 부여하려면 다음 단계를 수행하세요.

1. 사용자 A에게 `GetItem` 작업 수행 권한을 부여하는 리소스 기반 정책을 테이블 B에 연결합니다.
2. 사용자 A에게 테이블 B에 대한 `GetItem` 작업 수행 권한을 부여하는 자격 증명 기반 정책을 연결합니다.

DynamoDB 콘솔에서 제공되는 외부 액세스 미리 보기 옵션을 사용하여 새 정책이 리소스에 대한 퍼블릭 및 크로스 계정 액세스에 미치는 영향을 미리 볼 수 있습니다. 정책을 저장하기 전에 새로운 IAM Access Analyzer 검색 결과가 나오는지, 기존 결과가 해결되는지 여부를 확인할 수 있습니다. 활성 분석기가 표시되지 않으면 Access Analyzer로 이동을 선택하여 IAM Access Analyzer에서 계정 분석기를 생성합니다. 자세한 내용은 액세스 미리 보기를 참조하세요

DynamoDB 데이터 영역 및 컨트롤 플레인 API의 테이블 이름 파라미터는 테이블의 완전한 Amazon 리소스 이름(ARN)을 수락하여 크로스 계정 작업을 지원합니다. 전체 ARN 대신 테이블 이름 파라미터만 제공하는 경우 요청자가 속한 계정의 테이블에서 API 작업이 수행됩니다. 크로스 계정 액세스 권한을 사용하는 정책 예시는 크로스 계정 액세스에 대한 리소스 기반 정책 섹션을 참조하세요.

다른 계정의 보안 주체가 소유자 계정의 DynamoDB 테이블을 읽거나 해당 테이블에 데이터를 쓰는 경우에도 리소스 소유자 계정에 요금이 부과됩니다. 테이블에 프로비저닝된 처리량이 있는 경우 소유자 계정과 다른 계정의 요청자로부터 받은 모든 요청의 합계에 따라 요청의 제한(Auto Scaling이 비활성화된 경우) 또는 스케일 업/다운(Auto Scaling이 활성화된 경우) 여부가 결정됩니다.

요청은 소유자 및 요청자 계정 모두의 CloudTrail 로그에 로깅되므로 두 계정 각각은 어떤 계정이 어떤 데이터에 액세스했는지 추적할 수 있습니다.

> **Note**
>
> 컨트롤 플레인 API의 크로스 계정 액세스는 초당 트랜잭션 수(TPS) 한도가 500개로 더 낮습니다.

## DynamoDB의 리소스 기반 정책으로 퍼블릭 액세스 차단

퍼블릭 액세스 차단(BPA)은 Amazon Web Services(AWS) 계정의 DynamoDB 테이블, 인덱스 또는 스트림에 대한 퍼블릭 액세스를 허용하는 리소스 기반 정책의 연결을 식별하고 이를 방지하는 기능입니다. BPA를 사용하면 DynamoDB 리소스에 대한 퍼블릭 액세스를 차단할 수 있습니다. BPA는 리소스 기반 정책을 만들거나 수정하는 동안 검사를 수행하며 DynamoDB의 보안 태세를 개선하는 데 도움이 됩니다.

BPA는 자동 추론을 사용하여 리소스 기반 정책에 의해 부여된 액세스를 분석하고 리소스 기반 정책을 관리할 때 해당 권한이 발견되면 알려줍니다. 분석을 통해 정책에 사용되는 모든 리소스 기반 정책 문, 작업 및 조건 키 집합에 대한 액세스를 확인합니다.

> **Important**
>
> BPA는 테이블, 인덱스, 스트림과 같은 DynamoDB 리소스에 직접 연결된 리소스 기반 정책을 통해 퍼블릭 액세스가 허용되지 않도록 함으로써 리소스를 보호합니다. BPA를 사용하는 것 외에도 다음 정책을 주의 깊게 검토하여 퍼블릭 액세스를 허용하지 않는지 확인하세요.
>
> - 관련 AWS 보안 주체(예: IAM 역할)에 연결된 자격 증명 기반 정책
> - 관련 AWS 리소스(예: AWS Key Management Service(KMS) 키)에 연결된 리소스 기반 정책

보안 주체에 * 항목이 포함되지 않도록 하거나 지정된 조건 키 중 하나가 보안 주체의 리소스 액세스를 제한하는지 확인해야 합니다. AWS 계정 전반에서 리소스 기반 정책이 테이블, 인덱스 또는 스트림에 대한 퍼블릭 액세스를 허용하는 경우 DynamoDB는 정책 내의 사양이 수정되어 퍼블릭이 아닌 것으로 간주될 때까지 정책을 생성하거나 수정하지 못하도록 차단합니다.

`Principal` 블록 내에 하나 이상의 보안 주체를 지정하여 정책을 퍼블릭이 아닌 상태로 만들 수 있습니다. 다음 리소스 기반 정책 예시는 두 개의 보안 주체를 지정하여 퍼블릭 액세스를 차단합니다.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dynamodb:*",
  "Resource": "*"
}
```

특정 조건 키를 지정하여 액세스를 제한하는 정책도 퍼블릭으로 간주되지 않습니다. 리소스 기반 정책에 지정된 보안 주체 평가와 함께 다음과 같은 신뢰할 수 있는 조건 키를 사용하여 퍼블릭이 아닌 액세스에 대한 리소스 기반 정책의 평가를 완료합니다.

- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipalOrgID`
- `aws:PrincipalOrgPaths`

- aws:SourceAccount
- aws:SourceArn
- aws:SourceVpc
- aws:SourceVpce
- aws:UserId
- aws:PrincipalServiceName
- aws:PrincipalServiceNamesList
- aws:PrincipalIsAWSService
- aws:Ec2InstanceSourceVpc
- aws:SourceOrgID
- aws:SourceOrgPaths

또한 리소스 기반 정책을 퍼블릭이 아닌 상태로 설정하려면 Amazon 리소스 이름(ARN) 및 문자열 키의 값에 와일드카드나 변수가 포함되어서는 안 됩니다. 리소스 기반 정책에서 aws:PrincipalIsAWSService 키를 사용하는 경우 키 값을 true로 설정했는지 확인해야 합니다.

다음 정책은 지정된 계정의 John 사용자에 액세스 권한을 제한합니다. 이 조건에 따라 Principal이 제한되며 퍼블릭으로 간주되지 않습니다.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dynamodb:*",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/John"
    }
  }
}
```

다음은 StringEquals 연산자를 사용하는 퍼블릭이 아닌 리소스 기반 정책 제약 sourceVPC의 예입니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dynamodb:*",
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/
MusicCollection",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

## 리소스 기반 정책이 지원하는 DynamoDB API 작업

이 주제에서는 리소스 기반 정책이 지원하는 API 작업을 설명합니다. 그러나 크로스 계정 액세스의 경우 리소스 기반 정책을 통해 특정 DynamoDB API 집합만 사용할 수 있습니다. 리소스 기반 정책을 리소스 유형(예: 백업 및 가져오기)에 연결할 수 없습니다. 이러한 리소스 유형에서 작동하는 API와 부합하는 IAM 작업은 리소스 기반 정책에서 지원되는 IAM 작업에서 제외됩니다. 테이블 관리자가 동일한 계정 내에서 내부 테이블 설정을 구성하기 때문에 UpdateTimeToLive 및 DisableKinesisStreamingDestination과 같은 API는 리소스 기반 정책을 통한 크로스 계정 액세스를 지원하지 않습니다.

크로스 계정 액세스를 지원하는 DynamoDB 데이터 영역 및 컨트롤 플레인 API는 테이블 이름 오버로딩도 지원하므로 테이블 이름 대신 테이블 ARN을 지정할 수 있습니다. 이러한 API의 `TableName` 파라미터에 테이블 ARN을 지정할 수 있습니다. 하지만 이러한 API가 모두 크로스 계정 액세스를 지원하는 것은 아닙니다.

주제

- 데이터 영역 API 작업
- PartiQL API 작업
- 컨트롤 플레인 API 작업
- 버전 2019.11.21(현재) 글로벌 테이블 API 작업
- 버전 2017.11.29(레거시) 글로벌 테이블 API 작업
- 태그 API 작업
- 백업 및 복원 API 작업
- 연속 백업/복원(PITR) API 작업
- Contributor Insights API 작업
- API 내보내기 작업
- API 가져오기 작업
- Amazon Kinesis Data Streams API 작업
- 리소스 기반 정책 API 작업
- TTL API 작업
- 기타 API 작업
- DynamoDB Streams API 작업

## 데이터 영역 API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 데이터 영역 API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| 데이터 영역 - 테이블/인덱스 API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
|---|---|---|
| DeleteItem | 예 | 예 |
| GetItem | 예 | 예 |
| PutItem | 예 | 예 |
| Query | 예 | 예 |
| Scan | 예 | 예 |

| 데이터 영역 - 테이블/인덱스 API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
|---|---|---|
| UpdateItem | 예 | 예 |
| TransactGetItems | 예 | 예 |
| TransactWriteItems | 예 | 예 |
| BatchGetItem | 예 | 예 |
| BatchWriteItem | 예 | 예 |

## PartiQL API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 PartiQL API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| PartiQL API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
|---|---|---|
| BatchExecuteStatement | 예 | 아니요 |
| ExecuteStatement | 예 | 아니요 |
| ExecuteTransaction | 예 | 아니요 |

## 컨트롤 플레인 API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 컨트롤 플레인 API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| 컨트롤 플레인 - 테이블 API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
|---|---|---|
| CreateTable | 아니요 | 아니요 |
| DeleteTable | 예 | 예 |
| DescribeTable | 예 | 예 |

| 컨트롤 플레인 - 테이블 API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
| --- | --- | --- |
| UpdateTable | 예 | 예 |

## 버전 2019.11.21(현재) 글로벌 테이블 API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 버전 2019.11.21(현재) 글로벌 테이블 API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| 버전 2019.11.21(현재) 글로벌 테이블 API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
| --- | --- | --- |
| DescribeTableReplicaAutoScaling | 예 | 아니요 |
| UpdateTableReplicaAutoScaling | 예 | 아니요 |

## 버전 2017.11.29(레거시) 글로벌 테이블 API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 버전 2017.11.29(레거시) 글로벌 테이블 API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| 버전 2017.11.29(레거시) 글로벌 테이블 API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
| --- | --- | --- |
| CreateGlobalTable | 아니요 | 아니요 |
| DescribeGlobalTable | 아니요 | 아니요 |
| DescribeGlobalTableSettings | 아니요 | 아니요 |
| ListGlobalTables | 아니요 | 아니요 |
| UpdateGlobalTable | 아니요 | 아니요 |
| UpdateGlobalTableSettings | 아니요 | 아니요 |

## 태그 API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 태그와 관련하여 API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| 태그 API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
|---|---|---|
| ListTagsOfResource | 예 | 예 |
| TagResource | 예 | 예 |
| UntagResource | 예 | 예 |

## 백업 및 복원 API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 백업 및 복원과 관련하여 API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| 백업 및 복원 API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
|---|---|---|
| CreateBackup | 예 | 아니요 |
| DescribeBackup | 아니요 | 아니요 |
| DeleteBackup | 아니요 | 아니요 |
| RestoreTableFromBackup | 아니요 | 아니요 |

## 연속 백업/복원(PITR) API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 연속 백업/복원(PITR)과 관련하여 API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| 연속 백업/복원(PITR) API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
|---|---|---|
| DescribeContinuousBackups | 예 | 아니요 |
| RestoreTableToPointInTime | 예 | 아니요 |

| 연속 백업/복원(PITR) API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
| --- | --- | --- |
| UpdateContinuousBackups | 예 | 아니요 |

## Contributor Insights API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 연속 백업/복원(PITR)과 관련하여 API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| Contributor Insights API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
| --- | --- | --- |
| DescribeContributorInsights | 예 | 아니요 |
| ListContributorInsights | 아니요 | 아니요 |
| UpdateContributorInsights | 예 | 아니요 |

## API 내보내기 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 API 내보내기 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| API 내보내기 | 리소스 기반 정책 지원 | 크로스 계정 지원 |
| --- | --- | --- |
| DescribeExport | 아니요 | 아니요 |
| ExportTableToPointInTime | 예 | 아니요 |
| ListExports | 아니요 | 아니요 |

## API 가져오기 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 API 가져오기 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| API 가져오기 | 리소스 기반 정책 지원 | 크로스 계정 지원 |
|---|---|---|
| DescribeImport | 아니요 | 아니요 |
| ImportTable | 아니요 | 아니요 |
| ListImports | 아니요 | 아니요 |

## Amazon Kinesis Data Streams API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 Kinesis Data Streams API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| Kinesis API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
|---|---|---|
| DescribeKinesisStreamingDestination | 예 | 아니요 |
| DisableKinesisStreamingDestination | 예 | 아니요 |
| EnableKinesisStreamingDestination | 예 | 아니요 |
| UpdateKinesisStreamingDestination | 예 | 아니요 |

## 리소스 기반 정책 API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 리소스 기반 정책 API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| 리소스 기반 정책 API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
|---|---|---|
| GetResourcePolicy | 예 | 아니요 |
| PutResourcePolicy | 예 | 아니요 |

| 리소스 기반 정책 API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
| --- | --- | --- |
| DeleteResourcePolicy | 예 | 아니요 |

## TTL API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 TTL(Time-to-Live) API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| TTL API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
| --- | --- | --- |
| DescribeTimeToLive | 예 | 아니요 |
| UpdateTimeToLive | 예 | 아니요 |

## 기타 API 작업

다음 표에는 리소스 기반 정책 및 크로스 계정 액세스에 대해 기타 API 작업에서 제공하는 API 수준 지원이 나와 있습니다.

| 기타 API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
| --- | --- | --- |
| DescribeLimits | 아니요 | 아니요 |
| DescribeEndpoints | 아니요 | 아니요 |
| ListBackups | 아니요 | 아니요 |
| ListTables | 아니요 | 아니요 |

## DynamoDB Streams API 작업

다음 테이블에는 리소스 기반 정책 및 크로스 계정 액세스에 대한 DynamoDB Streams API의 API 수준 지원이 나열되어 있습니다.

| DynamoDB Streams API | 리소스 기반 정책 지원 | 크로스 계정 지원 |
|---|---|---|
| DescribeStream | 예 | 예 |
| GetRecords | 예 | 예 |
| GetShardIterator | 예 | 예 |
| ListStreams | 아니요 | 아니요 |

## IAM 자격 증명 기반 정책 및 DynamoDB 리소스 기반 정책을 사용한 권한 부여

자격 증명 기반 정책은 IAM 사용자, 사용자 그룹, 역할과 같은 자격 증명에 연결됩니다. 이는 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어하는 IAM 정책 문서입니다. 자격 증명 기반 정책은 관리형 정책 또는 인라인 정책이 될 수 있습니다.

리소스 기반 정책은 DynamoDB 테이블과 같은 리소스에 연결하는 IAM 정책 문서입니다. 이러한 정책은 지정된 보안 주체에 해당 리소스에 대한 특정 작업을 수행할 수 있는 권한을 부여하고 이러한 권한이 적용되는 조건을 정의합니다. 예를 들어, DynamoDB 테이블의 리소스 기반 정책에는 테이블과 연결된 인덱스도 포함됩니다. 리소스 기반 정책은 인라인 정책입니다. 관리형 리소스 기반 정책은 없습니다.

이러한 정책의 차이점에 대한 자세한 내용은 IAM 사용 설명서의 자격 증명 기반 정책 및 리소스 기반 정책을 참조하세요.

IAM 보안 주체가 리소스 소유자와 동일한 계정의 사용자인 경우 리소스 기반 정책만으로도 리소스에 대한 액세스 권한을 지정하기에 충분합니다. 여전히 리소스 기반 정책과 함께 IAM 자격 증명 기반 정책을 사용할 수도 있습니다. 크로스 계정 액세스의 경우 DynamoDB에서 리소스 기반 정책을 사용하는 교차 계정 액세스에 지정된 대로 자격 증명 및 리소스 정책 모두에서 액세스를 명시적으로 허용해야 합니다. 두 정책 유형을 모두 사용하는 경우 정책은 계정 내에서 요청 허용 여부 결정에 설명된 대로 평가됩니다.

## DynamoDB 리소스 기반 정책 예제

리소스 기반 정책의 `Resource` 필드에 ARN을 지정하는 경우, 지정된 ARN이 연결된 DynamoDB 리소스의 ARN과 일치하는 경우에만 정책이 적용됩니다.

> **Note**
>
> *####* 텍스트를 리소스별 정보로 바꿔야 합니다.

## 테이블에 대한 리소스 기반 정책

*MusicCollection*이라는 DynamoDB 테이블에 연결된 다음 리소스 기반 정책은 IAM 사용자 *John*과 *Jane*에게 *MusicCollection* 리소스에서 GetItem 및 BatchGetItem 작업을 수행할 수 있는 권한을 부여합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
        "Effect": "Allow",
        "Principal": {
          "AWS": [
            "arn:aws:iam::111122223333:user/username",
            "arn:aws:iam::111122223333:user/Jane"
          ]
        },
        "Action": [
          "dynamodb:GetItem",
          "dynamodb:BatchGetItem"
        ],
        "Resource": [
          "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
        ]
    }
  ]
}
```

## 스트림에 대한 리소스 기반 정책

2024-02-12T18:57:26.492라는 DynamoDB 스트림에 연결된 다음 리소스 기반 정책은 IAM 사용자 *John*과 *Jane*에게 2024-02-12T18:57:26.492 리소스에서 GetRecords, GetShardIterator, DescribeStream API 작업을 수행할 수 있는 권한을 부여합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:user/username",
          "arn:aws:iam::111122223333:user/Jane"
        ]
      },
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

## 지정된 리소스에서 모든 작업을 수행할 수 있는 액세스에 대한 리소스 기반 정책

사용자가 테이블 및 테이블과 관련된 모든 인덱스에 대한 모든 작업을 수행할 수 있도록 하려면 와일드카드(*)를 사용하여 테이블과 관련된 작업 및 리소스를 나타낼 수 있습니다. 리소스에 와일드카드 문자를 사용하면 사용자가 DynamoDB 테이블 및 아직 생성되지 않은 인덱스를 비롯한 모든 관련 인덱스에

액세스할 수 있습니다. 예를 들어, 다음 정책은 사용자 *John*에게 *MusicCollection* 테이블 및 향후 생성될 인덱스를 포함한 모든 인덱스에서 모든 작업을 수행할 수 있는 권한을 부여합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/role-name"
      },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
index/index-name"
      ]
    }
  ]
}
```

## 크로스 계정 액세스에 대한 리소스 기반 정책

크로스 계정 IAM 자격 증명에 DynamoDB 리소스에 대한 액세스 권한을 지정할 수 있습니다. 예를 들어 신뢰할 수 있는 계정의 사용자가 테이블의 콘텐츠를 읽는 액세스 권한을 얻어야 할 수 있습니다. 단, 해당 사용자는 해당 항목의 특정 항목과 특정 속성에만 액세스해야 합니다. 다음 정책은 신뢰할 수 있는 AWS 계정 ID *111111111111*에서 사용자 *John*이 GetItem API를 사용하여 계정 *123456789012*에 있는 테이블의 데이터에 액세스할 수 있도록 허용합니다. 이 정책은 사용자가 프라이머리 키 *Jane*이 있는 항목에만 액세스할 수 있으며 속성 `Artist` 및 `SongTitle`만 검색할 수 있도록 하고 다른 속성은 검색할 수 없도록 합니다.

> **Important**
>
> `SPECIFIC_ATTRIBUTES` 조건을 지정하지 않으면 반환된 항목의 모든 속성을 볼 수 있습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountTablePolicy",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:user/John"
            },
            "Action": "dynamodb:GetItem",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
}
```

이전 리소스 기반 정책 외에도 사용자 *John*에게 연결된 자격 증명 기반 정책도 `GetItem` API 작업을 허용해야 크로스 계정 액세스가 가능합니다. 다음은 사용자 *John*에게 연결해야 하는 자격 증명 기반 정책 예시입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "CrossAccountIdentityBasedPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
}
```

사용자 John은 `table-name` 파라미터에 테이블 ARN을 지정하여 계정 *123456789012*의 *MusicCollection* 테이블에 액세스하기 위한 `GetItem` 요청을 수행할 수 있습니다.

```
aws dynamodb get-item \
    --table-name arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --key '{"Artist": {"S": "Jane"}' \
    --projection-expression 'Artist, SongTitle' \
    --return-consumed-capacity TOTAL
```

## IP 주소 조건이 포함된 리소스 기반 정책

조건을 적용하여 소스 IP 주소, Virtual Private Cloud(VPC) 및 VPC 엔드포인트(VPCE)를 제한할 수 있습니다. 원래 요청의 소스 주소를 기반으로 권한을 지정할 수 있습니다. 예를 들어, 사용자가 기업 VPN 엔드포인트와 같은 특정 IP 소스에서 액세스하는 경우에만 DynamoDB 리소스에 액세스하도록 허용하고 싶을 수 있습니다. `Condition` 문에 이러한 IP 주소를 지정하세요.

다음 예시에서는 소스 IP가 `54.240.143.0/24` 및 `2001:DB8:1234:5678::/64`일 때 사용자 *John*이 모든 DynamoDB 리소스에 액세스할 수 있도록 허용합니다.

JSON

```
{
  "Id":"PolicyId2",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AllowIPmix",
      "Effect":"Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:user/username"
      },
      "Action":"dynamodb:*",
      "Resource":"*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "54.240.143.0/24",
            "2001:DB8:1234:5678::/64"
          ]
        }
      }
    }
  ]
}
```

소스가 특정 VPC 엔드포인트(예: *vpce-1a2b3c4d*)인 경우를 제외하고 DynamoDB 리소스에 대한 모든 액세스를 거부할 수도 있습니다.

JSON

```
{
  "Id":"PolicyId",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AccessToSpecificVPCEOnly",
```

```
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals":{
          "aws:sourceVpce":"vpce-1a2b3c4d"
        }
      }
    }
  ]
}
```

## IAM 역할이 지정된 리소스 기반 정책

리소스 기반 정책에서 IAM 서비스 역할을 지정할 수도 있습니다. 이 역할을 수임하는 IAM 엔터티는 해당 역할에 지정된 허용 작업 및 리소스 기반 정책 내의 특정 리소스 세트로만 제한됩니다.

다음 예시는 IAM 엔터티가 *MusicCollection* 및 *MusicCollection* DynamoDB 리소스에서 모든 DynamoDB 작업을 수행할 수 있도록 허용합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/role-name" },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/*"
      ]
    }
  ]
}
```

# DynamoDB 리소스 기반 정책 고려 사항

DynamoDB 리소스의 리소스 기반 정책을 정의할 때는 다음 사항을 고려해야 합니다.

## 일반적인 고려 사항

- 리소스 기반 정책 문서에 지원되는 최대 크기는 20KB입니다. DynamoDB는 이 한도를 기준으로 정책의 크기를 계산할 때 공백을 계산합니다.
- 주어진 리소스에 대한 정책의 후속 업데이트는 동일한 리소스에 대한 정책이 성공적으로 업데이트된 후 15초 동안 차단됩니다.
- 현재는 기존 스트림에만 리소스 기반 정책을 연결할 수 있습니다. 스트림을 생성하는 동안에는 정책을 스트림에 연결할 수 없습니다.

## 글로벌 테이블 고려 사항

- 리소스 기반 정책은 글로벌 테이블 버전 2017.11.29(레거시) 복제본에 지원되지 않습니다.
- 리소스 기반 정책 내에서 글로벌 테이블의 데이터를 복제하기 위한 DynamoDB 서비스 연결 역할(SLR)에 대한 작업이 거부되면 복제본 추가 또는 삭제가 실패하고 오류가 발생합니다.
- AWS::DynamoDB::GlobalTable 리소스는 스택 업데이트를 배포하는 리전 이외의 리전에서 동일한 스택 업데이트에서 복제본을 생성하고 해당 복제본에 리소스 기반 정책을 추가하는 것을 지원하지 않습니다.

## 크로스 계정 고려 사항

- 리소스 기반 정책을 사용하는 크로스 계정 액세스는 AWS 관리형 KMS 정책에 크로스 계정 액세스 권한을 부여할 수 없기 때문에 AWS 관리형 키를 사용한 암호화된 테이블을 지원하지 않습니다.

## AWS CloudFormation 고려 사항

- 리소스 기반 정책은 드리프트 감지를 지원하지 않습니다. AWS CloudFormation 스택 템플릿 외부에서 리소스 기반 정책을 업데이트하는 경우 CloudFormation 스택에 변경 사항을 업데이트해야 합니다.
- 리소스 기반 정책은 대역 외 변경을 지원하지 않습니다. CloudFormation 템플릿 외부에서 정책을 추가, 업데이트 또는 삭제하는 경우 템플릿 내에 정책이 변경되지 않으면 변경 내용을 덮어쓰지 않습니다.

예를 들어 템플릿에 리소스 기반 정책이 포함되어 있으며 나중에 템플릿 외부에서 업데이트한다고 가정해 보겠습니다. 템플릿 내에서 정책을 변경하지 않으면 DynamoDB의 업데이트된 정책이 템플릿 내의 정책과 동기화되지 않습니다.

반대로 템플릿에 리소스 기반 정책이 포함되어 있지 않지만 템플릿 외부에서 정책을 추가한다고 가정해 보겠습니다. 이 정책은 템플릿에 추가하지 않는 한 DynamoDB에서 제거되지 않습니다. 템플릿에 정책을 추가하고 스택을 업데이트하면 DynamoDB의 기존 정책이 템플릿에 정의된 것과 일치하도록 업데이트됩니다.

# DynamoDB 리소스 기반 정책 모범 사례

이 주제에서는 DynamoDB 리소스에 대한 액세스 권한을 정의하는 모범 사례와 이러한 리소스에 허용되는 작업을 설명합니다.

## DynamoDB 리소스에 대한 액세스 제어 간소화

DynamoDB 리소스에 액세스해야 하는 AWS Identity and Access Management 보안 주체가 리소스 소유자와 동일한 AWS 계정에 속한 경우 각 보안 주체에 대해 IAM 자격 증명 기반 정책이 필요하지 않습니다. 주어진 리소스에 연결된 리소스 기반 정책이면 충분합니다. 이러한 유형의 구성은 액세스 제어를 간소화합니다.

## 리소스 기반 정책으로 DynamoDB 리소스 보호

모든 DynamoDB 테이블 및 스트림에 대해 리소스 기반 정책을 생성하여 이러한 리소스에 대한 액세스 제어를 적용합니다. 리소스 기반 정책을 사용하면 리소스 수준에서 권한을 중앙 집중화하고, DynamoDB 테이블, 인덱스 및 스트림에 대한 액세스 제어를 간소화하고, 관리 오버헤드를 줄일 수 있습니다. 테이블이나 스트림에 리소스 기반 정책이 지정되지 않은 경우, IAM 보안 주체와 연결된 자격 증명 기반 정책에서 액세스를 허용하지 않는 한 테이블 또는 스트림에 대한 액세스는 암시적으로 거부됩니다.

## 최소 권한 적용

DynamoDB 리소스에 대한 리소스 기반 정책과 함께 권한을 설정하는 경우 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. 워크로드 또는 사용 사례에 필요한 권한을 탐색하는 동안 광범위한 권한으로 시작할 수 있습니다. 사용 사례가 발전함에 따라 최소 권한을 향해 나아가도록 부여하는 권한을 줄이기 위해 노력할 수 있습니다.

### 최소 권한 정책을 생성하기 위해 크로스 계정 액세스 활동 분석

IAM Access Analyzer는 리소스 기반 정책에 지정된 외부 엔터티에 대한 크로스 계정 액세스를 보고하고, 권한을 세분화하고 최소 권한을 준수하는 데 도움이 되는 가시성을 제공합니다. 정책 생성에 대한 자세한 내용은 IAM Access Analyzer 정책 생성을 참조하세요.

### IAM Access Analyzer를 사용하여 최소 권한 정책 생성

작업을 수행하는 데 필요한 권한만 부여하기 위해 AWS CloudTrail에 로깅된 액세스 활동에 따라 정책을 생성할 수 있습니다. IAM Access Analyzer는 정책에서 사용하는 서비스와 작업을 분석합니다.

## DynamoDB의 속성 기반 액세스 제어 사용

속성 기반 액세스 제어(ABAC)는 ID 기반 정책 또는 기타 AWS 정책(예: 리소스 기반 정책 및 조직 IAM 정책)의 태그 조건을 기반으로 액세스 권한을 정의하는 권한 부여 전략입니다. DynamoDB 테이블에 태그를 연결하면 태그 기반 조건을 기준으로 평가됩니다. 테이블과 연결된 인덱스는 테이블에 추가하는 태그를 상속합니다. 각 DynamoDB 테이블에 최대 50개의 태그를 추가할 수 있습니다. 테이블의 모든 태그에 지원되는 최대 크기는 10KB입니다. DynamoDB 리소스 태그 지정 및 태그 지정 제한에 대한 자세한 내용은 DynamoDB에서 리소스 태그 지정 및 the section called “DynamoDB의 태그 지정 제한 사항” 섹션을 참조하세요.

태그를 사용한 AWS 리소스에 대한 액세스 제어에 대한 자세한 내용은 IAM 사용 설명서의 다음 항목을 참조하세요.

- AWS용 ABAC란 무엇인가요?
- 태그를 사용한 AWS 리소스 액세스 제어

ABAC를 사용하면 팀과 애플리케이션에 서로 다른 액세스 수준을 적용하여 더 적은 수의 정책으로 DynamoDB 테이블에 작업을 수행할 수 있습니다. IAM 정책 조건 요소에서 태그를 지정하여 DynamoDB 테이블 또는 인덱스에 대한 액세스를 제어할 수 있습니다. 이러한 조건은 DynamoDB 테이블 및 인덱스에 대해 IAM 위탁자, 사용자 또는 역할이 가진 액세스 수준을 결정합니다. IAM 위탁자가 DynamoDB에 액세스 요청을 하면 리소스 및 ID의 태그가 IAM 정책의 태그 조건을 기준으로 평가됩니다. 그 이후에는 태그 조건이 충족되는 경우에만 정책이 적용됩니다. 이렇게 하면 다음 중 하나를 효과적으로 설명하는 IAM 정책을 만들 수 있습니다.

- 사용자가 키가 *X*, 값이 *Y*인 태그가 있는 리소스만 관리하도록 허용합니다.
- *X* 키로 태그가 지정된 리소스에 대한 모든 사용자의 액세스를 거부합니다.

예를 들어 사용자가 태그 키-값 페어 "environment": "staging"이 있는 테이블만 업데이트하도록 허용하는 정책을 생성할 수 있습니다. aws:ResourceTag 조건 키를 사용하여 해당 테이블에 연결된 태그를 기반으로 테이블에 대한 액세스를 허용하거나 거부할 수 있습니다.

AWS Management Console, AWS API, AWS Command Line Interface(AWS CLI), AWS SDK 또는 AWS CloudFormation을 사용하여 정책을 만드는 동안 속성 기반 조건을 포함할 수 있습니다.

다음 예제에서는 MusicTable이라는 테이블에 이름이 environment, 값이 production인 태그 키가 포함된 경우 UpdateItem 작업을 허용합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/MusicTable",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "production"
        }
      }
    }
  ]
}
```

주제

- ABAC를 사용해야 하는 이유는 무엇입니까?
- DynamoDB와 함께 ABAC를 구현하기 위한 조건 키
- DynamoDB와 함께 ABAC를 사용할 때의 고려 사항
- DynamoDB에서 ABAC 활성화
- DynamoDB 테이블 및 인덱스와 함께 ABAC 사용
- DynamoDB 테이블 및 인덱스와 함께 ABAC를 사용하는 예제
- DynamoDB 테이블 및 인덱스의 일반적인 ABAC 오류 해결

## ABAC를 사용해야 하는 이유는 무엇입니까?

- 더 간단한 정책 관리: 각 IAM 위탁자에 대한 액세스 수준을 정의하기 위해 서로 다른 정책을 만들 필요가 없으므로 더 적은 수의 정책을 사용합니다.
- 규모 조정 가능한 액세스 제어: ABAC를 사용하면 새 DynamoDB 리소스를 만들 때 정책을 업데이트할 필요가 없으므로 액세스 제어 규모 조정이 더 쉬워집니다. 태그를 사용하여 리소스 태그와 일치하는 태그가 포함된 IAM 위탁자에 대한 액세스를 승인할 수 있습니다. 새 IAM 위탁자 또는 DynamoDB 리소스를 온보딩하고 적절한 태그를 적용하여 정책을 변경할 필요 없이 필요한 권한을 자동으로 부여할 수 있습니다.
- 세분화된 권한 관리: 정책을 만들 때 최소 권한을 부여하는 것이 가장 좋습니다. ABAC를 사용하면 IAM 위탁자용 태그를 만들고 이를 사용하여 IAM 위탁자의 태그와 일치하는 특정 작업 및 리소스에 대한 액세스를 부여할 수 있습니다.
- 회사 디렉터리와 맞춤: 태그를 회사 디렉터리의 기존 직원 속성과 매핑하여 액세스 제어 정책을 조직 구조에 맞출 수 있습니다.

## DynamoDB와 함께 ABAC를 구현하기 위한 조건 키

AWS 정책에서 다음 조건 키를 사용하여 DynamoDB 테이블 및 인덱스에 대한 액세스 수준을 제어할 수 있습니다.

- aws:ResourceTag/tag-key: DynamoDB 테이블 또는 인덱스의 태그 키-값 페어가 정책의 태그 키 및 값과 일치하는지 여부에 따라 액세스를 제어합니다. 이 조건 키는 기존 테이블 또는 인덱스에서 작동하는 모든 API와 관련이 있습니다.

  `dynamodb:ResourceTag` 조건은 리소스에 태그를 연결하지 않은 것처럼 평가됩니다.
- aws:RequestTag/tag-key: 요청에서 전달된 태그 키-값 페어를 정책에서 지정한 태그 페어와 비교하도록 허용합니다. 이 조건 키는 요청 페이로드의 일부로 태그가 포함된 API와 관련이 있습니다. 이러한 API에는 CreateTable 및 TagResource가 포함됩니다.
- aws:TagKeys: 요청의 태그 키를 정책에서 지정한 키와 비교합니다. 이 조건 키는 요청 페이로드의 일부로 태그가 포함된 API와 관련이 있습니다. 해당 API에는 `CreateTable`, `TagResource` 및 `UntagResource`가 포함됩니다.

## DynamoDB와 함께 ABAC를 사용할 때의 고려 사항

ABAC를 DynamoDB 테이블 또는 인덱스와 함께 사용하는 경우 다음 고려 사항이 적용됩니다.

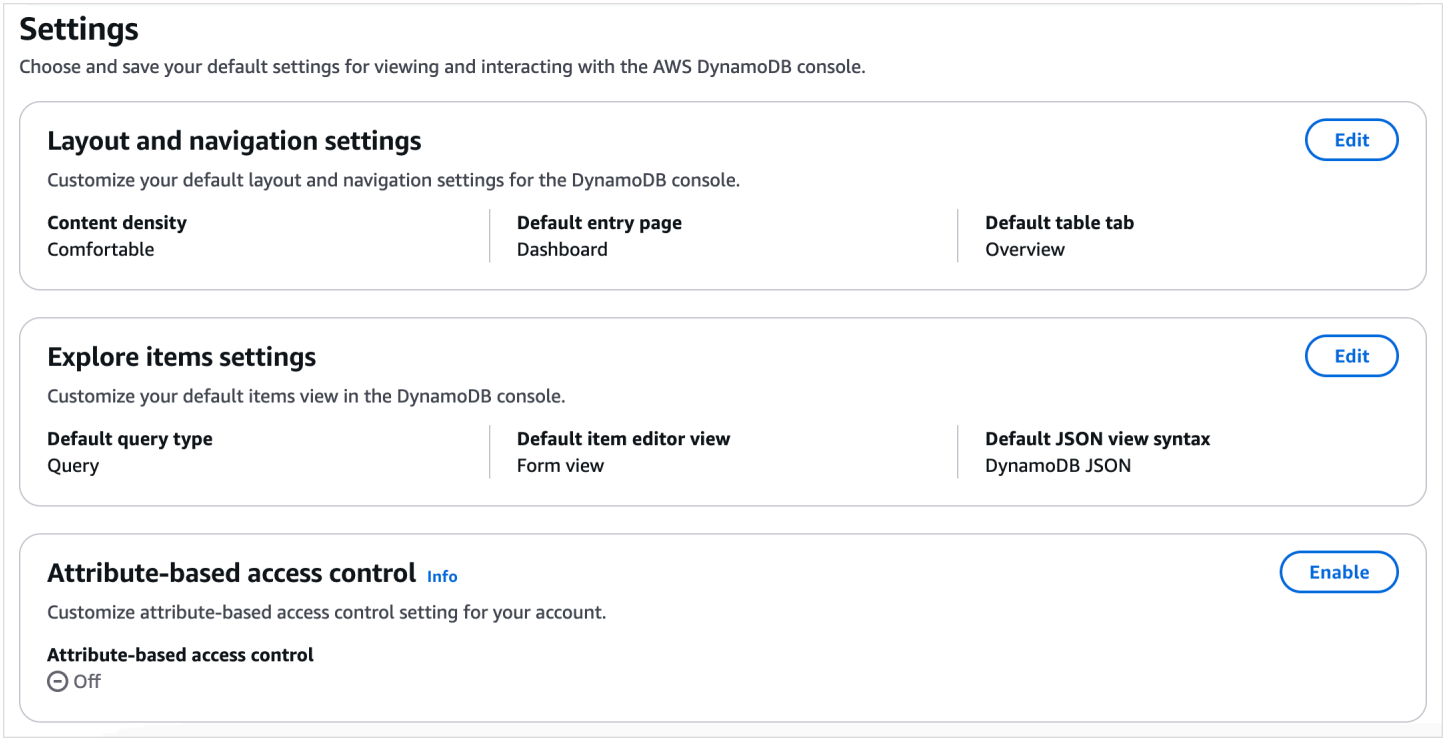
- DynamoDB 스트림에는 태그 지정 및 ABAC가 지원되지 않습니다.
- DynamoDB 백업에는 태그 지정 및 ABAC가 지원되지 않습니다. 백업과 함께 ABAC를 사용하려면 AWS Backup을 사용하는 것이 좋습니다.
- 태그는 복원된 테이블에 보존되지 않습니다. 정책에서 태그 기반 조건을 사용하려면 복원된 테이블에 태그를 추가해야 합니다.

## DynamoDB에서 ABAC 활성화

대부분의 AWS 계정에서는 ABAC가 기본적으로 활성화됩니다. DynamoDB 콘솔을 사용하여 계정에 ABAC가 활성화되어 있는지 확인할 수 있습니다. 이렇게 하려면 dynamodb:GetAbacStatus 권한이 있는 역할로 DynamoDB 콘솔을 열어야 합니다. 그런 다음 DynamoDB 콘솔의 설정 페이지를 엽니다.

속성 기반 액세스 제어 카드가 표시되지 않거나 카드에 켜짐 상태가 표시되면 계정에 ABAC가 활성화되었음을 의미합니다. 그러나 다음 이미지와 같이 꺼짐 상태의 속성 기반 액세스 제어 카드가 표시되면 계정에 ABAC가 활성화되지 않은 것입니다.

속성 기반 액세스 제어 - 활성화되지 않음




ID 기반 정책 또는 기타 정책에 지정된 태그 기반 조건을 여전히 감사해야 하는 AWS 계정에는 ABAC가 활성화되지 않습니다. 계정에 ABAC가 활성화되지 않은 경우 DynamoDB 테이블 또는 인덱스에 적용되도록 의도된 정책의 태그 기반 조건은 리소스 또는 API 요청에 태그가 없는 것처럼 평가됩니다. 계

정에 ABAC가 활성화되면 테이블 또는 API 요청에 연결된 태그를 고려하여 계정 정책의 태그 기반 조건이 평가됩니다.

계정에 ABAC를 활성화하려면 먼저 감사 정책 섹션에 설명된 대로 정책을 감사하는 것이 좋습니다. 그런 다음 IAM 정책에 ABAC에 필요한 권한을 포함합니다. 마지막으로 콘솔에서 ABAC 활성화에 설명된 단계를 수행하여 현재 리전의 계정에 ABAC를 활성화합니다. ABAC를 활성화하여 옵트인한 후 7일 이내에 옵트아웃할 수 있습니다.

주제

- ABAC를 활성화하기 전에 정책 감사
- ABAC를 활성화하는 데 필요한 IAM 권한
- 콘솔에서 ABAC 활성화

## ABAC를 활성화하기 전에 정책 감사

계정에 ABAC를 활성화하기 전에 정책을 감사하여 계정 내 정책에 존재할 수 있는 태그 기반 조건이 의도한 대로 설정되었는지 확인합니다. 정책을 감사하면 ABAC를 활성화한 후 DynamoDB 워크플로의 권한 부여 변경으로 인한 예상치 못한 상황을 방지할 수 있습니다. 태그와 함께 속성 기반 조건을 사용하는 예제와 ABAC 구현 전후의 동작을 보려면 DynamoDB 테이블 및 인덱스와 함께 ABAC를 사용하는 예제 섹션을 참조하세요.

## ABAC를 활성화하는 데 필요한 IAM 권한

현재 리전의 계정에 ABAC를 활성화하려면 `dynamodb:UpdateAbacStatus` 권한이 필요합니다. 계정에 ABAC가 활성화되어 있는지 확인하려면 `dynamodb:GetAbacStatus` 권한도 있어야 합니다. 이 권한을 사용하면 모든 리전의 계정에 대한 ABAC 상태를 볼 수 있습니다. 이러한 권한은 DynamoDB 콘솔에 액세스하는 데 필요한 권한에 더해 필요합니다.

다음 IAM 정책은 ABAC를 활성화하고 현재 리전의 계정에 대한 상태를 볼 수 있는 권한을 부여합니다.

```
{
"version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateAbacStatus",
                "dynamodb:GetAbacStatus"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

### 콘솔에서 ABAC 활성화

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 상단 탐색 창에서 ABAC를 활성화할 리전을 선택합니다.
3. 왼쪽 탐색 창에서 설정을 선택합니다.
4. 설정 페이지에서 다음 작업을 수행합니다.

   a. 속성 기반 액세스 제어 카드에서 활성화를 선택합니다.

   b. 속성 기반 액세스 제어 설정 확인 상자에서 활성화를 선택하여 선택을 확인합니다.

   이렇게 하면 현재 리전에 대한 ABAC가 활성화되고 속성 기반 액세스 제어 카드에 켜짐 상태가 표시됩니다.

   콘솔에서 ABAC를 활성화한 후 옵트아웃하려면 옵트인 후 7일 이내에 옵트아웃할 수 있습니다. 옵트아웃하려면 설정 페이지의 속성 기반 액세스 제어 카드에서 비활성화를 선택합니다.

   > **Note**
   >
   > ABAC의 상태를 업데이트하는 것은 비동기 작업입니다. 변경 사항 적용은 최종적으로 일관되므로 정책의 태그가 즉시 평가되지 않으면 얼마 동안 기다려야 할 수 있습니다.

## DynamoDB 테이블 및 인덱스와 함께 ABAC 사용

다음 단계에서는 ABAC를 사용하여 권한을 설정하는 방법을 보여줍니다. 이 예제 시나리오에서는 DynamoDB 테이블에 태그를 추가하고 태그 기반 조건을 포함하는 정책을 사용하여 IAM 역할을 생성합니다. 그런 다음 태그 조건을 일치시켜 DynamoDB 테이블에서 허용되는 권한을 테스트합니다.

주제

- 1단계: DynamoDB 테이블에 태그 추가
- 2단계: 태그 기반 조건을 포함한 정책을 사용하여 IAM 역할 생성

- 3단계: 허용된 권한 테스트

### 1단계: DynamoDB 테이블에 태그 추가

AWS Management Console, AWS API, AWS Command Line Interface(AWS CLI), AWS SDK 또는 AWS CloudFormation을 사용하여 새 DynamoDB 테이블 또는 기존 DynamoDB 테이블에 태그를 추가할 수 있습니다. 예를 들어 다음 tag-resource CLI 명령은 `MusicTable`이라는 테이블에 태그를 추가합니다.

```
aws dynamodb tag-resource –resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/
MusicTable –tags Key=environment,Value=staging
```

### 2단계: 태그 기반 조건을 포함한 정책을 사용하여 IAM 역할 생성

aws:ResourceTag/tag-key 조건 키로 IAM 정책을 만들어 IAM 정책에 지정된 태그 키-값 페어를 테이블에 연결된 키-값 페어와 비교합니다. 다음 예제 정책은 사용자가 테이블에 태그 키-값 페어 `"environment": "staging"`이 포함된 경우 테이블에 항목을 입력하거나 업데이트할 수 있도록 허용합니다. 테이블에 지정된 태그 키-값 페어가 없는 경우 이러한 작업은 거부됩니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "staging"
                }
            }
        }
    ]
}
```

### 3단계: 허용된 권한 테스트

1. AWS 계정의 테스트 사용자 또는 역할에 IAM 정책을 연결합니다. 사용하는 IAM 위탁자가 다른 정책을 통해 DynamoDB 테이블에 이미 액세스할 수 없는지 확인합니다.
2. DynamoDB 테이블에 값이 `"staging"`인 `"environment"` 태그 키가 포함되어 있는지 확인합니다.
3. 태그가 지정된 테이블에서 `dynamodb:PutItem` 및 `dynamodb:UpdateItem` 작업을 수행합니다. `"environment": "staging"` 태그 키-값 페어가 있는 경우 이러한 작업이 성공합니다.

   `"environment": "staging"` 태그 키-값 페어가 없는 테이블에서 이러한 작업을 수행하면 `AccessDeniedException`과 함께 요청이 실패합니다.

다음 섹션에 설명된 다른 샘플 사용 사례를 검토하여 ABAC를 구현하고 추가 테스트를 수행할 수도 있습니다.

## DynamoDB 테이블 및 인덱스와 함께 ABAC를 사용하는 예제

다음 예제에서는 태그를 사용하여 속성 기반 조건을 구현하는 몇 가지 사용 사례를 보여줍니다.

주제


- 예제 1: aws:ResourceTag를 사용하여 작업 허용
- 예제 2: aws:RequestTag를 사용하여 작업 허용
- 예제 3: aws:TagKeys를 사용하여 작업 거부


### 예제 1: aws:ResourceTag를 사용하여 작업 허용

`aws:ResourceTag/tag-key` 조건 키를 사용하여 IAM 정책에 지정된 태그 키-값 페어를 DynamoDB 테이블에 연결된 키-값 페어와 비교할 수 있습니다. 예를 들어 IAM 정책 및 테이블의 태그 조건이 일치하는 경우 PutItem과 같은 특정 작업을 허용할 수 있습니다. 이렇게 하려면 다음 단계를 수행하세요.

Using the AWS CLI

1. 테이블을 생성합니다. 다음 예제에서는 create-table AWS CLI 명령을 사용하여 `myMusicTable`이라는 테이블을 생성합니다.

```
aws dynamodb create-table \
  --table-name myMusicTable \
  --attribute-definitions AttributeName=id,AttributeType=S \
```

```
  --key-schema AttributeName=id,KeyType=HASH \
  --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
  --region us-east-1
```

2. 이 테이블에 태그를 추가합니다. 다음 tag-resource AWS CLI 명령 예제는 태그 키-값 페어 `Title: ProductManager`를 `myMusicTable`에 추가합니다.

```
aws dynamodb tag-resource --region us-east-1 --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/myMusicTable --tags Key=Title,Value=ProductManager
```

3. 다음 예제와 같이 인라인 정책을 만들고 AmazonDynamoDBReadOnlyAccess AWS 관리형 정책이 연결된 역할에 추가합니다.

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
      }
    }
  ]
}
```

   이 정책은 테이블에 연결된 태그 키와 값이 정책에 지정된 태그와 일치할 때 테이블에 대한 `PutItem` 작업을 허용합니다.

4. 3단계에 설명된 정책으로 역할을 수임합니다.
5. put-item AWS CLI 명령을 사용하여 항목을 `myMusicTable`에 입력합니다.

```
aws dynamodb put-item \
    --table-name myMusicTable --region us-east-1 \
    --item '{
        "id": {"S": "2023"},
```

```
        "title": {"S": "Happy Day"},
        "info": {"M": {
            "rating": {"N": "9"},
            "Artists": {"L": [{"S": "Acme Band"}, {"S": "No One You Know"}]},
            "release_date": {"S": "2023-07-21"}
        }}
    }'
```

6. 테이블을 스캔하여 항목이 테이블에 추가되었는지 확인합니다.

```
aws dynamodb scan --table-name myMusicTable  --region us-east-1
```

Using the AWS SDK for Java 2.x

1. 테이블을 생성합니다. 다음 예제에서는 CreateTable API를 사용하여 myMusicTable이라는 테이블을 생성합니다.

```
DynamoDbClient dynamoDB = DynamoDbClient.builder().region(region).build();
CreateTableRequest createTableRequest = CreateTableRequest.builder()
    .attributeDefinitions(
        Arrays.asList(
            AttributeDefinition.builder()
            .attributeName("id")
            .attributeType(ScalarAttributeType.S)
            .build()
        )
    )
    .keySchema(
        Arrays.asList(
            KeySchemaElement.builder()
            .attributeName("id")
            .keyType(KeyType.HASH)
            .build()
        )
    )
    .provisionedThroughput(ProvisionedThroughput.builder()
        .readCapacityUnits(5L)
        .writeCapacityUnits(5L)
        .build()
    )
    .tableName("myMusicTable")
    .build();
```

```
CreateTableResponse createTableResponse =
 dynamoDB.createTable(createTableRequest);
String tableArn = createTableResponse.tableDescription().tableArn();
String tableName = createTableResponse.tableDescription().tableName();
```

2. 이 테이블에 태그를 추가합니다. 다음 예제의 TagResource API는 태그 키-값 페어 `Title: ProductManager`를 `myMusicTable`에 추가합니다.

```
TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
    .resourceArn(tableArn)
    .tags(
        Arrays.asList(
            Tag.builder()
            .key("Title")
            .value("ProductManager")
            .build()
        )
    )
    .build();
dynamoDB.tagResource(tagResourceRequest);
```

3. 다음 예제와 같이 인라인 정책을 만들고 AmazonDynamoDBReadOnlyAccess AWS 관리형 정책이 연결된 역할에 추가합니다.

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
      }
    }
  ]
}
```

이 정책은 테이블에 연결된 태그 키와 값이 정책에 지정된 태그와 일치할 때 테이블에 대한 `PutItem` 작업을 허용합니다.

4. 3단계에 설명된 정책으로 역할을 수임합니다.
5. PutItem API를 사용하여 항목을 `myMusicTable`에 입력합니다.

```
HashMap<String, AttributeValue> info = new HashMap<>();
info.put("rating", AttributeValue.builder().s("9").build());
info.put("artists", AttributeValue.builder().ss(List.of("Acme Band","No One You
 Know").build());
info.put("release_date", AttributeValue.builder().s("2023-07-21").build());

HashMap<String, AttributeValue> itemValues = new HashMap<>();
itemValues.put("id", AttributeValue.builder().s("2023").build());
itemValues.put("title", AttributeValue.builder().s("Happy Day").build());
itemValues.put("info", AttributeValue.builder().m(info).build());


PutItemRequest putItemRequest = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();
dynamoDB.putItem(putItemRequest);
```

6. 테이블을 스캔하여 항목이 테이블에 추가되었는지 확인합니다.

```
ScanRequest scanRequest = ScanRequest.builder()
                .tableName(tableName)
                .build();

ScanResponse scanResponse = dynamoDB.scan(scanRequest);
```

Using the AWS SDK for Python (Boto3)

1. 테이블을 생성합니다. 다음 예제에서는 CreateTable API를 사용하여 `myMusicTable`이라는 테이블을 생성합니다.

```
create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
```

```
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    },
)

table_arn = create_table_response['TableDescription']['TableArn']
```

2. 이 테이블에 태그를 추가합니다. 다음 예제의 TagResource API는 태그 키-값 페어 `Title: ProductManager`를 `myMusicTable`에 추가합니다.

```
tag_resouce_response = ddb_client.tag_resource(
    ResourceArn=table_arn,
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ]
)
```

3. 다음 예제와 같이 인라인 정책을 만들고 AmazonDynamoDBReadOnlyAccess AWS 관리형 정책이 연결된 역할에 추가합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
```

```
        "Action": "dynamodb:PutItem",
        "Resource": "arn:aws:dynamodb:*:*:table/*",
        "Condition": {
            "StringEquals": {
            "aws:ResourceTag/Title": "ProductManager"
            }
        }
        }
    ]
    }
```

이 정책은 테이블에 연결된 태그 키와 값이 정책에 지정된 태그와 일치할 때 테이블에 대한 `PutItem` 작업을 허용합니다.

4. 3단계에 설명된 정책으로 역할을 수임합니다.
5. PutItem API를 사용하여 항목을 `myMusicTable`에 입력합니다.

```
put_item_response = client.put_item(
    TableName = 'myMusicTable'
    Item = {
        'id': '2023',
        'title': 'Happy Day',
        'info': {
            'rating': '9',
            'artists': ['Acme Band','No One You Know'],
            'release_date': '2023-07-21'
        }
    }
)
```

6. 테이블을 스캔하여 항목이 테이블에 추가되었는지 확인합니다.

```
scan_response = client.scan(
    TableName='myMusicTable'
)
```

ABAC이 없는 경우

AWS 계정에 ABAC가 활성화되지 않은 경우 IAM 정책 및 DynamoDB 테이블의 태그 조건이 일치하지 않습니다. 따라서 이 `PutItem` 작업은 `AmazonDynamoDBReadOnlyAccess` 정책의 영향으로 인해 `AccessDeniedException`을 반환합니다.

```
An error occurred (AccessDeniedException) when calling the PutItem operation:
 User: arn:aws:sts::123456789012:assumed-role/DynamoDBReadOnlyAccess/Alice is
 not authorized to perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-
east-1:123456789012:table/myMusicTable because no identity-based policy allows the
 dynamodb:PutItem action.
```

ABAC가 있는 경우

AWS 계정에 ABAC가 활성화된 경우 `put-item` 작업이 성공적으로 완료되고 테이블에 새 항목이 추가됩니다. 이는 테이블의 인라인 정책이 IAM 정책과 테이블의 태그 조건이 일치하는 경우 `PutItem` 작업을 허용하기 때문입니다.

## 예제 2: aws:RequestTag를 사용하여 작업 허용

aws:RequestTag/tag-key 조건 키를 사용하여 요청에 전달된 태그 키-값 페어를 IAM 정책에 지정된 태그 페어와 비교할 수 있습니다. 예를 들어 태그 조건이 일치하지 않는 경우 `aws:RequestTag`를 사용하여와 `CreateTable`과 같은 특정 작업을 허용할 수 있습니다. 이렇게 하려면 다음 단계를 수행하세요.

Using the AWS CLI

1. 다음 예제와 같이 인라인 정책을 만들고 AmazonDynamoDBReadOnlyAccess AWS 관리형 정책이 연결된 역할에 추가합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
```

```
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. "Owner": "John"의 태그 키-값 페어가 포함된 테이블을 만듭니다.

```
aws dynamodb create-table \
--attribute-definitions AttributeName=ID,AttributeType=S \
--key-schema AttributeName=ID,KeyType=HASH  \
--provisioned-throughput ReadCapacityUnits=1000,WriteCapacityUnits=500 \
--region us-east-1 \
--tags Key=Owner,Value=John \
--table-name myMusicTable
```

Using the AWS SDK for Python (Boto3)

1. 다음 예제와 같이 인라인 정책을 만들고 AmazonDynamoDBReadOnlyAccess AWS 관리형 정책이 연결된 역할에 추가합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
```

```
    ]
}
```

2. "Owner": "John"의 태그 키-값 페어가 포함된 테이블을 만듭니다.

```
ddb_client = boto3.client('dynamodb')

create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 1000,
        'WriteCapacityUnits': 500
    },
    Tags=[
        {
            'Key': 'Owner',
            'Value': 'John'
        },
    ],
)
```

### ABAC이 없는 경우

AWS 계정에 ABAC가 활성화되지 않은 경우 인라인 정책 및 DynamoDB 테이블의 태그 조건이 일치하지 않습니다. 따라서 CreateTable 요청이 실패하고 테이블이 생성되지 않습니다.

```
An error occurred (AccessDeniedException) when calling the CreateTable operation:
 User: arn:aws:sts::123456789012:assumed-role/Admin/John is not authorized to perform:
 dynamodb:CreateTable on resource: arn:aws:dynamodb:us-east-1:123456789012:table/
myMusicTable because no identity-based policy allows the dynamodb:CreateTable action.
```

ABAC가 있는 경우

AWS 계정에 ABAC가 활성화된 경우 테이블 만들기 요청이 성공적으로 완료됩니다. 태그 키-값 쌍 `"Owner": "John"`이 `CreateTable` 요청에 있으므로 인라인 정책을 통해 사용자 `John`이 `CreateTable` 작업을 수행할 수 있습니다.

### 예제 3: aws:TagKeys를 사용하여 작업 거부

aws:TagKeys 조건 키를 사용하여 요청의 태그 키를 IAM 정책에 지정된 키와 비교할 수 있습니다. 예를 들어 요청에 특정 태그 키가 없는 경우 `aws:TagKeys`를 사용하여 `CreateTable`과 같은 특정 작업을 거부할 수 있습니다. 이렇게 하려면 다음 단계를 수행하세요.

Using the AWS CLI

1. 다음 예제와 같이 AmazonDynamoDBFullAccess AWS 관리형 정책이 연결된 역할에 고객 관리형 정책을 추가합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. 정책이 연결된 역할을 수임하고 태그 키 `Title`로 테이블을 만듭니다.

```
aws dynamodb create-table \
--attribute-definitions AttributeName=ID,AttributeType=S \
--key-schema AttributeName=ID,KeyType=HASH  \
--provisioned-throughput ReadCapacityUnits=1000,WriteCapacityUnits=500 \
--region us-east-1 \
--tags Key=Title,Value=ProductManager \
--table-name myMusicTable
```

Using the AWS SDK for Python (Boto3)

1. 다음 예제와 같이 AmazonDynamoDBFullAccess AWS 관리형 정책이 연결된 역할에 고객 관리형 정책을 추가합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. 정책이 연결된 역할을 수임하고 태그 키 `Title`로 테이블을 만듭니다.

```
ddb_client = boto3.client('dynamodb')
```

```
create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 1000,
        'WriteCapacityUnits': 500
    },
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ],
)
```

ABAC이 없는 경우

AWS 계정에 ABAC가 활성화되지 않은 경우 DynamoDB는 `create-table` 명령의 태그 키를 IAM으로 전송하지 않습니다. `Null` 조건은 요청에 태그 키가 없는 경우 조건이 `false`로 평가되도록 합니다. Deny 정책이 일치하지 않으므로 `create-table` 명령이 성공적으로 완료됩니다.

ABAC가 있는 경우

AWS 계정에 ABAC가 활성화된 경우 `create-table` 명령에 전달된 태그 키가 IAM으로 전달됩니다. 태그 키 `Title`은 Deny 정책에 있는 조건 기반 태그 키 `CostCenter`를 기준으로 평가됩니다. `StringNotEquals` 연산자로 인해 태그 키 `Title`이 Deny 정책에 있는 태그 키와 일치하지 않습니다. 따라서 `CreateTable` 작업이 실패하고 테이블이 생성되지 않습니다. `create-table` 명령을 실행하면 `AccessDeniedException`이 반환됩니다.

```
An error occurred (AccessDeniedException) when calling the CreateTable operation:
 User: arn:aws:sts::123456789012:assumed-role/DynamoFullAccessRole/ProductManager is
 not authorized to perform: dynamodb:CreateTable on resource: arn:aws:dynamodb:us-
east-1:123456789012:table/myMusicTable with an explicit deny in an identity-based
 policy.
```

## DynamoDB 테이블 및 인덱스의 일반적인 ABAC 오류 해결

이 주제에서는 DynamoDB 테이블 또는 인덱스에서 ABAC를 구현하는 동안 발생할 수 있는 일반적인 오류 및 문제에 대한 문제 해결 조언을 제공합니다.

정책의 서비스별 조건 키로 인해 오류 발생

서비스별 조건 키는 유효한 조건 키로 간주되지 않습니다. 정책에서 이러한 키를 사용한 경우 오류가 발생합니다. 이 문제를 해결하려면 서비스별 조건 키를 적절한 조건 키로 바꾸어 DynamoDB에서 ABAC를 구현해야 합니다.

예를 들어 PutItem 요청을 수행하는 인라인 정책에서 `dynamodb:ResourceTag` 조건 키를 사용했다고 가정해 보겠습니다. `AccessDeniedException`에서 요청이 실패한다고 가정합니다. 다음 예제에서는 `dynamodb:ResourceTag` 조건 키가 있는 잘못된 인라인 정책을 보여줍니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

이 문제를 해결하려면 다음 예제와 같이 dynamodb:ResourceTag 조건 키를 aws:ResourceTag로 바꿉니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

ABAC를 옵트아웃할 수 없음

지원를 통해 계정에 대해 ABAC가 활성화된 경우 DynamoDB 콘솔을 통해 ABAC를 옵트아웃할 수 없습니다. 옵트아웃하려면 지원에 문의하세요.

다음과 같은 경우에만 ABAC를 직접 옵트아웃할 수 있습니다.

- DynamoDB 콘솔을 통해 옵트인하는 셀프 서비스 방법을 사용했습니다.
- 옵트인 후 7일 이내에 옵트아웃합니다.

# DynamoDB의 데이터 보호

Amazon DynamoDB에서는 미션 크리티컬 및 기본 데이터 스토리지에 적합하게 설계된, 내구성이 뛰어난 스토리지 인프라를 제공합니다. 데이터는 Amazon DynamoDB 리전의 여러 시설에 걸쳐 여러 디바이스에 중복 저장됩니다.

DynamoDB는 저장된 유휴 사용자 데이터뿐 아니라 온프레미스 클라이언트와 DynamoDB 간, DynamoDB와 동일 AWS 리전의 기타 AWS 리소스 간에 전송 중인 데이터를 보호합니다.

주제


- DynamoDB 저장 데이터 암호화
- VPC 엔드포인트 및 IAM 정책을 사용하여 DynamoDB 연결 보호


## DynamoDB 저장 데이터 암호화

Amazon DynamoDB에 저장된 모든 사용자 데이터는 저장 중 완전히 암호화됩니다. DynamoDB 저장 데이터 암호화는 AWS Key Management Service(AWS KMS)에 저장된 암호화 키를 사용하여 모든 저장 중 데이터를 암호화하여 향상된 보안을 제공합니다. 이 기능을 사용하면 중요한 데이터 보호와 관련된 운영 부담 및 복잡성을 줄일 수 있습니다. 저장 시 암호화를 사용하면 엄격한 암호화 규정 준수 및 규제 요구 사항이 필요한, 보안에 민감한 애플리케이션을 구축할 수 있습니다.

DynamoDB 유휴 시 암호화는 내구성이 뛰어난 미디어에 데이가 저장될 때마다 프라이머리 키, 로컬 및 글로벌 보조 인덱스, 스트림, 전역 테이블, 백업, DynamoDB Accelerator(DAX) 클러스터 등을 비롯한 데이터를 항상 암호화된 테이블에서 보호하여 추가 데이터 보호 계층을 제공합니다. 조직의 정책, 업계나 정부 규범 및 규정 준수 요건에 따라 유휴 시 암호화를 사용하여 애플리케이션의 데이터 보안을 강화해야 할 수 있습니다. 데이터베이스 애플리케이션의 암호화에 대한 자세한 내용은 AWS Database Encryption SDK를 참조하세요.

유휴 시 암호화는 테이블을 암호화하는 데 사용되는 암호화 키를 관리하기 위해 AWS KMS와 통합됩니다. 키 유형 및 상태에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS Key Management Service 개념을 참조하세요.

새 테이블을 만들 때 다음 AWS KMS key 유형 중 하나를 선택하여 테이블을 암호화할 수 있습니다. 언제든지 이러한 키 유형 간에 전환할 수 있습니다.

- AWS 소유 키 – 기본 암호화 유형. 키는 DynamoDB가 소유합니다(추가 비용 없음).
- AWS 관리형 키 – 이 키는 사용자의 계정에 저장되고 AWS KMS에 의해 관리됩니다(AWS KMS 비용 적용).
- 고객 관리형 키 - 사용자의 계정에 키가 저장되며 사용자가 생성, 소유, 관리하는 유형입니다. 키에 대해 사용자가 모든 것을 제어합니다(AWS KMS 비용 적용).

키 유형에 대한 자세한 내용은 고객 키 및 AWS 키를 참조하세요.

> **Note**
>
> - 저장 중 데이터 암호화를 활성화하고 새 DAX 클러스터를 생성할 경우 AWS 관리형 키를 사용하여 클러스터에서 저장 중 데이터를 암호화합니다.
> - 테이블에 정렬 키가 있는 경우 범위 경계를 표시하는 정렬 키 중 일부가 테이블 메타데이터에 일반 텍스트 형태로 저장됩니다.

암호화된 테이블에 액세스하면 DynamoDB가 테이블 데이터를 투명하게 해독합니다. 암호화된 테이블을 사용 또는 관리하기 위해 코드나 애플리케이션을 변경할 필요가 없습니다. DynamoDB는 사용자가 기대하는 한 자릿수 밀리초 지연 시간을 계속해서 제공하며 모든 DynamoDB 쿼리가 암호화된 데이터에 대해 원활하게 처리됩니다.

AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 Amazon DynamoDB API를 사용하여 기존 테이블에서 새로운 테이블을 생성하거나 암호화 키를 전환할 때 암호화 키를 지정할 수 있습니다. 자세한 방법은 DynamoDB의 암호화된 테이블 관리 단원을 참조하세요.

AWS 소유 키를 사용한 저장 데이터 암호화는 추가 비용 없이 제공됩니다. 그러나 AWS 관리형 키 및 고객 관리형 키에 대해서는 AWS KMS 비용이 부과됩니다. 요금에 대한 자세한 내용은 AWS KMS 요금을 참조하세요.

DynamoDB 저정 데이터 암호화는 AWS 중국(베이징), AWS 중국(닝샤), AWS GovCloud(미국)를 포함한 모든 AWS 리전에서 이용 가능합니다. 자세한 내용은 저장 시 DynamoDB 암호화: 작동 방식 및 DynamoDB 저장 데이터 암호화 사용 참고 사항(을)를 참조하세요.

## 저장 시 DynamoDB 암호화: 작동 방식

Amazon DynamoDB 저장 데이터 암호화는 256비트 고급 암호화 표준(AES-256)에 따라 데이터를 암호화하여 기본 스토리지에 대한 무단 액세스로부터 데이터를 보호할 수 있도록 지원합니다.

유휴 시 암호화는 테이블을 암호화하는 데 사용되는 암호화 키를 관리하기 위해 AWS Key Management Service(AWS KMS)와 통합됩니다.

> **Note**
>
> 2022년 5월, AWS KMS는 AWS 관리형 키에 대한 교체 일정을 3년(약 1,095일)에서 매년(약 365일)으로 변경했습니다.
> 새 AWS 관리형 키는 생성된 후 1년이 지나면 자동으로 교체되고, 그 후에도 대략 매년 자동으로 교체됩니다.

기존 AWS 관리형 키는 가장 최근 교체 후 1년이 지나면 자동으로 교체되고, 그 후에도 매년 자동으로 교체됩니다.

AWS 소유 키

AWS 소유 키는 AWS 계정에 저장되지 않습니다. 이들은 AWS가 여러 AWS 계정에서 사용하기 위해 소유 및 관리하는 KMS 키 모음의 일부입니다. AWS 서비스는 AWS 소유 키를 사용하여 데이터를 보호할 수 있습니다. DynamoDB에서 사용하는 AWS 소유 키는 매년(약 365일) 교체됩니다.

사용자는 AWS 소유 키를 확인, 관리 또는 사용하거나 사용을 감사할 수 없습니다. 하지만 데이터를 암호화하는 키를 보호하기 위해 어떤 작업을 수행하거나 어떤 프로그램을 변경할 필요가 없습니다.

월별 요금 또는 AWS 소유 키의 사용량에 따른 요금이 부과되지 않으며, 계정의 AWS KMS 할당량에 포함되지 않습니다.

AWS 관리형 키

AWS 관리형 키는 AWS KMS와 통합된 AWS 서비스가 고객의 계정에서 고객 대신 생성, 관리 및 사용하는 KMS 키입니다. 계정에서 AWS 관리형 키를 확인하고 키 정책을 확인하며, AWS CloudTrail 로그에서 사용을 감사할 수 있습니다. 하지만 이러한 KMS 키를 관리하거나 이들의 권한을 변경할 수는 없습니다.

유휴 시 암호화는 테이블을 암호화하는 데 사용되는 DynamoDB(aws/dynamodb)용 AWS 관리형 키를 관리하기 위해 AWS KMS와 자동으로 통합됩니다. 암호화된 DynamoDB 테이블을 생성할 때 AWS 관리형 키가 존재하지 않는 경우 AWS KMS에서 새 키를 자동으로 생성합니다. 이 키는 향후 생성되는 암호화된 테이블에 사용됩니다. AWS KMS는 클라우드에 맞게 조정된 키 관리 시스템을 제공하기 위해 안전하고 가용성이 높은 하드웨어 및 소프트웨어를 결합합니다.

AWS 관리형 키 권한 관리에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS 관리형 키 사용 권한 부여를 참조하세요.

고객 관리형 키

고객 관리형 키는 사용자가 생성, 소유 및 관리하는 AWS 계정의 KMS 키입니다. 사용자는 키 정책 설정 및 관리, IAM 정책, 권한 부여, 활성화 및 비활성화, 암호화 구성 요소 교체, 태그 추가, CMK를 가리키는 별칭 생성 및 삭제를 위한 KMS 예약 등을 포함해 KMS 키에 대한 완전한 제어 권한을 가집니다. 고객 관리형 키의 권한 관리에 대한 자세한 내용은 고객 관리형 키를 참조하세요.

고객 관리형 키를 테이블 수준 암호화 키로 지정하면 DynamoDB 테이블, 로컬 및 글로벌 보조 인덱스, 스트림은 동일한 고객 관리형 키로 암호화됩니다. 온디맨드 백업은 백업이 생성되는 당시에 지정된 테

이블 수준 암호화 키로 암호화됩니다. 테이블 수준 암호화 키를 업데이트하더라도 기존 온디맨드 백업과 연계된 암호화 키는 변경되지 않습니다.

고객 관리형 키의 상태를 비활성화로 설정하거나 삭제를 예약하면 모든 사용자와 DynamoDB 서비스가 데이터를 암호화 또는 암호 해독하고 테이블에서 읽고 쓰는 작업을 수행할 수 없습니다. DynamoDB는 사용자가 테이블에 지속적으로 액세스하고 데이터 유실을 방지하기 위해 암호화 키에 액세스할 수 있어야 합니다.

고객 관리형 키를 비활성화하거나 삭제를 예약한 경우에는 테이블 상태가 Inaccessible(접근 불가)로 바뀌게 됩니다. 테이블에서 작업을 계속하기 위해서는 7일 안에 DynamoDB 액세스를 지정된 암호화 키에 제공해야 합니다. 서비스에서 사용자의 암호화 키가 접근 불가 상태인 것을 탐지하면 바로 DynamoDB는 사용자에게 이를 알리기 위해 이메일 공지를 보냅니다.

> **Note**
>
> - 7일 이상 고객 관리형 키가 DynamoDB 서비스에 접근 불가 상태가 되면 테이블은 아카이브 상태로 넘어가며 더 이상 액세스할 수 없게 됩니다. DynamoDB는 사용자의 테이블에 대해 온디맨드 백업을 생성하며 이에 대한 비용이 청구됩니다. 이 온디맨드 백업을 새로운 테이블에 데이터를 복원할 때 사용할 수 있습니다. 복원을 시작하려면 테이블에 대한 최종 고객 관리형 키가 활성화되어야 하며, DynamoDB는 이에 액세스할 수 있어야 합니다.
> - 글로벌 테이블 복제본을 암호화하는 데 사용된 고객 관리형 키에 액세스할 수 없는 경우 DynamoDB는 복제 그룹에서 이 복제본을 제거합니다. 고객 관리형 키를 액세스할 수 없는 것으로 감지하고 20시간 후 복제본이 삭제되지 않고 이 리전에서와 이 리전으로의 복제가 중지됩니다.

자세한 내용은 키 활성화 및 키 삭제를 참조하세요.

AWS 관리형 키 사용 시 참고 사항

Amazon DynamoDB는 AWS KMS 계정에 저장된 KMS 키에 대한 액세스 권한이 있어야 테이블 데이터를 읽을 수 있습니다. DynamoDB는 봉투 암호화 및 키 계층 구조를 사용하여 데이터를 암호화합니다. AWS KMS 암호화 키는 이 키 계층 구조의 루트 키를 암호화하는 데 사용됩니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 봉투 암호화를 참조하세요.

DynamoDB는 모든 DynamoDB 작업에 대해 AWS KMS를 호출하지 않습니다. 활성 트래픽을 통한 호출자당 5분마다 키가 새로 고침됩니다.

연결을 재사용하도록 SDK를 구성했는지 확인합니다. 그렇게 되어 있지 않다면 각 DynamoDB 작업에 대해 새 AWS KMS 캐시 항목을 다시 설정해야 하므로 DynamoDB에서 지연 시간이 발생합니다. 뿐만 아니라 더 많은 AWS KMS 및 CloudTrail 비용이 발생할 수 있습니다. 예를 들어 Node.js SDK를 사용하여 이 작업을 수행하려면 `keepAlive`를 켠 상태에서 새 HTTPS 에이전트를 생성합니다. 자세한 내용은 AWS SDK for JavaScript 개발자 안내서에서 Node.js에서 keepAlive 구성을 참조하세요.

## DynamoDB 저장 데이터 암호화 사용 참고 사항

Amazon DynamoDB에서 저장 데이터 암호화를 사용할 때 다음을 고려하세요.

모든 테이블 데이터는 암호화됩니다.

서버 측 저장 데이터 암호화는 모든 DynamoDB 테이블에서 활성화되며 비활성화할 수 없습니다. 테이블에서 항목의 하위 집합만 암호화할 수 없습니다.

저장 시 암호화는 영구 스토리지 미디어에서 정적(저장 시) 상태인 데이터만 암호화합니다. 전송 중인 데이터 또는 사용 중인 데이터에 대한 데이터 보안 문제가 있는 경우 추가 조치를 취해야 합니다.

- 전송 중 데이터: DynamoDB의 모든 데이터는 전송 중에 암호화됩니다. 기본적으로 DynamoDB와의 통신은 보안 소켓 계층(SSL)/전송 계층 보안(TLS) 암호화를 사용하여 네트워크 트래픽을 보호하는 HTTPS 프로토콜을 사용합니다.
- 사용 중인 데이터: DynamoDB에 보내기 전에 클라이언트 측 암호화를 사용하여 데이터를 보호합니다. 자세한 내용은 Amazon DynamoDB Encryption Client 개발자 안내서의 클라이언트 측 및 서버 측 암호화를 참조하세요.

암호화된 테이블에서 스트림을 사용할 수 있습니다. DynamoDB 스트림은 항상 테이블 수준 암호화 키로 암호화됩니다. 자세한 내용은 DynamoDB Streams에 대한 변경 데이터 캡처 섹션을 참조하세요.

DynamoDB 백업은 암호화되고 이 백업에서 복원되는 테이블도 암호화가 활성화되어 있습니다. 백업 데이터를 암호화하기 위해 AWS 소유 키, AWS 관리형 키 또는 고객 관리형 키를 사용할 수 있습니다. 자세한 내용은 DynamoDB에 대한 백업 및 복원 섹션을 참조하세요.

로컬 보조 인덱스와 글로벌 보조 인덱스는 베이스 테이블과 동일한 키를 사용해 암호화됩니다.

### 암호화 유형

> **Note**
>
> 고객 관리형 키는 글로벌 테이블 버전 2017에서 지원되지 않습니다. DynamoDB 글로벌 테이블에서 고객 관리형 키를 사용하려면 테이블을 글로벌 테이블 버전 2019로 업그레이드하고 사용 설정해야 합니다.

AWS Management Console에서 AWS 관리형 키 또는 고객 관리형 키를 사용하여 데이터를 암호화할 때 암호화 유형은 `KMS`입니다. AWS 소유 키를 사용할 경우 암호화 유형은 `DEFAULT`입니다. Amazon DynamoDB API에서 AWS 관리형 키 또는 고객 관리형 키를 사용하여 데이터를 암호화할 때 암호화 유형은 `KMS`입니다. 암호화 유형이 없을 경우 AWS 소유 키를 사용하여 데이터를 암호화합니다. 언제든지 AWS 소유 키, AWS 관리형 키 및 고객 관리형 키 간에 전환할 수 있습니다. 콘솔, AWS Command Line Interface(AWS CLI) 또는 Amazon DynamoDB API를 사용해 암호화 키를 전환할 수 있습니다.

고객 관리형 키를 사용할 때 다음과 같은 제한을 유의하세요.

- DynamoDB Accelerator(DAX) 클러스터에 고객 관리형 키를 사용할 수 없습니다. 자세한 내용은 DAX 저장 데이터 암호화 섹션을 참조하세요.
- 트랜잭션을 사용하는 테이블을 암호화하기 위해 고객 관리형 키를 사용할 수 없습니다. 그러나 트랜잭션 전파의 내구성을 보장하기 위해 트랜잭션 요청의 복사본이 서비스에 의해 임시로 저장되고 AWS 소유 키를 사용하여 암호화됩니다. 테이블 및 보조 인덱스에서 커밋된 데이터는 항상 고객 관리형 키를 사용하여 저장 중 암호화됩니다.
- Contributor Insights를 사용하는 테이블을 암호화하기 위해 고객 관리형 키를 사용할 수 있습니다. 그러나 Amazon CloudWatch로 전송되는 데이터는 AWS 소유 키로 암호화됩니다.
- 새 고객 관리형 키로 전환하는 경우 프로세스가 완료될 때까지 원래 키를 사용하도록 설정해야 합니다. AWS에서 새 키로 암호화하기 전에 데이터를 해독하기 위해 원래 키가 필요합니다. 테이블의 SSEDescription 상태가 활성화되고 새 고객 관리형 키의 KMSMasterKeyArn이 표시되면 프로세스가 완료됩니다. 이 시점에서 원래 키를 비활성화하거나 삭제하도록 예약할 수 있습니다.
- 새 고객 관리 키가 표시되면 테이블과 모든 새 온디맨드 백업이 새 키로 암호화됩니다.
- 기존 온디맨드 백업은 백업이 생성될 때 사용된 고객 관리 키로 암호화된 상태로 유지됩니다. 이러한 백업을 복원하려면 동일한 키가 필요합니다. DescripbeBackup API를 사용하여 해당 백업의 SSEDescription을 확인하여 각 백업이 생성된 기간의 키를 식별할 수 있습니다.
- 고객 관리형 키를 비활성화하거나 삭제를 예약하는 경우 DynamoDB Streams에 있는 모든 데이터는 24시간 수명이 적용됩니다. 24시간이 지날 경우 모든 회수되지 않은 데이터는 트리밍 권한이 주어집니다.

- 고객 관리형 키를 비활성화하거나 삭제를 예약한 경우에는 유지 시간(TTL) 삭제가 30분간 진행됩니다. 이러한 TTL 삭제는 DynamoDB Streams로 계속 방출되며 표준 트리밍/보존 간격이 적용됩니다.

  자세한 내용은 키 활성화 및 키 삭제를 참조하세요.

KMS 키 및 데이터 키 사용

DynamoDB 암호화 기능은 AWS KMS key와 데이터 키 계층을 사용하여 테이블 데이터를 보호합니다. DynamoDB는 동일한 키 계층을 사용하여 내구성 있는 미디어에 기록되는 DynamoDB 스트림, 전역 테이블 및 백업을 보호합니다.

DynamoDB에서 테이블을 구현하기 전에 암호화 전략을 계획하는 것이 좋습니다. 민감한 데이터나 기밀 데이터를 DynamoDB에 저장하는 경우 계획에 클라이언트측 암호화를 포함하는 것을 고려해 보세요. 이렇게 하면 데이터를 원본에 최대한 가깝게 암호화하고 전체 수명 주기 동안 데이터를 보호할 수 있습니다. 자세한 내용은 DynamoDB 암호화 클라이언트 설명서를 참조하세요.

AWS KMS key

유휴 시 암호화는 AWS KMS key에서 DynamoDB 테이블을 보호합니다. 기본적으로 DynamoDB는 DynamoDB 서비스 계정에서 생성 및 관리되는 다중 테넌트 암호화 키인 AWS 소유 키를 사용합니다. 그러나 AWS 계정의 DynamoDB(`aws/dynamodb`)용 고객 관리형 키에서 DynamoDB 테이블을 암호화할 수 있습니다. 각 테이블마다 다른 KMS 키를 선택할 수 있습니다. 테이블에 대해 선택한 KMS 키는 로컬 및 전역 보조 인덱스, 스트림 및 백업을 암호화하는 데에도 사용됩니다.

테이블을 생성 또는 업데이트할 때 테이블에 대해 KMS 키를 선택합니다. 언제든지 DynamoDB 콘솔에서 또는 UpdateTable 작업을 사용하여 테이블에 대한 KMS 키를 변경할 수 있습니다. 키 전환 프로세스는 원활하며 가동 중지 시간 또는 서비스 저하가 발생하지 않습니다.

> **Important**
>
> DynamoDB는 대칭 KMS 키만 지원합니다. 비대칭 KMS 키를 사용하여 DynamoDB 테이블을 암호화할 수 없습니다.

고객 관리형 키를 사용하여 다음과 같은 기능을 얻을 수 있습니다.

- KMS 키에 대한 액세스를 제어하기 위한 키 정책, IAM 정책 및 권한 부여 설정을 포함하여 KMS 키를 생성하고 관리합니다 KMS 키를 활성화 및 비활성화하고, 자동 키 교체를 활성화 및 비활성화하고, KMS 키가 더 이상 사용되지 않을 때 KMS 키를 삭제할 수 있습니다.

- 가져온 키 구성 요소가 있는 고객 관리형 키를 사용하거나 고객이 소유하고 관리하는 사용자 지정 키 스토어에서 고객 관리형 키를 사용할 수 있습니다.
- AWS CloudTrail 로그에서 AWS KMS에 대한 DynamoDB API 호출을 검사하여 DynamoDB 테이블의 암호화 및 복호화를 감사할 수 있습니다.

다음과 같은 기능이 필요한 경우 AWS 관리형 키를 사용합니다.

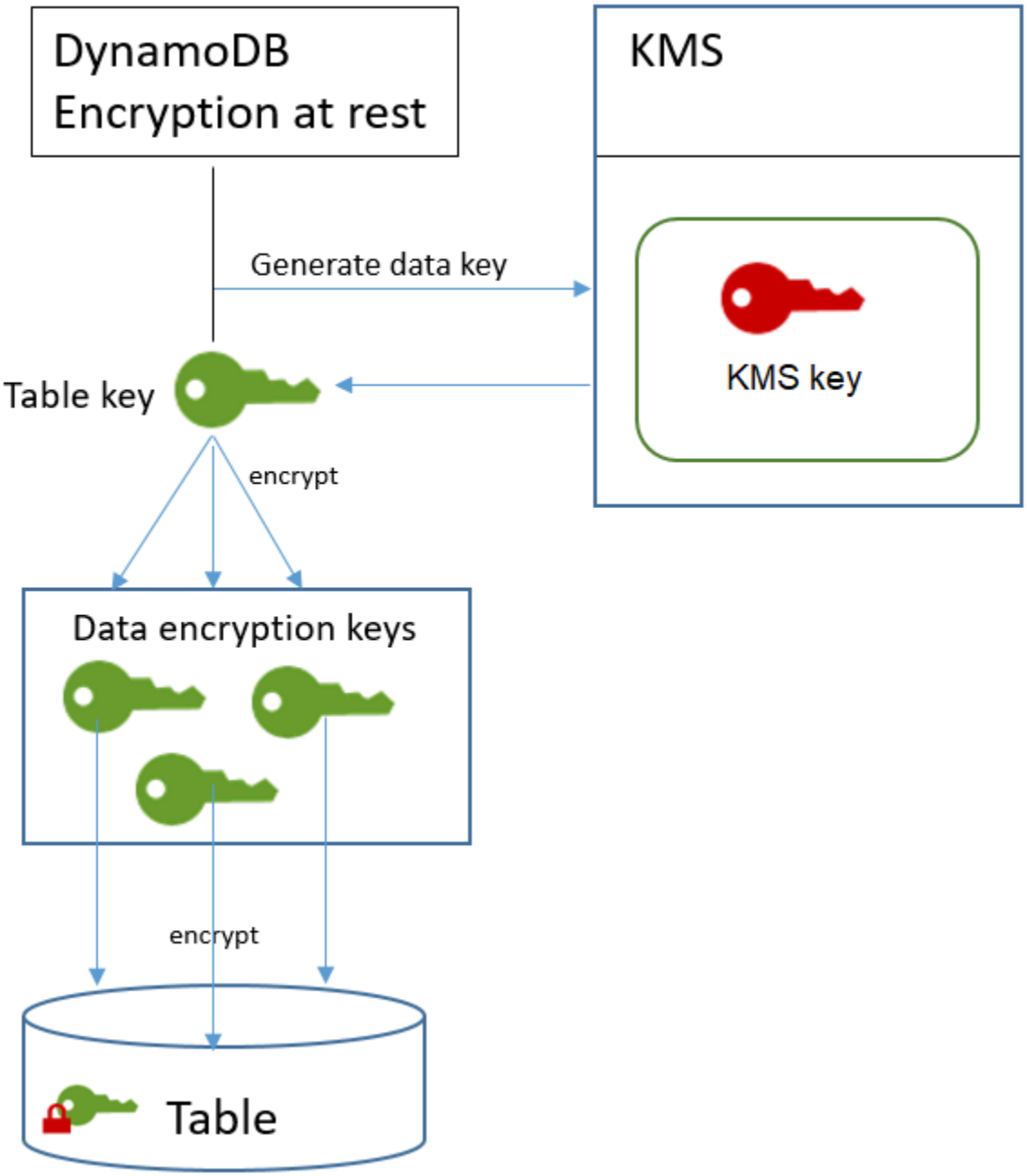
- KMS 키 및 해당 키 정책을 볼 수 있습니다. 키 정책을 변경할 수 없습니다.
- AWS CloudTrail 로그에서 AWS KMS에 대한 DynamoDB API 호출을 검사하여 DynamoDB 테이블의 암호화 및 복호화를 감사할 수 있습니다.

그러나 AWS 소유 키는 무료이며 그 사용은 AWS KMS 리소스 또는 요청 할당량에 포함되지 않습니다. 고객 관리형 키 및 AWS 관리형 키는 각 API 호출에 대해 요금이 부과되며 이러한 KMS 키에 AWS KMS 할당량이 적용됩니다.

테이블 키

DynamoDB는 테이블에 대해 KMS 키를 사용하여 테이블 키라고 하는 테이블에 대한 고유한 데이터 키를 생성하고 암호화합니다. 테이블 키는 암호화된 테이블의 수명 기간 동안 존속합니다.

테이블 키는 키 암호화 키로 사용됩니다. DynamoDB는 이 테이블 키를 사용하여 테이블 데이터를 암호화하는 데 사용되는 데이터 암호화 키를 보호합니다. DynamoDB는 테이블에 있는 각 기본 구조의 고유한 데이터 암호화 키를 생성하지만, 동일한 데이터 암호화 키로 여러 테이블 항목을 보호할 수 없습니다.




암호화된 테이블에 처음 액세스하면 DynamoDB가 KMS 키를 사용하여 테이블 키를 해제하도록 AWS KMS에 요청합니다. 그런 다음 일반 텍스트 테이블 키를 사용하여 데이터 암호화 키를 해독하고, 일반 텍스트 데이터 암호화 키를 사용하여 테이블 데이터를 해독합니다.

DynamoDB는 AWS KMS 외부에 테이블 키와 데이터 암호화 키를 저장하고 사용합니다. 고급 암호화 표준(AES) 암호화 및 256비트 암호화 키로 모든 키를 보호합니다. 그런 다음 필요에 따라 테이블 데이터를 해독할 때 사용할 수 있도록 암호화된 데이터로 암호화된 키를 저장합니다.

테이블의 KMS 키를 변경하면 DynamoDB가 새 테이블 키를 생성합니다. 그런 다음 새 테이블 키를 사용하여 데이터 암호화 키를 다시 암호화합니다.

테이블 키 캐싱

각 DynamoDB 작업마다 AWS KMS 를 호출하지 않도록 DynamoDB는 메모리에 있는 각 호출자의 일반 텍스트 테이블 키를 캐싱합니다. 5분의 비활성 시간이 경과하여 DynamoDB가 캐싱된 테이블 키를 요청할 경우 테이블 키를 해독하라는 새로운 요청을 AWS KMS에 보냅니다. 이 호출은 마지막 테이블 키 해제 요청 이후 KMS 키의 액세스 정책에 적용된 모든 변경 사항을 AWS KMS 또는 AWS Identity and Access Management(IAM)에 캡처합니다.

### KMS 키 사용 권한 부여

계정에서 고객 관리형 키 또는 AWS 관리형 키를 사용하여 DynamoDB 테이블을 보호할 경우 해당 KMS 키에 대한 정책이 DynamoDB에 사용자 대신 키를 사용할 권한을 부여해야 합니다. DynamoDB용 AWS 관리형 키의 권한 부여 컨텍스트에는 그 사용 권한을 위임하는 키 정책과 권한 부여가 포함되어 있습니다.

고객 관리형 키에 대한 정책 및 권한 부여를 완전히 제어할 수 있습니다. AWS 관리형 키는 계정에 있으므로 해당 정책 및 권한 부여를 볼 수 있습니다. 하지만 정책은 AWS에 의해 관리되므로 사용자가 정책을 변경할 수는 없습니다.

DynamoDB는 기본 AWS 소유 키를 사용하여 AWS 계정의 DynamoDB 테이블을 보호하기 위해 추가 인증이 필요하지 않습니다.

주제

- AWS 관리형 키에 대한 키 정책
- 고객 관리형 키에 대한 키 정책
- 권한 부여를 사용하여 DynamoDB 승인

### AWS 관리형 키에 대한 키 정책

DynamoDB가 암호화 작업에서 DynamoDB(`aws/dynamodb`)에 대해 AWS 관리형 키를 사용하는 경우 DynamoDB 리소스에 액세스하는 사용자를 수행합니다. AWS 관리형 키에 대한 키 정책은 계정의 모든 사용자에게 지정된 작업에 대해 AWS 관리형 키를 사용할 권한을 부여합니다. 그러나 DynamoDB가 사용자 대신 요청할 때만 권한이 부여됩니다. 키 정책의 ViaService 조건은 요청이 DynamoDB 서비스에서 이루어지지 않은 경우 어떤 사용자도 AWS 관리형 키를 사용하도록 허용하지 않습니다.

이 키 정책은 모든 AWS 관리형 키의 정책과 마찬가지로 AWS에 의해 설정됩니다. 변경은 할 수 없지만 언제든 보는 것은 가능합니다. 자세한 내용은 키 정책 보기를 참조하세요.

키 정책의 정책 설명문은 다음 효과를 갖습니다.

- 계정 내 사용자가 DynamoDB에서 대신 요청이 올 때만 암호화 작업에서 DynamoDB에 AWS 관리형 키를 사용할 수 있도록 합니다. 이 정책은 사용자들이 KMS 키의 권한 부여를 생성하는 것도 허용합니다.
- 계정의 승인된 IAM 자격 증명이 DynamoDB용 AWS 관리형 키의 속성을 보고 DynamoDB가 KMS 키를 사용하도록 허용하는 권한을 취소할 수 있습니다. DynamoDB는 지속적인 유지 관리 작업에 권한 부여를 사용합니다.
- DynamoDB가 읽기 전용 작업을 수행하여 계정의 DynamoDB용 AWS 관리형 키를 찾을 수 있게 합니다.

JSON

```
{
  "Version" : "2012-10-17",
  "Id" : "auto-dynamodb-1",
  "Statement" : [ {
    "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
    "Action" : [ "kms:Encrypt", "kms:Decrypt", "kms:ReEncrypt*",
 "kms:GenerateDataKey*", "kms:CreateGrant", "kms:DescribeKey" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:CallerAccount" : "111122223333",
        "kms:ViaService" : "dynamodb.us-west-2.amazonaws.com"
      }
    }
  }, {
    "Sid" : "Allow direct access to key metadata to the account",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
```

```
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*", "kms:RevokeGrant" ],
    "Resource" : "*"
  }, {
    "Sid" : "Allow DynamoDB Service with service principal name
 dynamodb.amazonaws.com to describe the key directly",
    "Effect" : "Allow",
    "Principal" : {
      "Service" : "dynamodb.amazonaws.com"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*" ],
    "Resource" : "*"
  } ]
}
```

고객 관리형 키에 대한 키 정책

DynamoDB 테이블을 보호하기 위해 고객 관리형 키를 선택하면 DynamoDB는 선택하는 보안 주체를 대신하여 KMS 키를 사용할 수 있는 권한을 얻습니다. 해당 보안 주체, 즉 사용자 또는 역할은 DynamoDB가 필요한 KMS 키에 대한 권한이 있어야 합니다. 키 정책, IAM 정책 또는 권한 부여에 이러한 권한을 제공할 수 있습니다.

DynamoDB는 고객 관리형 키에 대해 최소한 다음 권한이 있어야 합니다.

- kms:Encrypt
- kms:Decrypt
- kms:ReEncrypt*(kms:ReEncryptFrom and kms:ReEncryptTo용)
- kms:GenerateDataKey*(kms:GenerateDataKey 및 kms:GenerateDataKeyWithoutPlaintext용)
- kms:DescribeKey
- kms:CreateGrant

예를 들어 다음 예제 키 정책은 필수 권한만 제공합니다. 이 정책에는 다음과 같은 효과가 있습니다.

- DynamoDB가 암호화 작업에 KMS 키를 사용하고 권한 부여를 생성하도록 허용합니다. 단, DynamoDB를 사용할 권한이 있는 계정의 보안 주체를 대신해 동작하는 경우에 한합니다. 정책 문에 지정된 보안 주체가 DynamoDB를 사용할 권한이 없는 경우 DynamoDB 서비스에서 오는 경우에도 호출이 실패합니다.

- kms:ViaService 조건 키는 정책 문에 나열된 보안 주체를 대신하여 DynamoDB로부터 요청이 오는 경우에만 사용 권한을 허용합니다. 이러한 보안 주체는 이러한 작업을 직접 호출 할 수 없습니다. 참고: `kms:ViaService` 값(`dynamodb.*.amazonaws.com`)은 리전 위치에 별표(*)가 있습니다. DynamoDB는 DynamoDB 전역 테이블을 지원하기 위해 교차 리전 호출을 수행할 수 있도록 특정 AWS 리전과 독립적인 권한이 필요합니다.
- KMS 키 관리자(`db-team` 역할을 수임할 수 있는 사용자)에게 KMS 키에 대한 읽기 전용 액세스와 테이블을 보호하기 위해 DynamoDB가 필요로 하는 권한 부여를 포함하여 권한 부여를 취소할 수 있는 권한을 제공합니다.

예제 키 정책을 사용하기 전에 예제 보안 주체를 AWS 계정의 실제 보안 주체로 바꿉니다.

JSON

```
{
  "Id": "key-policy-dynamodb",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/db-lead"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
         "StringLike": {
           "kms:ViaService" : "dynamodb.*.amazonaws.com"
         }
      }
    },
    {
      "Sid":  "Allow administrators to view the KMS key and revoke grants",
      "Effect": "Allow",
```

```
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/db-team"
       },
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    }
  ]
}
```

권한 부여를 사용하여 DynamoDB 승인

키 정책 이외에 DynamoDB는 권한 부여를 사용하여 DynamoDB용 AWS 관리형 키(aws/dynamodb) 또는 고객 관리형 키에 대한 권한을 설정합니다. 계정에서 KMS 키에 대한 권한 부여를 보려면 ListGrants 작업을 사용합니다. DynamoDB는 AWS 소유 키를 사용하여 테이블을 보호하는 데 권한 부여 또는 추가 권한이 필요하지 않습니다.

DynamoDB는 배경 시스템 유지 관리 및 연속 데이터 보호 태스크를 수행할 때 권한 부여 권한을 사용합니다. 권한 부여를 사용하여 테이블 키도 생성합니다.

각 권한 부여는 테이블마다 다릅니다. 계정에 동일한 KMS 키로 암호화되는 여러 테이블이 있는 경우 각 테이블 유형별 권한 부여가 있습니다. 권한 부여는 DynamoDB 암호화 컨텍스트의 제한을 받는데, 여기에는 테이블 이름과 AWS 계정 ID가 포함되며, 더 이상 필요 없을 경우 권한 부여 제거 권한도 포함됩니다.

권한 부여를 생성하려면 DynamoDB가 암호화된 테이블을 생성한 사용자를 대신하여 `CreateGrant`를 호출할 수 있는 권한이 있어야 합니다. AWS 관리형 키의 경우 DynamoDB 는 키 정책에서 `kms:CreateGrant` 권한을 얻습니다. 이를 통해 계정 사용자는 DynamoDB가 승인된 사용자를 대신하여 요청할 때만 KMS 키에서 CreateGrant를 호출할 수 있습니다.

이 키 정책은 계정이 KMS 키에 대한 권한 부여를 취소하도록 허용할 수도 있습니다. 단, 활성 암호화 테이블에서 권한 부여를 취소할 경우에는 DynamoDB가 테이블을 보호하고 유지하지 못합니다.

## DynamoDB 암호화 컨텍스트

암호화 컨텍스트는 보안되지 않은 임의의 데이터를 포함하는 키-값 페어 세트입니다. 데이터 암호화 요청에 암호화 컨텍스트를 포함하는 경우 AWS KMS는 암호화된 데이터에 암호화 컨텍스트를 암호 방식으로 바인딩합니다. 따라서 동일한 암호화 컨텍스트로 전달해야 이 데이터를 해독할 수 있습니다.

DynamoDB는 모든 AWS KMS 암호화 작업에서 동일한 암호화 컨텍스트를 사용합니다. 고객 관리형 키 또는 AWS 관리형 키 를 사용하여 DynamoDB 테이블을 보호할 경우 암호화 컨텍스트를 사용하여 감사 레코드 및 로그에서 KMS 키의 사용을 식별할 수 있습니다. AWS CloudTrail 및 Amazon CloudWatch Logs 같은 로그에서 일반 텍스트에 나타나기도 합니다.

암호화 컨텍스트를 정책 및 권한 부여에서 승인 조건으로 사용할 수도 있습니다. DynamoDB는 암호화 컨텍스트를 사용하여 계정 및 리전에서 고객 관리형 키 또는 AWS 관리형 키에 대한 액세스를 허용하는 권한을 제한합니다.

AWS KMS에 요청할 때 DynamoDB 는 두 개의 키값 쌍이 있는 암호화 컨텍스트를 사용합니다.

```
"encryptionContextSubset": {
    "aws:dynamodb:tableName": "Books"
    "aws:dynamodb:subscriberId": "111122223333"
}
```

- 테이블 – 첫 번째 키값 쌍은 DynamoDB가 암호화 중인 테이블을 나타냅니다. 키는 `aws:dynamodb:tableName`입니다. 값은 테이블의 이름입니다.

  ```
  "aws:dynamodb:tableName": "<table-name>"
  ```

  예:

  ```
  "aws:dynamodb:tableName": "Books"
  ```

- 계정 – 두 번째 키값 쌍은 AWS 계정을 나타냅니다. 키는 `aws:dynamodb:subscriberId`입니다. 값은 계정 ID입니다.

  ```
  "aws:dynamodb:subscriberId": "<account-id>"
  ```

  예:

  ```
  "aws:dynamodb:subscriberId": "111122223333"
  ```

### AWS KMS와 DynamoDB 상호 작용 모니터링

고객 관리형 키 또는 AWS 관리형 키를 사용하여 DynamoDB 테이블을 보호할 경우 AWS CloudTrail 로그를 사용하여 DynamoDB가 사용자 대신 AWS KMS로 전송하는 요청을 추적할 수 있습니다.

`GenerateDataKey`, `Decrypt` 및 `CreateGrant` 요청은 이 섹션에서 다룹니다. 그리고 DynamoDB는 DescribeKey 작업을 사용하여 사용자가 선택한 KMS 키가 계정과 리전에 있는지 확인합니다. 그리고 RetireGrant 작업을 사용하여 사용자가 테이블을 삭제할 때 권한 부여를 제거합니다.

#### GenerateDataKey

사용자가 테이블에서 영구 암호화를 활성화하면 DynamoDB가 고유한 테이블 키를 생성합니다. 테이블에 대한 KMS 키를 지정하는 GenerateDataKey 요청을 AWS KMS로 보냅니다.

`GenerateDataKey` 작업을 기록하는 이벤트는 다음 예시 이벤트와 유사합니다. 사용자는 DynamoDB 서비스 계정입니다. 파라미터에는 KMS 키의 Amazon 리소스 이름(ARN), 256비트 키가 필요한 키 지정자, 테이블과 AWS 계정를 식별하는 암호화 컨텍스트가 포함됩니다.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:17Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:dynamodb:tableName": "Services",
            "aws:dynamodb:subscriberId": "111122223333"
        },
        "keySpec": "AES_256",
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "229386c1-111c-11e8-9e21-c11ed5a52190",
    "eventID": "e3c436e9-ebca-494e-9457-8123a1f5e979",
    "readOnly": true,
```

```
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "sharedEventID": "bf915fa6-6ceb-4659-8912-e36b69846aad"
}
```

### Decrypt

암호화된 DynamoDB 테이블에 액세스하면 DynamoDB가 테이블 키를 해독해야만 계층에서 그 아래에 있는 키를 해독할 수 있습니다. 이후 테이블에 있는 데이터를 해독합니다. 테이블 키를 복호화하려면 DynamoDB는 테이블에 대한 KMS 키를 지정하는 AWS KMS로 Decrypt 요청을 보냅니다.

`Decrypt` 작업을 기록하는 이벤트는 다음 예시 이벤트와 유사합니다. 사용자는 테이블에 액세스 중인 AWS 계정에 있는 보안 주체입니다. 파라미터에는 암호화된 테이블 키(암호화 텍스트 blob)와, 테이블 및 AWS 계정를 식별하는 암호화 컨텍스트가 포함됩니다. AWS KMS는 암호화 텍스트에서 KMS 키의 ID를 파생합니다.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T16:42:15Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDT3HGFQZX4RY6RU",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
```

```
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T16:42:39Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters":
    {
        "encryptionContext":
        {
            "aws:dynamodb:tableName": "Books",
            "aws:dynamodb:subscriberId": "111122223333"
        }
    },
    "responseElements": null,
    "requestID": "11cab293-11a6-11e8-8386-13160d3e5db5",
    "eventID": "b7d16574-e887-4b5b-a064-bf92f8ec9ad3",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


### CreateGrant

고객 관리형 키 또는 AWS 관리형 키를 사용하여 DynamoDB 테이블을 보호하면 DynamoDB는 권한 부여를 사용하여 서비스가 지속적 데이터 보호와 유지 관리 및 내구성 태스크를 수행하도록 허용합니다. 이러한 권한 부여가 AWS 소유 키에서는 필요하지 않습니다.

DynamoDB가 생성하는 권한 부여는 테이블마다 다릅니다. CreateGrant 요청에 있는 주체는 테이블을 생성한 사용자입니다.

`CreateGrant` 작업을 기록하는 이벤트는 다음 예시 이벤트와 유사합니다. 파라미터에는 테이블에 대한 KMS 키의 Amazon 리소스 이름(ARN), 피부여자 주체 및 삭제 주체(DynamoDB 서비스), 권한 부여가 적용되는 작업이 포함됩니다. 모든 암호화 작업에서 지정된 암호화 컨텍스트를 사용할 것을 요구하는 제한도 포함됩니다.

```
{
    "eventVersion": "1.05",
    "userIdentity":
    {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T00:12:02Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:15Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "retiringPrincipal": "dynamodb.us-west-2.amazonaws.com",
        "constraints": {
            "encryptionContextSubset": {
                "aws:dynamodb:tableName": "Books",
                "aws:dynamodb:subscriberId": "111122223333"
            }
```

```
        },
        "granteePrincipal": "dynamodb.us-west-2.amazonaws.com",
        "operations": [
            "DescribeKey",
            "GenerateDataKey",
            "Decrypt",
            "Encrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "RetireGrant"
        ]
    },
    "responseElements": {
        "grantId":
 "5c5cd4a3d68e65e77795f5ccc2516dff057308172b0cd107c85b5215c6e48bde"
    },
    "requestID": "2192b82a-111c-11e8-a528-f398979205d8",
    "eventID": "a03d65c3-9fee-4111-9816-8bf96b73df01",
    "readOnly": false,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


## DynamoDB의 암호화된 테이블 관리

AWS Management Console 또는 AWS Command Line Interface(AWS CLI)를 사용해 새로운 테이블에 암호화 키를 지정하고 Amazon DynamoDB에 있는 기존 테이블의 암호화 키를 업데이트할 수 있습니다.

주제

- 새 테이블에 대한 암호화 키 지정
- 암호화 키 업데이트

### 새 테이블에 대한 암호화 키 지정

Amazon DynamoDB 콘솔 또는 AWS CLI를 이용해 새로운 테이블에 암호화 키를 지정하기 위해 다음 단계를 따르세요.

#### 암호화된 테이블 생성(콘솔)

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 테이블(Tables)을 선택합니다.
3. Create Table(테이블 생성)을 선택합니다. 테이블 이름에 **`Music`**을 입력합니다. 기본 키의 경우 **`Artist`**를, 정렬 키의 경우 **`SongTitle`**을 각각 문자열로 입력합니다.
4. 설정(Settings)에서 설정 사용자 지정(Customize settings)이 선택되었는지 확인합니다.

   > **Note**
   >
   > 기본 설정 사용을 선택한 경우 AWS 소유 키를 사용하여 무료로 테이블이 저장 중 암호화됩니다.

5. Encryption at rest(유휴 시 암호화)에서 암호화 유형 -AWS 소유 키, AWS 관리형 키, 또는 고객 관리 키를 선택합니다.

   - Owned by Amazon DynamoDB.(Amazon DynamoDB가 소유.) AWS 소유 키로, 구체적으로는 DynamoDB가 소유하고 관리합니다. 이 키 사용에 대한 추가 요금은 부과되지 않습니다.
   - AWS 관리형 키. 키 별칭: `aws/dynamodb`. 이 키는 계정에 저장되고 AWS Key Management Service에서 관리합니다(AWS KMS). AWS KMS 비용이 적용됩니다.
   - 계정에 저장되며 본인이 소유하고 관리합니다. 고객 관리형 키. 이 키는 계정에 저장되고 AWS Key Management Service에서 관리합니다(AWS KMS). AWS KMS 비용이 적용됩니다.

   > **Note**
   >
   > 자체 키 소유 및 관리를 선택하는 경우 KMS 키 정책이 적절하게 설정되었는지 확인합니다. 예를 포함한 자세한 내용은 고객 관리형 키에 대한 키 정책을 참조하세요.

6. 테이블 생성(Create table)을 선택하여 암호화된 테이블을 생성합니다. 암호화 유형을 확인하려면 개요(Overview) 탭의 테이블 세부 정보를 클릭하고 추가 세부 정보(Additional details) 섹션을 검토합니다.

### 암호화된 테이블 생성(AWS CLI)

AWS CLI를 사용해 Amazon DynamoDB용 기본 AWS 소유 키, AWS 관리형 키 또는 고객 관리형 키가 있는 테이블을 생성합니다.

기본 AWS 소유 키를 사용하여 암호화된 테이블을 생성하려면

- 다음과 같이 암호화된 `Music` 테이블을 생성합니다.

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5
```

> **Note**
>
> 이 테이블은 이제 DynamoDB 서비스 계정에서 기본 AWS 소유 키를 사용하여 암호화됩니다.

DynamoDB용 AWS 관리형 키를 사용하여 암호화된 테이블을 생성하려면

- 다음과 같이 암호화된 `Music` 테이블을 생성합니다.

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS
```

테이블 설명의 SSEDescription 상태는 ENABLED로 설정되고 SSEType은 KMS입니다.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

DynamoDB용 고객 관리형 키를 사용하여 암호화된 테이블을 생성하려면

- 다음과 같이 암호화된 Music 테이블을 생성합니다.

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

테이블 설명의 SSEDescription 상태는 ENABLED로 설정되고 SSEType은 KMS입니다.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

### 암호화 키 업데이트

또한 언제든지 DynamoDB 콘솔 또는 AWS CLI를 사용하여 AWS 소유 키, AWS 관리형 키, 고객 관리형 키 중에서 기존 테이블의 암호화 키를 업데이트할 수 있습니다.

암호화 키 업데이트(콘솔)

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 테이블(Tables)을 선택합니다.
3. 업데이트할 테이블을 선택합니다.
4. 작업(Actions) 드롭다운을 선택한 다음에 설정 업데이트(Update settings) 옵션을 선택합니다.
5. 추가 설정(Additional settings) 탭으로 이동합니다.
6. 암호화(Encryption)에서 암호화 관리(Manage encryption)를 선택합니다.
7. 다음과 같이 암호화 유형을 선택합니다.

   - Amazon DynamoDB 소유. AWS KMS 키는 DynamoDB에서 소유하고 관리합니다. 이 키 사용에 대한 추가 요금은 부과되지 않습니다.
   - AWS 관리형 키 키 별칭: `aws/dynamodb`. 이 키는 사용자의 계정에 저장되고 AWS Key Management Service에서 관리합니다(AWS KMS). AWS KMS 비용이 적용됩니다.
   - 계정에 저장되며 본인이 소유하고 관리합니다. 이 키는 사용자의 계정에 저장되고 AWS Key Management Service에서 관리합니다(AWS KMS). AWS KMS 비용이 적용됩니다.

   > **Note**
   >
   > 자체 키 소유 및 관리를 선택하는 경우 KMS 키 정책이 적절하게 설정되었는지 확인합니다. 자세한 내용은 고객 관리형 키에 대한 키 정책을 참조하세요.

   그런 다음 저장을 선택하여 암호화된 테이블을 업데이트합니다. 암호화 유형을 확인하려면 개요 탭에서 테이블 세부 정보를 확인하세요.

암호화 키 업데이트(AWS CLI)

다음 예에서는 AWS CLI를 사용해 암호화된 테이블을 업데이트하는 방법을 보여줍니다.

기본 AWS 소유 키를 사용하여 암호화된 테이블을 업데이트하려면

- 암호화된 `Music` 테이블을 다음 예와 같이 업데이트합니다.

```
aws dynamodb update-table \
  --table-name Music \
```

```
  --sse-specification Enabled=false
```

> **Note**
>
> 이 테이블은 이제 DynamoDB 서비스 계정에서 기본 AWS 소유 키를 사용하여 암호화됩니다.

DynamoDB용 AWS 관리형 키를 사용하여 암호화된 테이블을 업데이트하려면

- 암호화된 `Music` 테이블을 다음 예와 같이 업데이트합니다.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true
```

테이블 설명의 `SSEDescription` 상태는 `ENABLED`로 설정되고 `SSEType`은 `KMS`입니다.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

DynamoDB용 고객 관리형 키를 사용하여 암호화된 테이블을 업데이트하려면

- 암호화된 `Music` 테이블을 다음 예와 같이 업데이트합니다.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

테이블 설명의 `SSEDescription` 상태는 `ENABLED`로 설정되고 `SSEType`은 `KMS`입니다.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
```

```
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

# VPC 엔드포인트 및 IAM 정책을 사용하여 DynamoDB 연결 보호

Amazon DynamoDB와 온프레미스 애플리케이션 사이 연결을 비롯해 DynamoDB와 동일한 AWS 리전의 다른 AWS 리소스 사이 연결이 보호를 받습니다.

## 엔드포인트에 필요한 정책

Amazon DynamoDB는 리전 엔드포인트 정보를 열거하도록 해 주는 DescribeEndpoints API를 제공합니다. 퍼블릭 DynamoDB 엔드포인트에 대한 요청의 경우 IAM 또는 VPC 엔드포인트 정책에 명시적 거부 또는 암시적 거부가 있더라도 API는 구성된 DynamoDB IAM 정책에 관계없이 응답합니다. 이는 DynamoDB가 의도적으로 `DescribeEndpoints` API에 대한 권한을 건너뛰기 때문입니다.

VPC 엔드포인트에서 요청하려면 IAM 및 Virtual Private Cloud(VPC) 엔드포인트 정책이 모두 IAM `dynamodb:DescribeEndpoints` 작업을 사용하여 요청하는 Identity and Access Management(IAM) 보안 주체에 대한 `DescribeEndpoints` API 호출을 승인해야 합니다. 그렇지 않은 경우 `DescribeEndpoints` API에 대한 액세스가 거부됩니다.

다음은 엔드포인트 정책의 예입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "dynamodb:DescribeEndpoints",
            "Resource": "*"
        }
    ]
}
```

### 서비스와 온프레미스 클라이언트 및 애플리케이션 간의 트래픽

프라이빗 네트워크와 AWS 사이에 두 연결 옵션이 있습니다.

- AWS Site-to-Site VPN 연결. 자세한 정보는 AWS Site-to-Site VPN 사용 설명서의 AWS Site-to-Site VPN란 무엇입니까? 단원을 참조하세요.
- AWS Direct Connect 연결. 자세한 정보는 AWS Direct Connect 사용 설명서의 AWS Direct Connect란 무엇입니까? 단원을 참조하세요.

네트워크를 통한 DynamoDB 액세스는 AWS에서 게시한 API를 통해 이루어집니다. 클라이언트가 전송 계층 보안(TLS) 1.2를 지원해야 합니다. TLS 1.3을 권장합니다. 클라이언트는 DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Diffie-Hellman Ephemeral)와 같은 PFS(전달 완전 보안)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다. 또한, 액세스 키 ID와 IAM 보안 주체와 관련된 비밀 액세스 키를 사용하여 요청에 서명하거나 AWS Security Token Service(STS)를 사용하여 요청에 서명할 수 있는 임시 보안 자격 증명을 생성할 수 있습니다.

### 같은 리전에 있는 AWS 리소스 사이의 트래픽

DynamoDB용 Amazon Virtual Private Cloud(Amazon VPC) 엔드포인트는 VPC 내의 논리적 엔터티로서, DynamoDB에만 연결을 허용합니다. Amazon VPC는 DynamoDB로 요청을 라우팅하고, 응답을 다시 VPC로 라우팅합니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 참조하세요. VPC 엔드포인트의 액세스 제어에 사용할 수 있는 정책의 예는 IAM 정책을 사용하여 DynamoDB에 대한 액세스 제어를 참조하세요.

> **Note**
>
> Amazon VPC 엔드포인트는 AWS Site-to-Site VPN 또는 AWS Direct Connect를 통해 액세스할 수 없습니다.

## AWS Identity and Access Management(IAM) 및 DynamoDB

AWS Identity and Access Management는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 있도록 지원하는 AWS 서비스입니다. 관리자는 Amazon DynamoDB 및 DynamoDB Accelerator 리소스를 사용하기 위해 인증되고(로그인되고) 권한을 부여받을 수 있는(권한 있는) 사용자를 제어합니다. IAM을 사용하여 Amazon DynamoDB 및 DynamoDB Accelerator 모두에 대한 액세스 권한을 관리하고 보안 정책을 구현할 수 있습니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

주제


- Amazon DynamoDB의 Identity and Access Management
- IAM 정책 조건을 사용하여 세분화된 액세스 제어 구현


# Amazon DynamoDB의 Identity and Access Management

AWS Identity and Access Management(IAM)은 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 있도록 지원하는 AWS 서비스입니다. IAM 관리자는 누가 DynamoDB 리소스를 사용하도록 인증되고(로그인됨) 권한이 부여되는지(권한 있음)를 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

주제


- 고객
- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- Amazon DynamoDB에서 IAM을 사용하는 방법
- Amazon DynamoDB 자격 증명 기반 정책 예제
- Amazon DynamoDB 자격 증명 및 액세스 문제 해결
- DynamoDB 예약 용량 구매를 방지하는 IAM 정책


## 고객

AWS Identity and Access Management(IAM)를 사용하는 방법은 DynamoDB에서 수행하는 작업에 따라 달라집니다.

서비스 사용자 - DynamoDB 서비스를 사용하여 작업을 수행하는 경우 필요한 보안 인증 정보와 권한을 관리자가 제공합니다. 더 많은 DynamoDB 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방법을 이해하면 관리자에게 올바른 권한을 요청하는 데 도움이 됩니다. DynamoDB의 기능에 액세스할 수 없는 경우 Amazon DynamoDB 자격 증명 및 액세스 문제 해결 섹션을 참조하세요.

서비스 관리자 - 회사에서 DynamoDB 리소스를 책임지고 있는 경우 DynamoDB에 대한 전체 액세스 권한을 가지고 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 DynamoDB 기능과

리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사가 DynamoDB에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 Amazon DynamoDB에서 IAM을 사용하는 방법 섹션을 참조하세요.

IAM 관리자 - IAM 관리자라면 DynamoDB에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 DynamoDB 자격 증명 기반 정책 예제를 보려면 Amazon DynamoDB 자격 증명 기반 정책 예제 섹션을 참조하세요.

## ID를 통한 인증

인증은 ID 자격 증명을 사용하여 AWS에 로그인하는 방식입니다. AWS 계정 루트 사용자이나 IAM 사용자로, 또는 IAM 역할을 수임하여 인증(AWS에 로그인)받아야 합니다.

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 AWS에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증, Google 또는 Facebook 보안 인증이 페더레이션 ID의 예제입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 연동을 사용하여 AWS에 액세스하면 간접적으로 역할을 수임합니다.

사용자 유형에 따라AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. AWS에 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 계정에 로그인하는 방법을 참조하세요.

AWS에 프로그래밍 방식으로 액세스하는 경우, AWS에서는 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK) 및 명령줄 인터페이스(CLI)를 제공합니다. AWS 도구를 사용하지 않는 경우, 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 자세한 방법은 IAM 사용 설명서에서 API 요청용 AWS Signature Version 4를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS은(는) 다중 인증(MFA)을 사용하여 계정의 보안을 강화하는 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 다중 인증 및 IAM 사용 설명서에서 IAM의 AWS 다중 인증을 참조하세요.

### AWS 계정 루트 사용자

AWS 계정을(를) 생성할 때는 해당 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 단일 로그인 ID로 시작합니다. 이 ID는 AWS 계정루트 사용자라고 하며, 계정을 생성할 때 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명을 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는

데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 보안 인증이 필요한 작업을 참조하세요.

페더레이션 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 포함한 사용자가 ID 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 AWS 서비스에 액세스하도록 요구합니다.

페더레이션 ID는 엔터프라이즈 사용자 디렉터리, 웹 ID 공급자, AWS Directory Service, Identity Center 디렉터리의 사용자 또는 자격 증명 소스를 통해 제공된 자격 증명을 사용하여 AWS 서비스에 액세스하는 모든 사용자입니다. 페더레이션 ID는 AWS 계정에 액세스할 때 역할을 수임하고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을(를) 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 모든 AWS 계정 및 애플리케이션에서 사용하기 위해 고유한 ID 소스의 사용자 및 그룹 집합에 연결하고 동기화할 수 있습니다. IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇인가요?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가지고 있는 AWS 계정 내 ID입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우, 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서에서 IAM 사용자 사용 사례를 참조하세요.

IAM 역할

IAM 역할은 특정 권한을 가지고 있는 AWS 계정 계정 내 ID입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. AWS Management Console에서 일시적으로 IAM 역할을 수임하려면 사용자

에서 IAM 역할로 전환(콘솔)하면 됩니다. AWS CLI 또는 AWS API 작업을 직접적으로 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 관련 역할에 대한 자세한 내용은 IAM 사용 설명서의 Create a role for a third-party identity provider (federation)를 참조하세요. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연관짓습니다. 권한 집합에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 집합을 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 하지만 일부 AWS 서비스을(를) 사용하면 리소스에 정책을 직접 연결할 수 있습니다(역할을 프록시로서 사용하는 대신) . 교차 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하세요.
- 교차 서비스 액세스 - 일부 AWS 서비스는 다른 AWS 서비스의 특성을 사용합니다. 예를 들어, 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 위탁자의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여 AWS에서 작업을 수행하는 사람은 보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 AWS 서비스를 직접 호출하는 위탁자의 권한과 요청하는 AWS 서비스를 함께 사용하여 다운스트림 서비스에 대한 요청을 수행합니다. FAS 요청은 서비스에서 완료를 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 받은 경우에만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.
  - 서비스 연결 역할 – 서비스 연결 역할은 AWS 서비스에 연결된 서비스 역할의 한 유형입니다. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 링크 역할은

AWS 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수는 없습니다.

- Amazon EC2에서 실행 중인 애플리케이션 – IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS역할을 할당하고 해당 역할을 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

## 정책을 사용하여 액세스 관리

정책을 생성하고 AWS ID 또는 리소스에 연결하여 AWS에서 내 액세스를 제어합니다. 정책은 ID 또는 리소스와 연결될 때 해당 권한을 정의하는 AWS의 객체입니다. AWS는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청을 보낼 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 AWS에 JSON 문서로 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWSJSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자 및 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console, AWS CLI 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

### ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 AWS 계정에 속한 다수의 사용자, 그룹 및 역할에 독립적으로 추가할 수 있는 정책입니다. 관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책

이 포함되어 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 AWS 서비스가 포함될 수 있습니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACLs는 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3, AWS WAF 및 Amazon VPC는 ACL을 지원하는 대표적인 서비스입니다. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS는 비교적 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 ID 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔티티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCP) – SCP는 AWS Organizations에서 조직 또는 조직 단위(OU)에 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations는 기업이 소유하는 여러 개의 AWS 계정을 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 특성을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 AWS 계정 루트 사용자을

비롯하여 멤버 계정의 엔티티에 대한 권한을 제한합니다. 조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서에서 Service control policies을 참조하세요.

- 리소스 제어 정책(RCP) - RCP는 소유한 각 리소스에 연결된 IAM 정책을 업데이트하지 않고 계정의 리소스에 대해 사용 가능한 최대 권한을 설정하는 데 사용할 수 있는 JSON 정책입니다. RCP는 멤버 계정의 리소스에 대한 권한을 제한하며 조직에 속하는지 여부에 관계없이 AWS 계정 루트 사용자를 포함한 ID에 대한 유효 권한에 영향을 줄 수 있습니다. RCP를 지원하는 AWS 서비스 목록을 포함하여 Organizations 및 RCP에 대한 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCP)을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 ID 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련될 때 AWS가 요청을 허용할지를 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

## Amazon DynamoDB에서 IAM을 사용하는 방법

IAM을 사용하여 DynamoDB에 대한 액세스를 관리하기 전에 DynamoDB와 함께 사용할 수 있는 IAM 기능을 알아보세요.

| IAM 기능 | DynamoDB 지원 |
| --- | --- |
| ID 기반 정책 | 예 |
| 리소스 기반 정책 | 예 |
| 정책 작업 | 예 |
| 정책 리소스 | 예 |
| 정책 조건 키 | 예 |

| IAM 기능 | DynamoDB 지원 |
| --- | --- |
| ACLs | 아니요 |
| ABAC(정책의 태그) | 예 |
| 임시 보안 인증 | 예 |
| 보안 주체 권한 | 예 |
| 서비스 역할 | 예 |
| 서비스 연결 역할 | 예 |

DynamoDB 및 기타 AWS 서비스가 대부분의 IAM 기능과 작동하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서의 IAM으로 작업하는 AWS 서비스를 참조하세요.

DynamoDB에 대한 자격 증명 기반 정책

ID 기반 정책 지원: 예

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. ID 기반 정책에서는 위탁자가 연결된 사용자 또는 역할에 적용되므로 위탁자를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

DynamoDB 자격 증명 기반 정책 예제

DynamoDB 자격 증명 기반 정책의 예를 보려면 Amazon DynamoDB 자격 증명 기반 정책 예제 섹션을 참조하세요.

DynamoDB 내 리소스 기반 정책

리소스 기반 정책 지원: 예

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 AWS 서비스가 포함될 수 있습니다.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 위탁자로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하세요. 보안 주체와 리소스가 서로 다른 AWS 계정에 있는 경우, 신뢰할 수 있는 계정의 IAM 관리자는 보안 주체 엔터티(사용자 또는 역할)에도 리소스 액세스 권한을 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 위탁자에 액세스를 부여하는 경우, 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

DynamoDB의 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action` 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 일반적으로 정책 작업의 이름은 연결된 AWSAPI 작업의 이름과 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

DynamoDB 작업 목록을 보려면 서비스 권한 부여 참조에서 Amazon DynamoDB에서 정의한 작업을 참조하세요.

DynamoDB의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
aws
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
```

```
      "aws:action1",
      "aws:action2"
         ]
```

DynamoDB 자격 증명 기반 정책의 예를 보려면 Amazon DynamoDB 자격 증명 기반 정책 예제 섹션을 참조하세요.

DynamoDB의 정책 리소스

정책 리소스 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 `Resource`또는 `NotResource`요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

DynamoDB 리소스 유형 및 해당 ARN 목록을 보려면 서비스 승인 참조에서 Amazon DynamoDB에서 정의한 리소스를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 Amazon DynamoDB에서 정의한 작업을 참조하세요.

DynamoDB 자격 증명 기반 정책의 예를 보려면 Amazon DynamoDB 자격 증명 기반 정책 예제 섹션을 참조하세요.

DynamoDB의 정책 조건 키

서비스별 정책 조건 키 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Condition` 요소(또는 `Condition` 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. `Condition` 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 `Condition` 요소를 지정하거나 단일 `Condition` 요소에서 여러 키를 지정하는 경우, AWS는 논리적 `AND` 작업을 사용하여 평가합니다. 단일 조건 키의 여러 값을 지정하는 경우, AWS는 논리적 `OR` 작업을 사용하여 조건을 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS는 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

DynamoDB 조건 키 목록을 보려면 서비스 승인 참조에서 Amazon DynamoDB에 사용되는 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 Amazon DynamoDB에서 정의한 작업을 참조하세요.

DynamoDB 자격 증명 기반 정책의 예를 보려면 Amazon DynamoDB 자격 증명 기반 정책 예제 섹션을 참조하세요.

DynamoDB의 액세스 제어 목록(ACL)

ACL 지원: 아니요

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

DynamoDB의 ABAC(속성 기반 액세스 제어)

ABAC 지원(정책의 태그): 예

속성 기반 액세스 제어(ABAC)는 속성에 근거하여 권한을 정의하는 권한 부여 전략입니다. AWS에서는 이러한 속성을 태그라고 합니다. IAM 엔티티(사용자 또는 역할) 및 많은 AWS리소스에 태그를 연결할 수 있습니다. ABAC의 첫 번째 단계로 개체 및 리소스에 태그를 지정합니다. 그런 다음 위탁자의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계합니다.

ABAC는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로운 상황에 도움이 됩니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 통한 권한 정의를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 속성 기반 액세스 제어(ABAC) 사용을 참조하세요.

DynamoDB에서 임시 보안 인증 정보 사용

임시 자격 증명 지원: 예

일부 AWS 서비스는 임시 자격 증명을 사용하여 로그인할 때 작동하지 않습니다. 임시 보안 인증으로 작동하는 AWS 서비스를 비롯한 추가 정보는 IAM 사용 설명서의 IAM으로 작업하는 AWS 서비스를 참조하세요.

사용자 이름과 암호를 제외한 다른 방법을 사용하여 AWS Management Console에 로그인하면 임시 자격 증명을 사용하는 것입니다. 예를 들어, 회사의 Single Sign-On(SSO) 링크를 사용하여 AWS에 액세스하면 해당 프로세스에서 자동으로 임시 자격 증명을 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을 전환할 때 임시 자격 증명을 자동으로 생성합니다. 역할 전환에 대한 자세한 내용은 IAM 사용 설명서의 사용자에서 IAM 역할로 전환(콘솔)을 참조하세요.

AWS CLI 또는 AWS API를 사용하여 임시 자격 증명을 수동으로 만들 수 있습니다 그런 다음 이러한 임시 자격 증명을 사용하여 AWS에 액세스할 수 있습니다. AWS에서는 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성할 것을 권장합니다. 자세한 정보는 IAM의 임시 보안 자격 증명 섹션을 참조하세요.

DynamoDB의 서비스 간 보안 주체 권한

전달 액세스 세션(FAS) 지원: 예

IAM 사용자 또는 역할을 사용하여 AWS에서 작업을 수행하는 사람은 위탁자로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 AWS 서비스를 직접 호출하는 위탁자의 권한과 요청하는 AWS 서비스를 함께 사용하여 다운스트림 서비스에 대한 요청을 수행합니다. FAS 요청은 서비스에서 완료를 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 받은 경우에만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

DynamoDB의 서비스 역할

서비스 역할 지원: 예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.

> **Warning**
>
> 서비스 역할에 대한 권한을 변경하면 DynamoDB 기능이 중단될 수 있습니다. DynamoDB에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집하세요.

DynamoDB에 대한 서비스 연결 역할

서비스 링크 역할 지원: 예

서비스 연결 역할은 AWS 서비스에 연결된 서비스 역할의 한 유형입니다. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 링크 역할은 AWS 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 IAM으로 작업하는 AWS 서비스를 참조하세요. 서비스 연결 역할 열에서 `Yes`이(가) 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

DynamoDB에서 지원되는 서비스 연결 역할

DynamoDB에서는 다음과 같은 서비스 연결 역할이 지원됩니다.

- DynamoDB는 여러 AWS 리전 간의 글로벌 테이블 복제를 위해 서비스 연결 역할 AWSServiceRoleForDynamoDBReplication을 사용합니다. AWSServiceRoleForDynamoDBReplication 서비스 연결 역할에 대한 자세한 내용은 DynamoDB 글로벌 테이블 보안 섹션을 참조하세요.
- DynamoDB Accelerator(DAX)는 DAX 클러스터를 구성하고 유지 관리하기 위해 서비스 연결 역할 AWSServiceRoleForDAX를 사용합니다. AWSServiceRoleForDAX 서비스 연결 역할에 대한 자세한 내용은 the section called “DAX에 대한 서비스 연결 역할 사용” 섹션을 참조하세요.

이러한 DynamoDB 서비스 연결 역할 외에도 DynamoDB는 프로비저닝된 용량 모드 테이블에서 처리량 설정을 자동으로 관리하기 위해 Application Auto Scaling 서비스를 사용합니다. Application Auto

Scaling 서비스는 오토 스케일링이 활성화된 DynamoDB 테이블의 처리량 설정을 관리하기 위해 서비스 연결 역할 AWSServiceRoleForApplicationAutoScaling_DynamoDBTable을 사용합니다. 자세한 내용은 Application Auto Scaling에 대한 서비스 연결 역할을 참조하세요.

## Amazon DynamoDB 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할에는 DynamoDB 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 AWS API를 사용해 태스크를 수행할 수 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 맡을 수 있습니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성(콘솔)을 참조하세요.

각 리소스 유형에 대한 ARN 형식을 포함하여 DynamoDB에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 승인 참조에서 Amazon DynamoDB에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

주제

- 정책 모범 사례
- DynamoDB 콘솔 사용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용
- Amazon DynamoDB에서 자격 증명 기반 정책 사용

### 정책 모범 사례

자격 증명 기반 정책에 따라 계정에서 사용자가 DynamoDB 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책으로 시작하고 최소 권한을 향해 나아가기 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. AWS 계정에서 사용할 수 있습니다. 사용 사례에 고유한 AWS 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있

는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.

- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS CloudFormation와 같이, 특정 AWS 서비스를 통해 사용되는 경우에만 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 – AWS 계정에 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우, 추가 보안을 위해 MFA를 설정합니다. API 작업을 직접 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

DynamoDB 콘솔 사용

Amazon DynamoDB 콘솔에 액세스하려면 최소한의 권한 집합이 있어야 합니다. 이러한 권한은 AWS 계정에서 DynamoDB 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 AWS API만 직접 호출하는 사용자에게 최소 콘솔 권한을 허용할 필요가 없습니다. 대신 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 DynamoDB 콘솔을 여전히 사용할 수 있도록 하려면 DynamoDB `ConsoleAccess` 또는 `ReadOnly` AWS 관리형 정책을 엔터티에 추가합니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하세요.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로 이 태스크를 완료할 수 있는 권한이 포함됩니다.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon DynamoDB에서 자격 증명 기반 정책 사용

이 항목에서는 Amazon DynamoDB에서 AWS Identity and Access Management(IAM) 정책을 사용하는 방법을 설명하고 예시를 제공합니다. 이 예제에서는 계정 관리자가 IAM 자격 증명(사용자, 그룹, 역할)에 권한 정책을 연결함으로써 Amazon DynamoDB 리소스에 대한 작업을 수행할 권한을 부여하는 방법을 보여 줍니다.

이 주제의 섹션에서는 다음 내용을 학습합니다.

- Amazon DynamoDB 콘솔을 사용하는 데 필요한 IAM 권한
- Amazon DynamoDB에 대한 AWS 관리형(미리 정의된) IAM 정책

- 고객 관리형 정책 예

다음은 권한 정책의 예제입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        }
    ]
}
```

앞의 정책에는 *account-id*에 의해 지정된 AWS 계정이 소유하는 us-west-2 AWS 리전 내 테이블에서 세 가지 DynamoDB 작업(dynamodb:DescribeTable, dynamodb:Query, 및 dynamodb:Scan)에 대한 권한을 부여하는 문이 하나 있습니다. Resource 값의 Amazon 리소스 이름(ARN)은 권한이 적용되는 테이블을 지정합니다.

Amazon DynamoDB 콘솔을 사용하는 데 필요한 IAM 권한

사용자가 DynamoDB 콘솔로 작업하려면 AWS 계정의 DynamoDB 리소스로 작업하도록 허용하는 최소 권한 집합이 있어야 합니다. 이 DynamoDB 권한 이외에 콘솔에는 다음 권한이 필요합니다.

- 지표 및 그래프를 표시할 수 있는 Amazon CloudWatch 권한
- DynamoDB 데이터를 내보내고 가져오기 위한 AWS Data Pipeline 권한
- 내보내기 및 가져오기 작업에 필요한 역할에 액세스할 수 있는 AWS Identity and Access Management 권한.
- CloudWatch 경보가 트리거될 때마다 사용자에게 알릴 수 있는 Amazon Simple Notification Service 권한

- DynamoDB Streams 레코드를 처리할 수 있는 AWS Lambda 권한

최소 필수 권한보다 더 제한적인 IAM 정책을 만들면 콘솔은 해당 IAM 정책에 연결된 사용자에 대해 의도대로 작동하지 않습니다. 이 사용자가 DynamoDB 콘솔을 계속 사용할 수 있도록 하려면 `AmazonDynamoDBReadOnlyAccess` AWS 관리형 정책도 사용자에게 연결합니다(Amazon DynamoDB에 대한 AWS 관리형(미리 정의된) IAM 정책 참조).

AWS CLI 또는 Amazon DynamoDB API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다.

> **Note**
>
> VPC 엔드포인트를 참조하는 경우 IAM 작업(dynamodb:DescribeEndpoints)을 사용하여 요청하는 IAM 보안 주체에 대한 DescribeEndpoints API 호출도 승인해야 합니다. 자세한 내용은 엔드포인트에 필요한 정책을 참조하세요.

Amazon DynamoDB에 대한 AWS 관리형(미리 정의된) IAM 정책

AWS는 AWS에서 생성하고 관리하는 독립형 IAM 정책을 제공하여 몇 가지 일반적인 사용 사례를 처리합니다. 이러한 AWS 관리형 정책은 사용자가 필요한 권한을 조사할 필요가 없도록 일반 사용 사례에 필요한 권한을 부여합니다. 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

계정의 사용자에게 연결할 수 있는 다음 AWS 관리형 정책은 DynamoDB에 고유하며, 사용 사례 시나리오별로 그룹화되어 있습니다.

- AmazonDynamoDBReadOnlyAccess - AWS Management Console을 통해 DynamoDB 리소스에 대한 읽기 전용 액세스 권한을 부여합니다.
- AmazonDynamoDBFullAccess - AWS Management Console을 통해 DynamoDB 리소스에 대한 전체 액세스 권한을 부여합니다.

IAM 콘솔에 로그인하고 이 콘솔에서 특정 정책을 검색하여 이러한 AWS 관리형 권한 정책을 검토할 수 있습니다.

> **Important**
>
> 가장 좋은 방법은 필요한 사용자, 역할 또는 그룹에 최소 권한을 부여하는 사용자 지정 IAM 정책을 생성하는 것입니다.

## 고객 관리형 정책 예

이 단원에서는 다양한 DynamoDB 작업에 대한 권한을 부여하는 정책의 예를 제공합니다. 이러한 정책은 AWS SDK 또는 AWS CLI를 사용하는 경우에 유효합니다. 콘솔을 사용하는 경우 콘솔에 해당하는 추가 권한을 부여해야 합니다. 자세한 내용은 Amazon DynamoDB 콘솔을 사용하는 데 필요한 IAM 권한 섹션을 참조하세요.

> **Note**
>
> 다음 정책 예시에서는 모두 AWS 리전 중 하나를 사용하고 가상의 계정 ID와 테이블 이름을 포함합니다.

예시:

- 테이블에서 모든 DynamoDB 작업에 대한 권한을 부여하는 IAM 정책
- DynamoDB 테이블의 항목에 대한 읽기 전용 권한을 부여하는 IAM 정책
- 특정 DynamoDB 테이블과 관련 인덱스에 대한 액세스 권한을 부여하는 IAM 정책
- DynamoDB 테이블에서 읽기, 쓰기, 업데이트 및 삭제 액세스 권한에 대한 IAM 정책
- 동일한 AWS 계정의 다른 DynamoDB 환경에 대한 IAM 정책
- DynamoDB 예약 용량 구매를 방지하는 IAM 정책
- DynamoDB 스트림에 대해서만 읽기 액세스 권한을 부여하는 IAM 정책(테이블은 부여하지 않음)
- AWS Lambda 함수가 DynamoDB 스트림 레코드에 액세스하도록 허용하는 IAM 정책
- DynamoDB Accelerator(DAX) 클러스터에 대한 읽기 및 쓰기 액세스 권한을 위한 IAM 정책

IAM 사용 설명서에는 다음 세 가지 추가 DynamoDB 예시가 포함되어 있습니다.

- Amazon DynamoDB: 특정 테이블에 대한 액세스 허용
- Amazon DynamoDB: 특정 열에 대한 액세스 허용
- Amazon DynamoDB: Amazon Cognito ID를 기준으로 DynamoDB에 대한 행 수준 액세스 허용

### 테이블에서 모든 DynamoDB 작업에 대한 권한을 부여하는 IAM 정책

다음 정책은 Books라는 테이블에서 모든 DynamoDB 작업을 위한 권한을 부여합니다. Resource에 지정된 리소스 ARN은 특정 AWS 리전 내의 테이블을 식별합니다. Resource ARN에서 테이블 이름 Books를 와일드카드 문자(*)로 바꾸면 모든 DynamoDB 작업이 계정의 모든 테이블에서 허용됩니다.

이 정책이나 IAM 정책에서 와일드카드 문자를 사용하려면 먼저 보안에 미칠 수 있는 영향을 신중히 고려하세요.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

> **Note**
>
> 이 예제에서는 와일드카드 문자(*)를 사용하여 관리, 데이터 작업, 모니터링, DynamoDB 예약 용량 구매 등의 모든 작업을 허용합니다. 대신 허용할 각 작업과 해당 사용자, 역할 또는 그룹에 필요한 작업만 명시적으로 지정하는 것이 좋습니다.

DynamoDB 테이블의 항목에 대한 읽기 전용 권한을 부여하는 IAM 정책

다음 권한 정책은 `GetItem`, `BatchGetItem`, `Scan`, `Query` 및 `ConditionCheckItem` DynamoDB 작업에 대한 권한만 부여하며 결과적으로 `Books` 테이블에 대한 읽기 전용 액세스 권한을 설정합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
```

```
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

특정 DynamoDB 테이블과 관련 인덱스에 대한 액세스 권한을 부여하는 IAM 정책

다음 정책은 Books라는 DynamoDB 테이블과 해당 테이블의 모든 인덱스에서 데이터 수정 작업에 대한 권한을 부여합니다. 인덱스 작동 방식에 대한 자세한 내용은 DynamoDB에서 보조 인덱스를 사용하여 데이터 액세스 개선 단원을 참조하세요.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessTableAllIndexesOnBooks",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
              "dynamodb:GetItem",
              "dynamodb:BatchGetItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
```

```
}
```

DynamoDB 테이블에서 읽기, 쓰기, 업데이트 및 삭제 액세스 권한에 대한 IAM 정책

애플리케이션이 Amazon DynamoDB 테이블, 인덱스 및 스트림에서 데이터를 생성하고, 읽고, 업데이트하고, 삭제하도록 허용해야 하는 경우 이 정책을 사용합니다. AWS 리전 이름, 계정 ID, 계정 ID, 테이블 이름 또는 와일드카드 문자(*)는 적절하게 대체합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBIndexAndStreamAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetShardIterator",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/stream/*"
            ]
        },
        {
            "Sid": "DynamoDBTableAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:DescribeTable",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Scan",
```

```
                "dynamodb:Query",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        },
        {
            "Sid": "DynamoDBDescribeLimitsAccess",
            "Effect": "Allow",
            "Action": "dynamodb:DescribeLimits",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

모든 AWS 리전에서 이 계정의 모든 DynamoDB 테이블을 포함하도록 이 정책을 확장하려면 리전 및 테이블 이름에 와일드카드(*)를 사용합니다. 예:

```
"Resource":[
               "arn:aws:dynamodb:*:123456789012:table/*",
               "arn:aws:dynamodb:*:123456789012:table/*/index/*"
               ]
```

동일한 AWS 계정의 다른 DynamoDB 환경에 대한 IAM 정책

별도의 환경이 있고 각 환경에서 `ProductCatalog`라는 테이블의 자체 버전을 유지한다고 가정해 보겠습니다. 동일한 AWS 계정에서 2개의 `ProductCatalog` 테이블을 생성하는 경우 권한이 설정되는 방식 때문에 한 환경에서의 작업이 다른 환경에 영향을 줄 수 있습니다. 예를 들어 동시 제어 영역 작업(예:`CreateTable`)의 수에 대한 할당량은 AWS 계정 수준에서 설정됩니다.

따라서 한 환경에서 작업을 수행할 때마다 다른 환경에서 사용 가능한 작업의 수가 감소합니다. 또한 한 환경의 코드가 다른 환경의 테이블에 실수로 액세스할 수 있는 위험도 있습니다.

> **Note**
>
> 프로덕션 및 테스트 워크로드를 분리하여 이벤트의 잠재적 '영향 범위'를 제어할 수 있으려면 테스트 및 프로덕션 워크로드에 대해 별도의 AWS 계정을 생성하는 것이 좋습니다. 자세한 내용은 AWS 계정 관리 및 분리 단원을 참조하세요.

또한 Amit과 Alice라는 두 개발자가 ProductCatalog 테이블을 테스트 중이라고 가정해 보겠습니다. 개발자가 각각 별도의 AWS 계정을 요구하는 대신 동일한 테스트 AWS 계정을 공유할 수 있습니다. 이 테스트 계정에서 Alice_ProductCatalog 및 Amit_ProductCatalog와 같이 각 개발자가 작업할 수 있는 동일한 테이블의 복제본을 만들 수 있습니다. 이 경우 테스트 환경용으로 생성한 AWS 계정에서 사용자 Alice 및 Amit을 만들 수 있습니다. 그런 다음 이들 사용자에게 각자가 소유하는 테이블에 대해 DynamoDB 작업을 수행할 수 있는 권한을 부여할 수 있습니다.

이러한 IAM 사용자 권한을 부여하려면 다음 중 하나를 수행할 수 있습니다.

- 사용자마다 개별 정책을 만들고 각 정책을 사용자에게 개별적으로 연결합니다. 예를 들면, 다음 정책을 사용자 Alice에 연결하여 이 사용자가 Alice_ProductCatalog 테이블에서 모든 DynamoDB 작업에 액세스하도록 허용할 수 있습니다.

  JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnAliceTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:DeleteItem",
              "dynamodb:DescribeContributorInsights",
              "dynamodb:RestoreTableToPointInTime",
              "dynamodb:ListTagsOfResource",
              "dynamodb:CreateTableReplica",
              "dynamodb:UpdateContributorInsights",
              "dynamodb:CreateBackup",
              "dynamodb:DeleteTable",
              "dynamodb:UpdateTableReplicaAutoScaling",
              "dynamodb:UpdateContinuousBackups",
              "dynamodb:TagResource",
              "dynamodb:DescribeTable",
              "dynamodb:GetItem",
              "dynamodb:DescribeContinuousBackups",
              "dynamodb:BatchGetItem",
              "dynamodb:UpdateTimeToLive",
              "dynamodb:BatchWriteItem",
              "dynamodb:ConditionCheckItem",
              "dynamodb:UntagResource",
              "dynamodb:PutItem",
```

```
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteTableReplica",
                "dynamodb:DescribeTimeToLive",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:UpdateTable",
                "dynamodb:DescribeTableReplicaAutoScaling",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:DescribeLimits",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
Alice_ProductCatalog/*"
        }
    ]
}
```

그런 다음 사용자 Amit에 대해 다른 리소스(`Amit_ProductCatalog` 테이블)를 사용하여 유사한 정책을 만들 수 있습니다.

- 정책을 개별 사용자에게 연결하는 대신, IAM 정책 변수를 사용하여 단일 정책을 작성하고 이 정책을 그룹에 연결할 수도 있습니다. 이 경우 그룹을 만들어야 하고, 여기서는 Alice와 Amit 두 사용자 모두 해당 그룹에 추가해야 합니다. 다음 예제에서는 `${aws:username}_ProductCatalog` 테이블에서 모든 DynamoDB 작업을 수행할 수 있는 권한을 부여합니다. 정책이 평가될 때 정책 변수 `${aws:username}`은 요청자의 사용자 이름으로 대체됩니다. 예를 들어 Alice가 항목을 추가하라는 요청을 보내는 경우 이 작업은 Alice가 항목을 `Alice_ProductCatalog` 테이블에 추가하는 경우에만 허용됩니다.

  JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ActionsOnUserSpecificTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
```

```
                "dynamodb:BatchWriteItem",
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
${aws:username}_ProductCatalog"
        },
        {
            "Sid": "AdditionalPrivileges",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables",
                "dynamodb:DescribeTable",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/*"
        }
    ]
}
```

> **Note**
>
> IAM 정책 변수를 사용할 때는 정책에서 IAM 정책 언어의 2012-10-17 버전을 명시적으로 지정해야 합니다. IAM 정책 언어의 기본 버전(2008-10-17)은 정책 변수를 지원하지 않습니다.

평소대로 특정 테이블을 리소스로 식별하는 대신, 아래 예제와 같이 와일드카드 문자(*)를 사용하여 테이블 이름에 요청자의 사용자 이름이 접두사로 붙은 모든 테이블에 대한 권한을 부여할 수 있습니다.

```
"Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/${aws:username}_*"
```

DynamoDB 예약 용량 구매를 방지하는 IAM 정책

Amazon DynamoDB 예약 용량을 구입할 경우 선납금을 1회 지불하고 일정 기간에 대해 최소 사용량 수준에 맞춰 약정 요금을 지불하므로 상당한 비용 절감을 얻을 수 있습니다. AWS Management Console을 사용하여 예약 용량을 확인하고 구매할 수 있습니다. 하지만 조직의 모든 사용자가 예

약 용량을 구매할 수 있도록 하고 싶지 않을 수 있습니다. 예약 용량에 대한 자세한 내용은 Amazon DynamoDB 요금 단원을 참조하세요.

DynamoDB는 예약 용량 관리에 대한 액세스 권한을 제어하기 위한 다음과 같은 API 작업을 제공합니다.

- `dynamodb:DescribeReservedCapacity` - 현재 유효한 예약 용량 구매를 반환합니다.
- `dynamodb:DescribeReservedCapacityOfferings` - 현재 AWS에서 제공하는 예약 용량 계획에 대한 세부 정보를 반환합니다.
- `dynamodb:PurchaseReservedCapacityOfferings` - 실제 예약 용량 구매를 수행합니다.

AWS Management Console은 이러한 API 작업을 사용하여 예약 용량 정보를 표시하고 예약 용량을 구매합니다. 이러한 작업은 콘솔을 통해서만 액세스할 수 있으므로 애플리케이션 프로그램에서는 호출할 수 없습니다. 하지만 IAM 권한 정책에서 이들 작업에 대한 액세스를 허용 또는 거부할 수 있습니다.

다음 정책을 사용하면 사용자가 AWS Management Console을 사용하여 예약 용량 구매 및 제품을 조회할 수는 있지만, 새로운 구매는 거부됩니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

이 정책에서는 와일드카드 문자(*)를 사용하여 모두에 대해 설명 권한을 허용하고, 모두에 대해 DynamoDB 예약 용량 구매를 거부합니다.

DynamoDB 스트림에 대해서만 읽기 액세스 권한을 부여하는 IAM 정책(테이블은 부여하지 않음)

테이블에서 DynamoDB Streams를 활성화하면 테이블 항목의 모든 수정에 대한 정보가 캡처됩니다. 자세한 내용은 DynamoDB Streams에 대한 변경 데이터 캡처 섹션을 참조하세요.

때로는 애플리케이션이 DynamoDB 테이블에서 데이터를 읽는 것은 금지하지만 테이블의 스트림에는 액세스하도록 허용하려는 경우도 있을 수 있습니다. 예를 들어, 항목 업데이트가 감지될 때 스트림을 폴링하고 Lambda 함수를 호출한 후 추가 처리를 수행하도록 AWS Lambda를 구성할 수 있습니다.

다음 작업은 DynamoDB Streams에 대한 액세스를 제어하는 데 사용할 수 있습니다.

- `dynamodb:DescribeStream`
- `dynamodb:GetRecords`
- `dynamodb:GetShardIterator`
- `dynamodb:ListStreams`

다음 예제 정책은 `GameScores`라는 테이블의 스트림에 액세스할 수 있는 권한을 사용자에게 부여합니다. ARN의 와일드카드 문자(*)는 해당 테이블과 연결된 모든 스트림과 일치합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessGameScoresStreamOnly",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
    ]
```

```
}
```

이 정책에서는 GameScores 테이블 자체가 아니라 해당 테이블의 스트림에 액세스하도록 허용합니다.

AWS Lambda 함수가 DynamoDB 스트림 레코드에 액세스하도록 허용하는 IAM 정책

DynamoDB 스트림의 새 이벤트를 기반으로 특정 작업이 수행되기를 원할 경우 이러한 새 이벤트를 통해 트리거되는 AWS Lambda 함수를 작성할 수 있습니다. 이러한 Lambda 함수는 DynamoDB 스트림에서 데이터를 읽을 수 있는 권한을 필요로 합니다. DynamoDB Streams와 Lambda를 함께 사용하는 방법에 대한 자세한 내용은 DynamoDB Streams 및 AWS Lambda 트리거 단원을 참조하세요.

Lambda에 권한을 부여하려면 Lambda 함수의 IAM 역할(실행 역할이라고도 함)과 연결되는 권한 정책을 사용합니다. 이 정책은 Lambda 함수를 생성할 때 지정합니다.

예를 들어, 다음의 권한 정책을 실행 역할과 연결하여 나열된 DynamoDB Streams 작업을 수행할 수 있는 권한을 Lambda에 부여할 수 있습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "APIAccessForDynamoDBStreams",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
    ]
}
```

자세한 내용은 AWS Lambda 개발자 안내서의 AWS Lambda 권한 단원을 참조하세요.

### DynamoDB Accelerator(DAX) 클러스터에 대한 읽기 및 쓰기 액세스 권한을 위한 IAM 정책

다음 정책에서는 DynamoDB Accelerator(DAX)에 대한 읽기, 쓰기, 업데이트 및 삭제 액세스 권한을 허용하지만 연결된 DynamoDB 테이블에 대해서는 허용하지 않습니다. 이 정책을 사용하려면 AWS 리전 이름, 계정 ID, DAX 클러스터 이름을 대체합니다.

> **Note**
>
> 이 정책은 DAX 클러스터에 대한 액세스 권한은 부여하지만 연결된 DynamoDB 테이블에 대한 액세스 권한은 부여하지 않습니다. DAX 클러스터가 사용자를 대신해 DynamoDB 테이블에서 이와 같은 작업을 수행하려면 클러스터에 올바른 정책이 있어야 합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonDynamoDBDAXDataOperations",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:PutItem",
                "dax:ConditionCheckItem",
                "dax:BatchGetItem",
                "dax:BatchWriteItem",
                "dax:DeleteItem",
                "dax:Query",
                "dax:UpdateItem",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:eu-west-1:123456789012:cache/MyDAXCluster"
        }
    ]
}
```

계정의 모든 AWS 리전에 대한 DAX 액세스 권한을 포함하도록 이 정책을 확장하려면 리전 이름에 와일드카드 문자(*)를 사용합니다.

```
"Resource": "arn:aws:dax:*:123456789012:cache/MyDAXCluster"
```

## Amazon DynamoDB 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 DynamoDB 및 IAM에서 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

주제

- DynamoDB에서 작업을 수행할 권한이 없음
- iam:PassRole을 수행하도록 인증되지 않음
- 내 AWS 계정 외부의 사람이 내 DynamoDB 리소스에 액세스할 수 있게 허용하고 싶음

### DynamoDB에서 작업을 수행할 권한이 없음

AWS Management Console에서 작업을 수행할 권한이 없다는 메시지가 나타나는 경우 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 사용자 이름과 비밀번호를 제공한 사람입니다.

다음 예제 오류는 `mateojackson` 사용자가 콘솔을 사용하여 가상 *`my-example-widget`* 리소스에 대한 세부 정보를 보려고 하지만 가상 `aws:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

이 경우 Mateo는 *`my-example-widget`* 작업을 사용하여 `aws:`*`GetWidget`* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

### iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 DynamoDB에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스에서는 새로운 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 해당 서비스에 기존 역할을 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예시 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 DynamoDB에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 iam:PassRole 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 AWS 계정 외부의 사람이 내 DynamoDB 리소스에 액세스할 수 있게 허용하고 싶음

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- DynamoDB에서 이러한 기능을 지원하는지 여부를 알아보려면 Amazon DynamoDB에서 IAM을 사용하는 방법 섹션을 참조하세요.
- 소유하고 있는 AWS 계정의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 자신이 소유한 다른 AWS 계정의 IAM 사용자에 대한 액세스 권한 제공을 참조하세요.
- 리소스에 대한 액세스 권한을 서드 파티 AWS 계정에게 제공하는 방법을 알아보려면 IAM 사용 설명서의 서드 파티가 소유한 AWS 계정에 대한 액세스 제공을 참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

## DynamoDB 예약 용량 구매를 방지하는 IAM 정책

Amazon DynamoDB 예약 용량을 구입할 경우 선납금을 1회 지불하고 일정 기간에 대해 최소 사용량 수준에 맞춰 약정 요금을 지불하므로 상당한 비용 절감을 얻을 수 있습니다. AWS Management Console을 사용하여 예약 용량을 확인하고 구매할 수 있습니다. 하지만 조직의 모든 사용자가 예약 용량을 구매할 수 있도록 하고 싶지 않을 수 있습니다. 예약 용량에 대한 자세한 내용은 Amazon DynamoDB 요금 단원을 참조하세요.

DynamoDB는 예약 용량 관리에 대한 액세스 권한을 제어하기 위한 다음과 같은 API 작업을 제공합니다.

- `dynamodb:DescribeReservedCapacity` - 현재 유효한 예약 용량 구매를 반환합니다.
- `dynamodb:DescribeReservedCapacityOfferings` - 현재 AWS에서 제공하는 예약 용량 계획에 대한 세부 정보를 반환합니다.
- `dynamodb:PurchaseReservedCapacityOfferings` - 실제 예약 용량 구매를 수행합니다.

AWS Management Console은 이러한 API 작업을 사용하여 예약 용량 정보를 표시하고 예약 용량을 구매합니다. 이러한 작업은 콘솔을 통해서만 액세스할 수 있으므로 애플리케이션 프로그램에서는 호출할 수 없습니다. 하지만 IAM 권한 정책에서 이들 작업에 대한 액세스를 허용 또는 거부할 수 있습니다.

다음 정책을 사용하면 사용자가 AWS Management Console을 사용하여 예약 용량 구매 및 제품을 조회할 수는 있지만, 새로운 구매는 거부됩니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

이 정책에서는 와일드카드 문자(*)를 사용하여 모두에 대해 설명 권한을 허용하고, 모두에 대해 DynamoDB 예약 용량 구매를 거부합니다.

## IAM 정책 조건을 사용하여 세분화된 액세스 제어 구현

DynamoDB에서 권한을 부여할 때 권한 정책이 적용되는 방식을 결정하는 조건을 지정할 수 있습니다.

### 개요

DynamoDB에서는 IAM 정책을 사용하여 권한을 부여할 때 조건을 지정하는 옵션이 있습니다(Amazon DynamoDB의 Identity and Access Management 참조). 예를 들어, 다음을 수행할 수 있습니다.

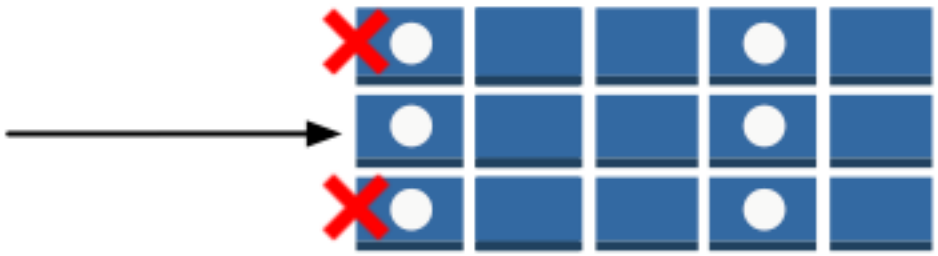
- 사용자에게 테이블 또는 보조 인덱스의 특정 항목 및 속성에 대한 읽기 전용 액세스를 허용하는 권한을 부여할 수 있습니다.
- 사용자 ID를 기준으로 테이블의 특정 속성에 대한 쓰기 전용 액세스 권한을 해당 사용자에게 부여할 수 있습니다.

다음 단원의 사용 사례에서 설명되어 있듯이 DynamoDB에서는 조건 키를 사용하여 IAM 정책에서 조건을 지정할 수 있습니다.

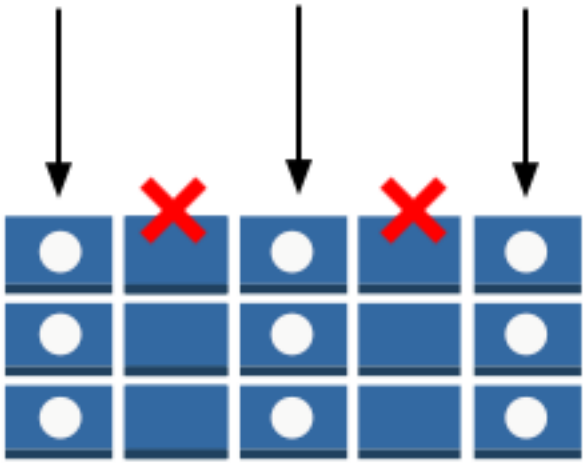
#### 권한 사용 사례

DynamoDB API 작업에 대한 액세스를 제어할 수 있을 뿐만 아니라, 개별 데이터 항목과 속성에 대한 액세스도 제어할 수 있습니다. 예를 들어, 다음을 수행할 수 있습니다.

- 테이블에 대한 권한을 부여하되, 특정 기본 키 값을 기반으로 해당 테이블에서 특정 항목에 대한 액세스를 제한할 수 있습니다. 다음 그림과 같이, 모든 사용자가 저장한 게임 데이터가 단일 테이블에 저장되지만 어떤 사용자도 자신이 소유하지 않은 데이터 항목에 액세스할 수 없는 소셜 네트워크 게임 앱이 그 예입니다.



- 속성의 하위 집합만 사용자에게 표시되도록 정보를 숨길 수 있습니다. 사용자 위치에 따라 인근 공항의 항공편 데이터를 표시하는 앱이 그 예입니다. 항공사 이름, 도착 및 출발 시간, 항공편 번호가 모두 표시됩니다. 하지만 다음 그림과 같이, 기장 이름과 승객 수와 같은 속성은 숨겨집니다.



이러한 종류의 세분화된 액세스 제어를 구현하려면 보안 자격 증명에 액세스하기 위한 조건과 관련 권한을 지정하는 IAM 권한 정책을 작성합니다. 그런 다음 IAM 콘솔을 사용하여 만드는 사용자, 그룹 또는 역할에 이 정책을 적용할 수 있습니다. IAM 정책은 테이블의 개별 항목에 대한 액세스 또는 이러한 항목의 속성에 대한 액세스를 제한하거나 이러한 두 가지 액세스를 동시에 제한할 수 있습니다.

또는 웹 자격 증명 연동을 사용하여 Login with Amazon, Facebook 또는 Google에서 인증된 사용자의 액세스를 제어할 수도 있습니다. 자세한 내용은 웹 아이덴티티 페더레이션 사용 섹션을 참조하세요.

IAM `Condition` 요소를 사용하여 세분화된 액세스 제어 정책을 구현합니다. `Condition` 요소를 권한 정책에 추가하여 특정 비즈니스 요구 사항에 따라 DynamoDB 테이블 및 인덱스의 항목 및 속성에 대한 액세스를 허용하거나 거부할 수 있습니다.

플레이어가 다양한 게임을 선택하고 플레이할 수 있는 모바일 게임 앱을 예로 들어 보겠습니다. 이 앱은 `GameScores`라는 DynamoDB 테이블을 사용하여 높은 점수와 기타 사용자 데이터를 추적합니다. 테이블의 각 항목은 사용자 ID와 사용자가 플레이한 게임의 이름에 따라 고유하게 식별됩니다. `GameScores` 테이블은 파티션 키(`UserId`)와 정렬 키(`GameTitle`)로 구성된 기본 키를 갖습니다. 사용자는 자신의 사용자 ID와 연결된 게임 데이터에만 액세스할 수 있습니다. 게임을 플레이하려는 사용자는 보안 정책이 연결된 `GameRole`이라는 IAM 역할에 속해야 합니다.

이 앱에서 사용자 권한을 관리하기 위해 다음과 같은 권한 정책을 작성할 수 있습니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

GameScores 테이블(Resource 요소)에서의 특정 DynamoDB 작업(Action 요소)에 대한 권한을 부여하는 이외에, Condition 요소는 다음의 DynamoDB 고유 조건 키를 사용하여 다음과 같이 권한을 제한합니다.

- dynamodb:LeadingKeys - 이 조건 키는 사용자가 파티션 키 값이 자신의 사용자 ID와 일치하는 항목만 액세스하도록 허용합니다. 이 ID(${www.amazon.com:user_id})는 치환 변수입니다. 치환 변수에 대한 자세한 내용은 웹 아이덴티티 페더레이션 사용 단원을 참조하세요.
- dynamodb:Attributes - 이 조건 키는 권한 정책에 나열된 작업만 지정된 속성 값을 반환할 수 있도록 이러한 속성에 대한 액세스를 제한합니다. 또한 StringEqualsIfExists 절을 사용하면 앱에서 작업이 적용될 특정 속성 목록만 제공되고 모든 속성을 요청할 수는 없습니다.

IAM 정책을 평가하면 결과가 항상 true(액세스 허용됨)이거나 false(액세스 거부됨)입니다. Condition 요소 중 한 부분이라도 false이면 전체 정책이 false로 평가되어 액세스가 거부됩니다.

> **Important**
>
> dynamodb:Attributes를 사용하는 경우 해당 정책에 나열된 테이블 및 보조 인덱스에 대한 모든 기본 키 및 인덱스 속성의 이름을 지정해야 합니다. 그렇지 않으면, DynamoDB에서 이러한 키 속성을 사용하여 요청된 작업을 수행할 수 없습니다.

IAM 정책 설명서에는 수평 탭(U+0009), 라인 피드(U+000A), 캐리지 리턴(U+000D), 그리고 U+0020 ~ U+00FF 범위의 문자 등 유니코드 문자만 넣을 수 있습니다.

## 조건 지정: 조건 키 사용

AWS는 액세스 제어를 위해 IAM을 지원하는 모든 AWS 서비스에 대해 사전 정의된 조건 키(AWS 차원의 조건 키) 집합을 제공합니다. 예를 들어 aws:SourceIp 조건 키를 사용하여 요청자의 IP 주소를 확인한 후 작업을 수행하도록 허용할 수 있습니다. AWS 차원 키의 전체 목록은 IAM 사용 설명서의 사용 가능한 조건 키를 참조하세요.

다음 표는 DynamoDB에 적용되는 DynamoDB 서비스별 조건 키를 보여 줍니다.

| DynamoDB 조건 키 | 설명 |
| --- | --- |
| dynamodb:LeadingKeys | 테이블의 첫 번째 키 속성, 즉 파티션 키를 나타냅니다. 키 이름 LeadingKeys 는 단일 항목 작업에 사용되는 경우에도 복수형입니다. 또 |

| DynamoDB 조건 키 | 설명 |
| --- | --- |
| | 한 조건에서 ForAllValues 를 사용할 때 LeadingKeys 변경자를 사용해야 합니다. |
| dynamodb:Select | Query 또는 Scan 요청의 Select 파라미터를 나타냅니다. Select는 다음 값 중 하나일 수 있습니다.<br>• ALL_ATTRIBUTES<br>• ALL_PROJECTED_ATTRIBUTES<br>• SPECIFIC_ATTRIBUTES<br>• COUNT |
| dynamodb:Attributes | 요청 내 속성 이름 목록이나 요청에서 반환된 속성 목록을 나타냅니다. Attributes 값은 다음과 같이 특정 DynamoDB API 작업용 파라미터와 동일한 방식으로 이름이 지정되고 동일한 의미를 갖습니다.<br>• AttributesToGet<br>사용처: BatchGetItem, GetItem, Query, Scan<br>• AttributeUpdates<br>사용처: UpdateItem<br>• Expected<br>사용처: DeleteItem, PutItem, UpdateItem<br>• Item<br>사용처: PutItem<br>• ScanFilter<br>사용처: Scan |

| DynamoDB 조건 키 | 설명 |
| --- | --- |
| dynamodb:ReturnValues | 요청의 ReturnValues 파라미터를 나타냅니다. ReturnValues 는 다음 값 중 하나일 수 있습니다.<br>• ALL_OLD<br>• UPDATED_OLD<br>• ALL_NEW<br>• UPDATED_NEW<br>• NONE |
| dynamodb:ReturnConsumedCapacity | 요청의 ReturnConsumedCapacity 파라미터를 나타냅니다. ReturnConsumedCapacity 는 다음 값 중 하나일 수 있습니다.<br>• TOTAL<br>• NONE |

사용자 액세스 제한

여러 IAM 권한 정책을 사용하면 사용자가 테이블에서 해시 키 값이 사용자 식별자와 일치하는 항목만 액세스할 수 있습니다. 예를 들어 앞서 다룬 게임 앱은 이러한 방식으로 액세스를 제한하므로 사용자가 자신의 사용자 ID와 연결된 게임 데이터에만 액세스할 수 있습니다. IAM 치환 변수인 ${www.amazon.com:user_id}, ${graph.facebook.com:id} 및 ${accounts.google.com:sub}에는 Login with Amazon, Facebook 및 Google용 사용자 식별자가 포함되어 있습니다. 애플리케이션이 이러한 ID 공급자 중 하나에 로그인하는 방식에 대해 알아보려면 웹 아이덴티티 페더레이션 사용 단원을 참조하세요.

> **Important**
>
> 글로벌 테이블 복제를 제한하는 데는 세분화된 액세스 제어가 지원되지 않습니다. 글로벌 테이블 복제에 사용되는 DynamoDB 서비스 위탁자 또는 서비스 연결 역할에 세분화된 액세스 제어를 위한 정책 조건을 적용하면 글로벌 테이블 내에서 복제가 중단될 수 있습니다.

> **Note**
>
> 다음 단원의 각 예제에서는 `Effect` 절을 `Allow`로 설정하고 허용할 작업, 리소스 및 파라미터만 지정합니다. 액세스는 IAM 정책에 명시적으로 나열된 항목에만 허용됩니다.
> 경우에 따라, `Effect` 절을 `Deny`로 설정하고 정책의 모든 논리를 반전시켜 거부 기반 정책이 되도록 이러한 정책을 다시 작성할 수도 있습니다. 하지만 허용 기반 정책에 비해 올바르게 작성하기가 어려우므로 DynamoDB에서는 거부 기반 정책을 사용하지 않는 것이 좋습니다. 또한, DynamoDB API에 대한 향후 변경(또는 기존 API 입력에 대한 변경)으로 인해 거부 기반 정책이 무효화될 수도 있습니다.

정책 예: 조건을 사용하여 세부적인 액세스 제어 구현

이 단원에서는 DynamoDB 테이블 및 인덱스에 대한 세분화된 액세스 제어를 구현하기 위한 몇 가지 정책을 보여 줍니다.

> **Note**
>
> 모든 예는 us-west-2 리전을 사용하며 가상의 계정 ID를 포함합니다.

아래 동영상에서는 IAM 정책 조건을 사용하는 DynamoDB의 세분화된 액세스 제어를 설명합니다.

1: 특정 파티션 키 값이 있는 항목에 대한 액세스를 제한하는 권한을 부여

다음 권한 정책은 `GamesScore` 테이블에서 DynamoDB 작업의 집합을 허용하는 권한을 부여합니다. 이 정책은 `dynamodb:LeadingKeys` 조건 키를 사용하여 사용자 작업을 `UserID` 파티션 키 값이 이 앱의 Login with Amazon 고유 사용자 ID와 일치하는 항목으로만 제한합니다.

> **Important**
>
> 작업 목록에는 `Scan` 권한이 포함되지 않는데, `Scan`이 주요 키와 상관없이 모든 항목을 반환하기 때문입니다.

JSON

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"FullAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

> **Note**
>
> 정책 변수를 사용하는 경우 정책에 명시적으로 버전 `2012-10-17`을 지정해야 합니다. 액세스 정책 언어의 기본 버전 `2008-10-17`은 정책 변수를 지원하지 않습니다.

읽기 전용 액세스를 구현하려면 데이터를 수정할 수 있는 작업을 제거할 수 있습니다. 다음 정책에서는 읽기 전용 액세스를 제공하는 작업만 조건에 포함됩니다.

JSON

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ReadOnlyAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

> **Important**
>
> dynamodb:Attributes를 사용하는 경우 정책에 나열된 테이블 및 보조 인덱스에 대한 모든 기본 키 및 인덱스 키 속성의 이름을 지정해야 합니다. 그렇지 않으면, DynamoDB에서 이러한 키 속성을 사용하여 요청된 작업을 수행할 수 없습니다.

2: 테이블의 특정 속성에 대한 액세스를 제한하는 권한을 부여

다음 권한 정책은 dynamodb:Attributes 조건 키를 추가하여 테이블에서 2개의 속성에만 액세스를 허용합니다. 이러한 속성은 조건부 쓰기 또는 스캔 필터에서 읽거나 쓰거나 평가할 수 있습니다.

JSON

```
{
   "Version":"2012-10-17",
```

```
   "Statement":[
      {
         "Sid":"LimitAccessToSpecificAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem",
            "dynamodb:Scan"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "UserId",
                  "TopScore"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

> **Note**
>
> 이 정책은 이름이 지정된 속성 집합에 대한 액세스를 허용하는 허용 목록 방식을 사용합니다. 또는 이에 상응하는 정책으로 다른 속성에 대한 액세스를 거부하는 정책을 작성할 수 있습니다. 이러한 거부 목록 방식은 사용하지 않는 것이 좋습니다. Wikipedia(http://en.wikipedia.org/wiki/Principle_of_least_privilege)에 설명된 대로 사용자가 최소 권한 원칙에 따라 이러한 거부

되는 속성의 이름을 결정하고 허용 목록 방식을 사용하여 거부되는 속성을 지정하는 대신 허용되는 값을 모두 열거할 수 있습니다.

이 정책에서는 PutItem, DeleteItem 또는 BatchWriteItem을 허용하지 않습니다. 이들 작업은 항상 이전 항목 전체를 대체함으로써 사용자가 액세스할 수 없는 속성의 이전 값을 삭제할 수 있게 허용합니다.

이 권한 정책의 StringEqualsIfExists 절은 다음을 보장합니다.

- 사용자가 Select 파라미터를 지정하는 경우 해당 값은 SPECIFIC_ATTRIBUTES이어야 합니다. 이렇게 하면 API 작업이 인덱스 프로젝션 등의 방법으로 허용되지 않는 속성을 반환할 수 없게 됩니다.
- 사용자가 ReturnValues 파라미터를 지정하는 경우 해당 값은 NONE, UPDATED_OLD 또는 UPDATED_NEW이어야 합니다. 이는 UpdateItem 작업이 항목을 바꾸기 전에 항목이 존재하는지 여부를 확인하기 위해, 또한 요청한 경우 이전 속성 값을 반환할 수 있도록 암시적인 읽기 작업도 수행하기 때문에 필요합니다. 이 방식으로 ReturnValues를 제한하면 사용자가 허용된 속성만 읽거나 쓸 수 있습니다.
- StringEqualsIfExists 절은 허용되는 작업의 컨텍스트에서 요청당 이러한 파라미터 중 하나만 (Select 또는 ReturnValues) 사용할 수 있도록 합니다.

다음은 이 정책에 대한 몇 가지 변형 형태입니다.

- 읽기 작업만 허용하려면 허용된 작업 목록에서 UpdateItem을 제거하면 됩니다. 나머지 작업 중 어떤 것도 ReturnValues를 허용하지 않으므로 조건에서 ReturnValues를 제거할 수 있습니다. 또한 Select 파라미터에는 항상 값(별도로 지정하지 않는 경우 ALL_ATTRIBUTES)이 있으므로 StringEqualsIfExists를 StringEquals로 변경할 수도 있습니다.
- 쓰기 작업만 허용하려면 허용된 작업 목록에서 UpdateItem을 제외한 모든 작업을 제거하면 됩니다. UpdateItem에서는 Select 파라미터를 사용하지 않으므로 조건에서 Select를 제거할 수 있습니다. 또한 ReturnValues 파라미터에는 항상 값(별도로 지정하지 않는 경우 NONE)이 있으므로 StringEqualsIfExists를 StringEquals로 변경해야 합니다.
- 이름이 패턴과 일치하는 모든 속성을 허용하려면 StringLike 대신에 StringEquals를 사용하고 다중 문자 패턴 와일드카드(*)를 사용합니다.

3: 특정 속성의 업데이트를 금지하는 권한을 부여

다음 권한 정책은 사용자 액세스를 dynamodb:Attributes 조건 키로 식별되는 특정 속성만 업데이트하도록 제한합니다. StringNotLike 조건은 애플리케이션이 dynamodb:Attributes 조건 키를 사용하여 지정된 속성을 업데이트하는 것을 금지합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"PreventUpdatesOnCertainAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
         "Condition":{
            "ForAllValues:StringNotLike":{
               "dynamodb:Attributes":[
                  "FreeGamesAvailable",
                  "BossLevelUnlocked"
               ]
            },
            "StringEquals":{
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

다음을 참조하세요.

- UpdateItem 작업은 다른 쓰기 작업과 마찬가지로 업데이트 전후의 값을 반환할 수 있도록 항목에 대한 읽기 액세스가 필요합니다. 이 정책에서는 dynamodb:ReturnValues 조건 키를 지정하여 작

업이 업데이트를 허용할 속성에만 액세스하도록 제한합니다. 조건 키는 요청에서 ReturnValues를 제한하여 NONE, UPDATED_OLD 또는 UPDATED_NEW만 지정하며 ALL_OLD 또는 ALL_NEW를 포함하지 않습니다.

- PutItem 및 DeleteItem 작업은 전체 항목을 대체하므로 애플리케이션이 어떤 속성도 수정할 수 있습니다. 그러므로 애플리케이션을 특정 속성만 업데이트하도록 허용할 경우 이러한 API에 대한 권한을 부여하면 안 됩니다.

4: 인덱스에서 프로젝션 속성만 쿼리하는 권한을 부여

다음 권한 정책은 dynamodb:Attributes 조건 키를 사용하여 보조 인덱스(TopScoreDateTimeIndex)에 대한 쿼리를 허용합니다. 또한 이 정책은 인덱스로 프로젝션된 특정 속성만 요청하도록 쿼리를 제한합니다.

또한 이 정책은 애플리케이션이 쿼리에서 속성 목록을 지정하도록 하기 위해 dynamodb:Select 조건 키를 지정하여 DynamoDB Query 작업의 Select 파라미터가 SPECIFIC_ATTRIBUTES가 되도록 요구합니다. 속성의 목록은 dynamodb:Attributes 조건 키를 사용하여 제공되는 특정 목록으로 제한됩니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"QueryOnlyProjectedIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "TopScoreDateTime",
                  "GameTitle",
                  "Wins",
                  "Losses",
```

```
                  "Attempts"
               ]
            },
            "StringEquals":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

다음 권한 정책은 비슷하지만 쿼리가 인덱스로 프로젝션된 모든 속성을 요청해야 합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"QueryAllIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:Select":"ALL_PROJECTED_ATTRIBUTES"
            }
         }
      }
   ]
}
```

5: 액세스를 특정 속성 및 파티션 키 값으로 제한하는 권한을 부여

다음 권한 정책은 테이블 및 테이블 인덱스(Resource 요소로 지정)에서 특정 DynamoDB 작업(Action 요소로 지정)을 허용합니다. 이 정책은 dynamodb:LeadingKeys 조건 키를 사용하여 파티션 키 값이 사용자의 Facebook ID와 일치하는 항목으로만 권한을 제한합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"LimitAccessToCertainAttributesAndKeyValues",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${graph.facebook.com:id}"
               ],
               "dynamodb:Attributes":[
                  "attribute-A",
                  "attribute-B"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
```

```
            }
        }
    ]
}
```

다음을 참조하세요.

- 정책(UpdateItem)에서 허용하는 쓰기 작업은 attribute-A 또는 attribute-B만 수정할 수 있습니다.
- 이 정책은 UpdateItem을 허용하므로 애플리케이션이 새 항목을 삽입할 수 있으며 숨겨진 속성은 새 항목에서 null이 됩니다. 이러한 속성이 TopScoreDateTimeIndex로 프로젝션되면 이 정책은 쿼리로 인한 테이블에서의 가져오기를 방지하는 부가적인 이점도 얻게 됩니다.
- 애플리케이션은 dynamodb:Attributes에 나열되지 않은 속성은 읽을 수 없습니다. 이 정책을 사용하면 애플리케이션은 읽기 요청에서 Select 파라미터를 SPECIFIC_ATTRIBUTES로 설정해야 하며, 허용 목록의 속성만 요청할 수 있습니다. 쓰기 요청의 경우, 애플리케이션은 ReturnValues를 ALL_OLD 또는 ALL_NEW로 설정할 수 없으며 다른 속성에 근거하여 조건부 쓰기 작업을 수행할 수 없습니다.

## 관련 주제

- Amazon DynamoDB의 Identity and Access Management
- DynamoDB API 권한: 작업, 리소스 및 조건 참조

## 웹 아이덴티티 페더레이션 사용

대규모 사용자를 대상으로 하는 애플리케이션을 작성하려는 경우 필요에 따라 인증 및 권한 부여를 위해 웹 자격 증명 연동 기능을 사용할 수 있습니다. 웹 자격 증명 연동을 사용하면 개별 사용자를 생성할 필요가 없습니다. 그 대신에 사용자는 자격 증명 공급자에 로그인한 후 AWS Security Token Service(AWS STS)에서 임시 보안 자격 증명을 얻을 수 있습니다. 그러면 앱은 이러한 자격 증명을 사용하여 AWS 서비스에 액세스할 수 있습니다.

웹 자격 증명 연동은 다음 자격 증명 공급자를 지원합니다.

- Login with Amazon
- Facebook
- Google

### 웹 ID 페더레이션 관련 추가 리소스

다음 리소스는 웹 ID 페더레이션에 대해 자세히 알아보는 데 도움이 됩니다.

- AWS 개발자 블로그의 게시글 AWS SDK for .NET을 사용한 웹 자격 증명 연동에서는 Facebook에서 웹 자격 증명 연동을 사용하는 방법을 안내합니다. 여기에는 웹 자격 증명으로 IAM 역할을 수임하는 방법과 임시 보안 자격 증명을 사용해 AWS 리소스에 액세스하는 방법을 보여주는 C# 코드 조각이 포함되어 있습니다.
- AWS Mobile SDK for iOS 및 AWS Mobile SDK for Android에는 샘플 앱이 포함되어 있습니다. 이러한 앱에는 자격 증명 공급자를 호출하는 방법과 이러한 공급자의 정보를 사용하여 임시 보안 자격 증명을 가져오고 사용하는 방법을 보여 주는 코드가 포함되어 있습니다.
- 모바일 애플리케이션을 사용한 웹 자격 증명 연동 항목에서는 웹 자격 증명 연동에 대해 설명하며, 웹 자격 증명 연동을 사용하여 AWS 리소스에 액세스하는 방법의 예를 보여 줍니다.

### 웹 아이덴티티 페더레이션에 대한 정책 예

DynamoDB에서 웹 자격 증명 연동을 사용하는 방법을 보려면 IAM 정책 조건을 사용하여 세분화된 액세스 제어 구현에 소개된 GameScores 테이블을 다시 살펴보세요. 다음은 GameScores의 기본 키입니다.

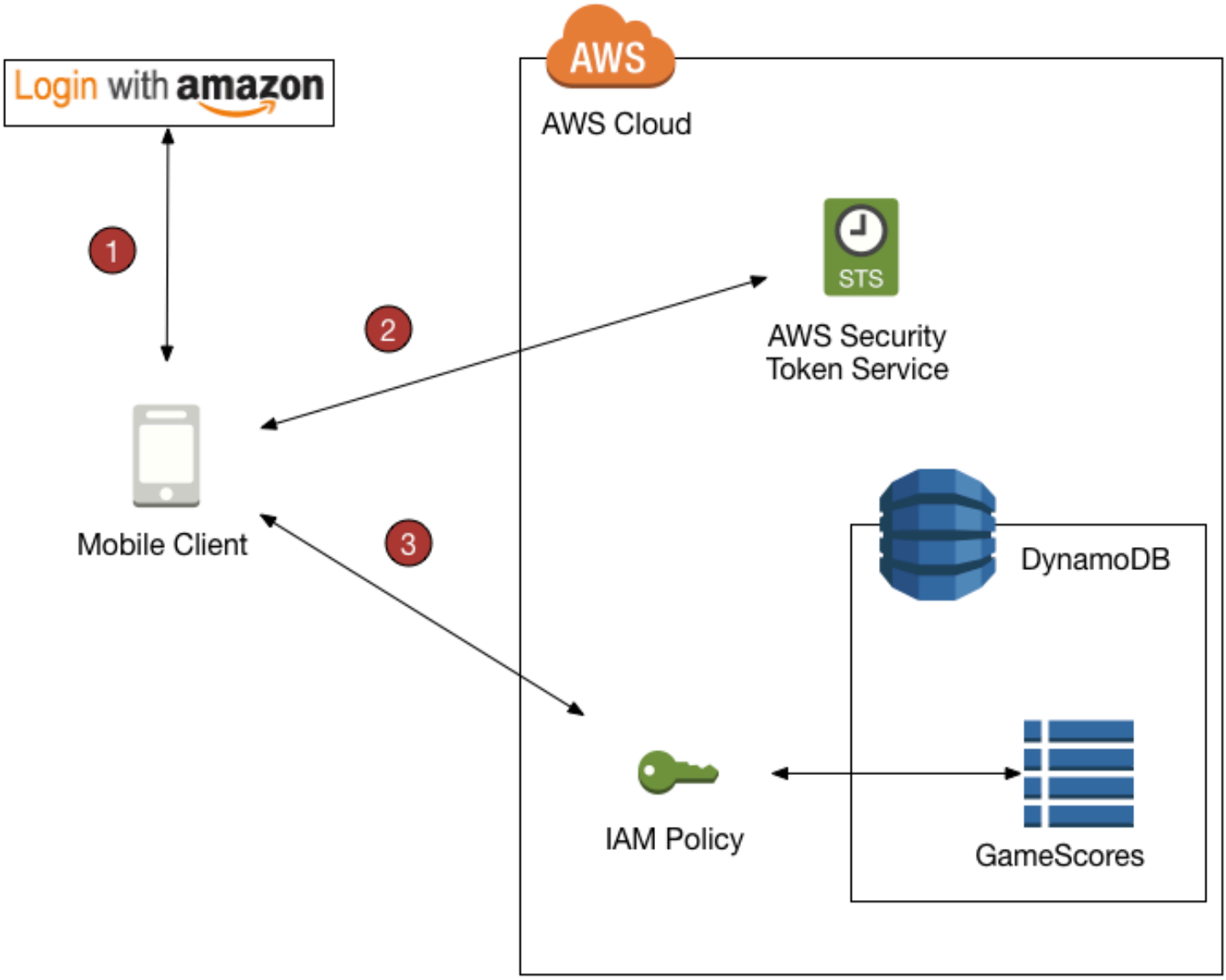
| 테이블 이름 | 기본 키 속성 | 파티션 키 이름 및 유형 | 정렬 키 이름 및 유형 |
|---|---|---|---|
| GameScores(UserId, GameTitle, ...) | 복합 | 속성 이름: UserId<br>유형: 문자열 | 속성 이름: GameTitle<br>유형: 문자열 |

이번에는 모바일 게임 앱에서 이 테이블을 사용하며 해당 앱에서 수천 명 또는 수만 명의 사용자를 지원해야 한다고 가정해 보겠습니다. 이 규모에서는 개별 앱 사용자를 관리하고 각 사용자가 GameScores 테이블에서 자신의 고유 데이터에만 액세스할 수 있도록 하는 것이 매우 어렵습니다. 다행히 많은 사용자가 이미 Facebook, Google 또는 Login with Amazon 같은 타사 자격 증명 공급자의 계정을 이미 보유하고 있습니다. 따라서 인증 작업에 대해 이러한 공급자 중 하나를 활용할 수 있습니다.

웹 자격 증명 연동을 사용하여 이 작업을 수행하려면 앱 개발자가 자격 증명 공급자(예: Login with Amazon)에 앱을 등록하고 고유한 앱 ID를 받아야 합니다. 그러고 나면 개발자는 IAM 역할을 생성해야 합니다. (이 예에서 이 역할의 이름은 GameRole입니다.) 이 역할에는 앱이 GameScores 테이블에 액세스할 수 있는 조건을 지정하는 IAM 정책 문서가 연결되어 있어야 합니다.

게임을 플레이하려는 사용자는 해당 게임 앱에서 자신의 Login with Amazon 계정에 로그인해야 합니다. 그런 다음 앱은 Login with Amazon 앱 ID를 지정하고 GameRole의 멤버십을 요청하여 AWS Security Token Service(AWS STS)를 호출합니다. AWS STS는 임시 AWS 자격 증명을 앱에 반환하여 앱이 GameRole 정책 문서에 따라 GameScores 테이블에 액세스하도록 허용합니다.

다음 다이어그램은 이러한 각 단계가 어떻게 서로 연결되는지 보여 줍니다.




웹 아이덴티티 페더레이션 개요

1. 앱은 타사 자격 증명 공급자를 호출하여 사용자와 앱을 인증합니다. 자격 증명 공급자가 웹 자격 증명 토큰을 앱에 반환합니다.

2. 앱이 AWS STS를 호출하고 웹 자격 증명 토큰을 입력으로 전달합니다. AWS STS는 앱에 권한을 부여하고 임시 AWS 액세스 자격 증명을 제공합니다. 앱은 IAM 역할(GameRole)을 수임하고 역할의 보안 정책에 따라 AWS 리소스에 액세스할 수 있습니다.
3. 앱은 DynamoDB를 호출하여 GameScores 테이블에 액세스합니다. GameRole을 위임했으므로 앱은 해당 역할과 연결된 보안 정책을 준수해야 합니다. 정책 문서는 앱이 사용자와 관련이 없는 데이터에 액세스할 수 없도록 합니다.

다시 한번 다음은 IAM 정책 조건을 사용하여 세분화된 액세스 제어 구현에 나온 GameRole의 보안 정책입니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
```

```
                "TopScoreDateTime"
            ]
         },
         "StringEqualsIfExists":{
            "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
         }
      }
   }
]
}
```

Condition 절은 앱에 표시되는 GameScores의 항목을 결정합니다. 이 작업은 Login with Amazon ID를 GameScores의 UserId 파티션 키 값과 비교하여 수행됩니다. 이 정책에 나열된 DynamoDB 작업 중 하나를 통해서는 현재 사용자에게 속한 항목만 처리할 수 있습니다. 테이블의 다른 항목에는 액세스할 수 없습니다. 또한 정책에 나열된 특정 속성만 액세스할 수 있습니다.

웹 아이덴티티 페더레이션을 사용하기 위한 준비

애플리케이션 개발자이며 앱에 대해 웹 자격 증명 연동을 사용하려면 다음 단계를 따릅니다.

1. 타사 자격 증명 공급자에 개발자로 가입합니다. 다음 외부 링크는 지원되는 자격 증명 공급 가입에 대한 정보를 제공합니다.
   - Login with Amazon 개발자 센터
   - Facebook 사이트의 Registration
   - Google 사이트의 Using OAuth 2.0 to Access Google APIs
2. 자신의 앱을 자격 증명 공급자에 등록합니다. 이 작업을 수행하면 공급자가 앱에 고유한 ID를 제공합니다. 앱이 여러 자격 증명 공급자에서 작동하도록 하려면 각 공급자로부터 앱 ID를 얻어야 합니다.
3. 하나 이상의 IAM 역할을 생성합니다.  각 앱에서는 자격 증명 공급자마다 한 개의 역할이 필요합니다 예를 들면 사용자가 Login with Amazon을 사용하여 로그인한 앱에 의해 위임될 수 있는 역할, 사용자가 Facebook을 사용하여 로그인한 동일 앱에 대한 두 번째 역할, 그리고 사용자가 Google을 사용하여 로그인하는 앱에 대한 세 번째 역할을 생성할 수 있습니다.

   역할 생성 과정의 일부로 IAM 정책을 해당 역할에 연결해야 합니다. 정책 문서에 앱이 필요로 하는 DynamoDB 리소스와, 이러한 리소스에 액세스할 수 있는 권한을 정의해야 합니다.

자세한 내용은 IAM 사용 설명서의 웹 자격 증명 연동 정보 단원을 참조하세요.

> **Note**
>
> AWS Security Token Service 대신 Amazon Cognito를 사용할 수 있습니다. Amazon Cognito는 모바일 앱의 임시 자격 증명을 관리하는 기본 서비스입니다. 자세한 내용은 Amazon Cognito 개발자 안내서의 자격 증명 얻기를 참조하세요.

### DynamoDB 콘솔을 사용하여 IAM 정책 생성

DynamoDB 콘솔은 웹 자격 증명 연동에 사용할 IAM 정책을 생성하는 데 도움이 됩니다. 정책을 생성하려면 DynamoDB 테이블을 선택하고 정책에 포함할 자격 증명 공급자, 작업 및 속성을 지정해야 합니다. 그러면 DynamoDB 콘솔에서 IAM 역할에 연결할 수 있는 정책이 생성합니다.

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 탐색 창에서 테이블을 선택합니다.
3. 테이블 목록에서 IAM 정책을 생성할 테이블을 선택합니다.
4. 작업 버튼을 클릭하고 액세스 제어 정책 생성을 선택합니다.
5. 정책에 대해 ID 공급자, 작업 및 속성을 선택합니다.

   원하는 대로 설정되었으면 정책 생성을 선택합니다. 생성된 정책이 표시됩니다.
6. 설명서 참조를 선택하고 생성된 정책을 IAM 역할에 연결하는 데 필요한 단계를 따릅니다.

### 웹 아이덴티티 페더레이션을 사용할 앱 작성

웹 자격 증명 연동을 사용하려면 사용자가 생성한 IAM 역할을 앱이 수임해야 합니다. 그 시점 이후로 앱은 사용자가 역할에 연결한 액세스 정책을 준수합니다.

런타임 시 앱에 웹 자격 증명 연동이 사용되는 경우 다음 단계를 따라야 합니다.

1. 타사 자격 증명 공급자를 사용하여 인증합니다. 앱은 제공되는 인터페이스를 사용하여 자격 증명 공급자를 호출해야 합니다. 사용자를 인증하는 정확한 방법은 앱이 실행 중인 플랫폼과 공급자에 따라 달라집니다. 일반적으로 사용자가 아직 로그인하지 않은 경우 자격 증명 공급자가 해당 공급자의 로그인 페이지 표시를 처리합니다.

   자격 증명 공급자는 사용자를 인증한 후 웹 자격 증명 토큰을 앱에 반환합니다. 이 토큰의 형식은 공급자마다 다르지만, 일반적으로는 매우 긴 문자열 형식입니다.

2. 임시 AWS 보안 자격 증명을 얻습니다. 이렇게 하려면 앱이 AssumeRoleWithWebIdentity 요청을 AWS STS(AWS Security Token Service)로 보냅니다. 이 요청은 다음을 포함합니다.
   - 이전 단계의 웹 자격 증명 토큰
   - 자격 증명 공급자에게서 받은 앱 ID
   - 이 앱의 자격 증명 공급자에 대해 생성한 IAM 역할의 Amazon 리소스 이름(ARN)

   AWS STS는 일정 시간(기본적으로 3600초)이 지나면 만료되는 AWS 보안 자격 증명 집합을 반환합니다.

   다음은 샘플 요청과 AWS STS의 AssumeRoleWithWebIdentity 작업에서 받은 응답입니다. 웹 자격 증명 토큰은 Login with Amazon 자격 증명 공급자로부터 얻었습니다.

```
GET / HTTP/1.1
Host: sts.amazonaws.com
Content-Type: application/json; charset=utf-8
URL: https://sts.amazonaws.com/?ProviderId=www.amazon.com
&DurationSeconds=900&Action=AssumeRoleWithWebIdentity
&Version=2011-06-15&RoleSessionName=web-identity-federation
&RoleArn=arn:aws:iam::123456789012:role/GameRole
&WebIdentityToken=Atza|IQEBLjAsAhQluyKqyBiYZ8-kclvGTYM81e...(remaining characters
 omitted)
```

```
<AssumeRoleWithWebIdentityResponse
  xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
  <AssumeRoleWithWebIdentityResult>
    <SubjectFromWebIdentityToken>amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE</
SubjectFromWebIdentityToken>
    <Credentials>
      <SessionToken>AQoDYXdzEMf//////////wEa8AP6nNDwcSLnf+cHupC...(remaining
 characters omitted)</SessionToken>
      <SecretAccessKey>8Jhi60+EWUUbbUShTEsjTxqQtM8UKvsM6XAjdA==</SecretAccessKey>
      <Expiration>2013-10-01T22:14:35Z</Expiration>
      <AccessKeyId>06198791C436IEXAMPLE</AccessKeyId>
    </Credentials>
    <AssumedRoleUser>
      <Arn>arn:aws:sts::123456789012:assumed-role/GameRole/web-identity-federation</
Arn>
      <AssumedRoleId>AROAJU4SA2VW5SZRF2YMG:web-identity-federation</AssumedRoleId>
    </AssumedRoleUser>
  </AssumeRoleWithWebIdentityResult>
```

```
  <ResponseMetadata>
    <RequestId>c265ac8e-2ae4-11e3-8775-6969323a932d</RequestId>
  </ResponseMetadata>
</AssumeRoleWithWebIdentityResponse>
```

3. AWS 소스에 액세스합니다. AWS STS의 응답에는 앱이 DynamoDB 리소스에 액세스하는 데 필요한 정보가 포함되어 있습니다.
   - `AccessKeyID`, `SecretAccessKey` 및 `SessionToken` 필드에는 이 사용자와 이 앱에만 유효한 보안 자격 증명이 들어 있습니다.
   - `Expiration` 필드는 이러한 자격 증명이 유효 상태로 유지되는 제한 시간을 지정합니다.
   - `AssumedRoleId` 필드에는 앱에 의해 수임된 세션별 IAM 역할의 이름이 포함되어 있습니다. 앱은 이 세션 기간 동안 IAM 정책 문서의 액세스 제어를 준수합니다.
   - `SubjectFromWebIdentityToken` 필드에는 이 특정 자격 증명 공급자에 대한 IAM 정책 변수에 표시되는 고유 ID가 포함되어 있습니다. 다음은 지원되는 공급자에 대한 IAM 정책 변수이며, 몇 가지 예를 들면 다음과 같습니다.

| 정책 변수 | 예제 값 |
| --- | --- |
| `${www.amazon.com:user_id}` | `amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE` |
| `${graph.facebook.com:id}` | `123456789` |
| `${accounts.google.com:sub}` | `12345678901234567890 1` |

이러한 정책 변수가 사용되는 IAM 정책의 예는 정책 예: 조건을 사용하여 세부적인 액세스 제어 구현 단원을 참조하세요.

AWS STS가 임시 액세스 자격 증명을 생성하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 임시 보안 자격 증명 요청 단원을 참조하세요.

## DynamoDB API 권한: 작업, 리소스 및 조건 참조

Amazon DynamoDB의 Identity and Access Management를 설정하고 IAM 자격 증명에 연결할 수 있는 권한 정책(자격 증명 기반 정책)을 작성할 때 IAM 사용 설명서에서 Amazon DynamoDB에 사용되는 작업, 리소스 및 조건 키 목록을 참조로 사용할 수 있습니다. 해당 페이지에는 각 DynamoDB API 작업, 작업을 수행할 권한을 부여할 수 있는 해당 작업, 권한을 부여할 수 있는 대상 AWS 리소스가 나열되어 있

습니다. 정책의 Action필드에서 작업을 지정하고, 정책의 Resource필드에서 리소스 값을 지정합니다.

DynamoDB 정책에서 AWS 차원 조건 키를 사용하여 조건을 표현할 수 있습니다. AWS 차원 키의 전체 목록은 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

AWS 차원 조건 키 이외에도 DynamoDB에는 조건에 사용할 수 있는 고유한 특정 키가 있습니다. 자세한 내용은 IAM 정책 조건을 사용하여 세분화된 액세스 제어 구현 섹션을 참조하세요.

관련 주제

- Amazon DynamoDB의 Identity and Access Management
- IAM 정책 조건을 사용하여 세분화된 액세스 제어 구현

# DynamoDB에 대한 업종별 규정 준수 확인

AWS 서비스가 특정 규정 준수 프로그램의 범위에 포함되는지 알아보려면 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하고 관심 있는 규정 준수 프로그램을 선택하세요. 일반적인 정보는 AWS 규정 준수 프로그램을 참조하세요.

AWS Artifact(을)를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 AWS Artifact에서 보고서 다운로드를 참조하세요.

AWS 서비스 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다.AWS에서는 규정 준수를 지원할 다음과 같은 리소스를 제공합니다.

- 보안 규정 준수 및 거버넌스 - 이러한 솔루션 구현 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수 기능을 배포하는 단계를 제공합니다.
- HIPAA 적격 서비스 참조 - HIPAA 적격 서비스가 나열되어 있습니다. 모든 AWS 서비스에 HIPAA 자격이 있는 것은 아닙니다.
- AWS 규정 준수 리소스 - 고객 조직이 속한 산업 및 위치에 적용될 수 있는 워크북 및 가이드 컬렉션입니다.
- AWS 고객 규정 준수 가이드 - 규정 준수의 관점에서 공동 책임 모델을 이해합니다. 이 가이드에서는 AWS 서비스를 보호하기 위한 모범 사례를 요약하고 여러 프레임워크(미국 표준 기술 연구소(NIST), 결제 카드 산업 보안 표준 위원회(PCI), 국제 표준화기구(ISO) 등)에서 보안 컨트롤에 대한 지침을 매핑합니다.
- AWS Config 개발자 가이드의 규칙을 사용하여 리소스 평가 – AWS Config 서비스는 내부 사례, 산업 지침 및 규제에 대한 리소스 구성의 준수 상태를 평가합니다.

- AWS Security Hub – 이 AWS 서비스(은)는 AWS 내 보안 상태에 대한 포괄적인 보기를 제공합니다. Security Hub는 보안 컨트롤을 사용하여 AWS리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하세요.
- Amazon GuardDuty - 이 AWS 서비스는 의심스럽고 악의적인 활동이 있는지 환경을 모니터링하여 AWS 계정, 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 탐지합니다. GuardDuty는 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 따르는 데 도움을 줄 수 있습니다.
- AWS Audit Manager - 이 AWS 서비스은(는) AWS 사용을 지속해서 감사하여 위험을 관리하고 규정 및 업계 표준을 준수하는 방법을 간소화할 수 있도록 지원합니다.

# Amazon DynamoDB의 복원성 및 재해 복구

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 리전을 제공하며 이러한 가용 리전은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

더 먼 지역적 거리를 두고 데이터 또는 애플리케이션을 복제해야 하는 경우 AWS 로컬 리전을 사용하세요. AWS 로컬 리전은 기존 AWS 리전을 보완하기 위해 설계된 단일 데이터 센터입니다. 모든 AWS 리전과 마찬가지로 AWS 로컬 리전은 다른 AWS 리전과 완벽히 격리되어 있습니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

AWS 글로벌 인프라 외에도 Amazon DynamoDB는 데이터 복원성과 백업 요구 사항을 지원하는 다양한 기능을 제공합니다.

온디맨드 백업 및 복원

DynamoDB는 온디맨드 백업 기능을 제공합니다. 따라서 테이블의 전체 백업을 생성하여 장기간 유지하고 보관할 수 있습니다. 자세한 내용은 DynamoDB에 대한 온디맨드 백업 및 복원을 참조하세요.

특정 시점으로 복구

특정 시점으로 복구를 사용하면 우발적인 쓰기 또는 삭제 작업으로부터 DynamoDB 테이블을 보호할 수 있습니다. 특정 시점으로 복구를 설정해 두면 온디맨드 백업의 생성, 유지 관리, 예약을 걱정할 필요가 없습니다. 자세한 내용은 DynamoDB의 특정 시점으로 복구를 참조하세요.

여러 AWS 리전에서 동기화되는 글로벌 테이블

DynamoDB는 테이블의 데이터와 트래픽을 충분한 수의 서버로 자동 분산하여 처리량 및 스토리지 요구 사항을 처리하면서도 일관되고 빠른 성능을 유지합니다. 모든 데이터가 SSD(Solid State Disk)에 저장되고 AWS 리전의 여러 가용 영역에 걸쳐 자동 복제되기 때문에 확실한 고가용성과 데이터 내구성을 제공합니다. 전역 테이블을 사용하여 AWS 리전 간에 DynamoDB 테이블을 동기화할 수 있습니다.

# Amazon DynamoDB의 인프라 보안

관리형 서비스인 Amazon DynamoDB는 AWS Well-Architected Framework의 인프라 보호에 설명된 AWS 글로벌 네트워크 보안 절차로 보호됩니다.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 DynamoDB에 액세스합니다. 클라이언트는 TLS(전송 계층 보안) 버전 1.2 또는 1.3을 사용할 수 있습니다. 클라이언트는 DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Diffie-Hellman Ephemeral)와 같은 PFS(전달 완전 보안)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다. 또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

DynamoDB에 대한 Virtual Private Cloud(VPC) 엔드포인트를 사용하면 VPC의 Amazon EC2 인스턴스가 퍼블릭 인터넷에 노출되지 않고도 프라이빗 IP 주소를 사용해 DynamoDB에 액세스할 수 있게 할 수 있습니다. 자세한 내용은 Amazon VPC 엔드포인트를 사용하여 DynamoDB에 액세스 섹션을 참조하세요.

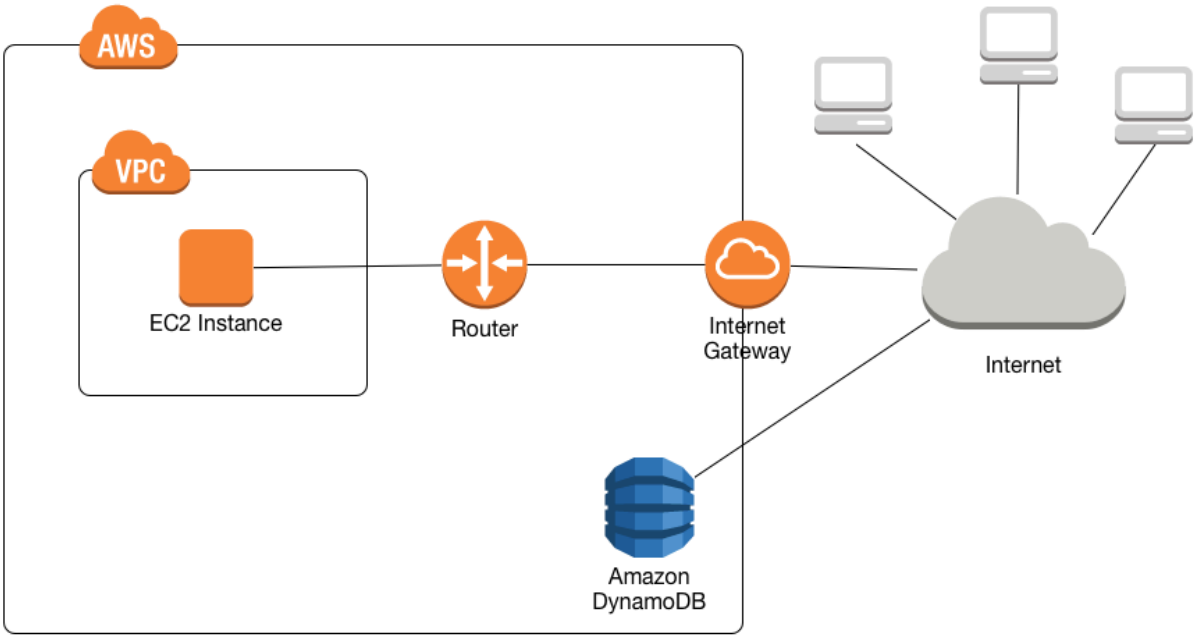
## Amazon VPC 엔드포인트를 사용하여 DynamoDB에 액세스

보안상의 이유로 많은 AWS 고객은 Amazon Virtual Private Cloud 환경(Amazon VPC)에서 애플리케이션을 실행합니다. Amazon VPC를 통해 Amazon EC2 인스턴스를 퍼블릭 인터넷을 비롯한 다른 네트워크와 논리적으로 분리된 Virtual Private Cloud로 시작할 수 있습니다. Amazon VPC를 사용하면 IP 주소 범위, 서브넷, 라우팅 테이블, 네트워크 게이트웨이 및 보안 설정을 통제할 수 있습니다.

> **Note**
>
> 2013년 12월 4일 이후 AWS 계정을 생성했다면 각 AWS 리전에 기본 VPC가 갖추어져 있습니다. 기본 VPC는 바로 사용할 수 있으므로 별도의 구성 단계 없이 즉시 사용할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 기본 VPC 및 기본 서브넷을 참조하세요.

퍼블릭 인터넷에 액세스하려면 VPC를 인터넷에 연결하는 가상 라우터인 인터넷 게이트웨이가 VPC에 있어야 합니다. 이를 통해 VPC의 Amazon EC2에서 실행되는 애플리케이션이 Amazon DynamoDB와 같은 인터넷 리소스에 액세스할 수 있습니다.

기본적으로 DynamoDB와의 통신은 SSL/TLS 암호화를 사용하여 네트워크 트래픽을 보호하는 HTTPS 프로토콜을 사용합니다. 다음 다이어그램은 DynamoDB가 VPC 엔드포인트 대신 인터넷 게이트웨이를 사용하도록 하여 DynamoDB에 액세스하는 VPC의 Amazon EC2 인스턴스를 보여줍니다.



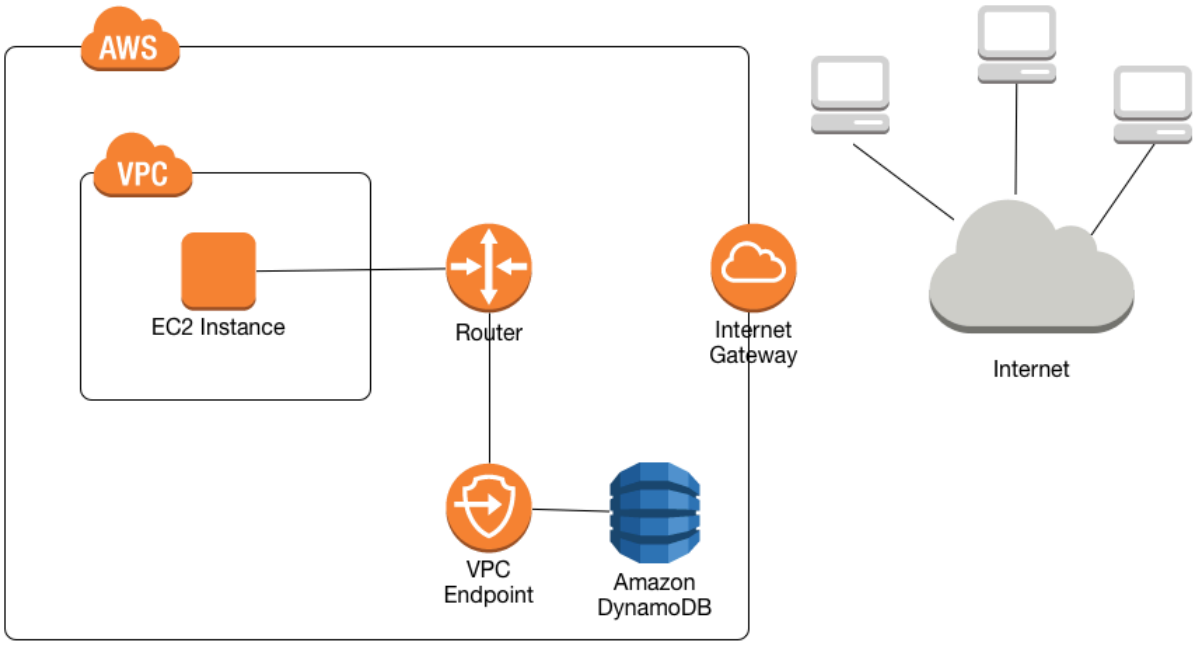

많은 고객이 퍼블릭 인터넷을 통해 데이터를 주고받는 것에 대한 합법적인 개인 정보 보호 및 보안 문제를 우려합니다. 가상 사설 네트워크(VPN)를 사용하여 모든 DynamoDB 네트워크 트래픽을 자체 기업 네트워크 인프라를 통해 라우팅함으로써 이러한 문제를 해결할 수 있습니다. 그러나 이러한 접근 방식은 대역폭 및 가용성 문제를 일으킬 수 있습니다.

DynamoDB에 대한 VPC 엔드포인트는 이러한 문제를 완화할 수 있습니다. DynamoDB에 대한 VPC 엔드포인트를 사용하면 VPC의 Amazon EC2 인스턴스가 퍼블릭 인터넷에 노출되지 않고도 프라이빗 IP 주소를 사용해 DynamoDB에 액세스할 수 있습니다. EC2 인스턴스에 퍼블릭 IP 주소를 지정할 필요가 없으며 VPC에서 인터넷 게이트웨이, NAT 디바이스 또는 가상 프라이빗 게이트웨이가 필요 없습니다. 엔드포인트 정책을 사용하여 DynamoDB에 대한 액세스를 제어합니다. VPC와 AWS 서비스 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다.

> **Note**
>
> 퍼블릭 IP 주소를 사용하는 경우에도 AWS에 호스팅된 인스턴스와 서비스 간의 모든 VPC 통신은 AWS 네트워크 내에서 프라이빗 상태를 유지합니다. AWS 네트워크에 목적지가 있는 AWS 네트워크에서 시작된 패킷은 AWS 중국 리전으로 오가는 트래픽을 제외하고 AWS 글로벌 네트워크에 남아 있습니다.

DynamoDB에 대한 VPC 엔드포인트를 생성하면 리전 내의 DynamoDB 엔드포인트(예: dynamodb.us-west-2.amazonaws.com)에 대한 모든 요청이 Amazon 네트워크 내의 프라이빗 DynamoDB 종단점으로 라우팅됩니다. VPC의 EC2 인스턴스에서 실행 중인 애플리케이션을 수정할 필요가 없습니다. 엔드포인트 이름은 변함이 없지만 DynamoDB로의 경로는 전적으로 Amazon 네트워크 내에 유지되며 퍼블릭 인터넷에 액세스하지 않습니다.

다음 다이어그램은 VPC의 EC2 인스턴스가 DynamoDB에 액세스하기 위해 VPC 엔드포인트를 사용하는 방법을 보여 줍니다.




자세한 내용은 the section called “자습서: DynamoDB에 대한 VPC 엔드포인트 사용” 섹션을 참조하세요.

## Amazon VPC 엔드포인트 및 DynamoDB 공유

VPC 서브넷의 게이트웨이 엔드포인트를 통해 DynamoDB 서비스에 액세스할 수 있으려면 해당 VPC 서브넷에 대한 소유자 계정 권한이 있어야 합니다.

VPC 서브넷의 게이트웨이 엔드포인트에 DynamoDB에 대한 액세스 권한이 부여되면 해당 서브넷에 액세스할 수 있는 모든 AWS 계정이 DynamoDB를 사용할 수 있습니다. 즉, VPC 서브넷 내의 모든 계정 사용자는 액세스 권한이 있는 모든 DynamoDB 테이블을 사용할 수 있습니다. 여기에는 VPC 서브넷과는 다른 계정에 연결된 DynamoDB 테이블이 포함됩니다. VPC 서브넷 소유자는 재량에 따라 서브넷 내의 특정 사용자가 게이트웨이 엔드포인트를 통해 DynamoDB 서비스를 사용하는 것을 제한할 수 있습니다.

## 자습서: DynamoDB에 대한 VPC 엔드포인트 사용

이 단원에서는 DynamoDB에 대해 VPC 엔드포인트를 설정하고 사용하는 절차를 살펴봅니다.

주제


- 1단계: Amazon EC2 인스턴스 시작
- 2단계: Amazon EC2 인스턴스 구성
- 3단계: DynamoDB에 대한 VPC 엔드포인트 생성
- 4단계: (선택 사항) 정리


### 1단계: Amazon EC2 인스턴스 시작

이 단계에서는 기본 Amazon VPC에서 Amazon EC2 인스턴스를 시작합니다. 그런 다음 DynamoDB에 대해 VPC 엔드포인트를 생성하고 사용할 수 있습니다.

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 인스턴스 시작을 선택하고 다음을 수행합니다.

   1단계: Amazon 머신 이미지(AMI) 선택

   - AMI 목록 맨 위에서 [Amazon Linux AMI]로 이동한 후 [Select]를 선택합니다.

   2단계: 인스턴스 유형 선택

   - 인스턴스 유형 목록 맨 위에서 [t2.micro]를 선택합니다.
   - [Next: Configure Instance Details]를 선택합니다.

3단계: 인스턴스 세부 정보 구성

- [Network]로 이동한 후 기본 VPC를 선택합니다.

  다음: 스토리지 추가를 선택합니다.

4단계: 스토리지 추가

- [Next: Tag Instance]를 선택하여 이 단계를 건너뜁니다.

5단계: 인스턴스에 태그 지정

- [Next: Configure Security Group]을 선택하여 이 단계를 건너뜁니다.

6단계: 보안 그룹 구성

- 기존 보안 그룹 선택을 선택합니다.
- 보안 그룹 목록에서 기본값을 선택합니다. 이것이 VPC의 기본 보안 그룹입니다.
- [Next: Review and Launch]를 선택합니다.

7단계: 인스턴스 시작 검토

- 시작을 선택합니다.

3. [Select an existing key pair or create a new key pair] 창에서 다음 중 하나를 수행합니다.
   - Amazon EC2 키 페어가 없는 경우 Create a new key pair(새 키 페어 생성)를 선택하고 지침을 따릅니다. 프라이빗 키 파일(.pem 파일)을 다운로드하라는 메시지가 표시됩니다. 이 파일은 나중에 Amazon EC2 인스턴스에 로그인할 때 필요합니다.
   - Amazon EC2 키 페어가 이미 있으면 Select a key pair(키 페어 선택)로 이동하고 목록에서 해당 키 페어를 선택합니다. Amazon EC2 인스턴스에 로그인하려면 프라이빗 키 파일(.pem 파일)이 있어야 합니다.
4. 구성된 키 페어가 있으면 Launch Instances(인스턴스 시작)를 선택합니다.
5. Amazon EC2 콘솔 홈 페이지로 돌아가서 시작한 인스턴스를 선택합니다. 하단 창의 [Description] 탭에서 인스턴스의 [Public DNS]를 확인합니다. 예를 들면 `ec2-00-00-00-00.us-east-1.compute.amazonaws.com`입니다.

이 퍼블릭 DNS 이름은 이 자습서의 다음 단계(2단계: Amazon EC2 인스턴스 구성)에서 필요하므로 기록해 둡니다.

> **Note**
>
> Amazon EC2 인스턴스가 사용 가능한 상태가 되는 데 몇 분 정도 걸립니다. 다음 단계로 이동하기 전에 [Instance State]가 `running`이고 모든 [Status Checks]가 통과되었는지 확인하세요.

### 2단계: Amazon EC2 인스턴스 구성

Amazon EC2 인스턴스를 사용할 수 있는 경우 해당 인스턴스에 로그인하고 첫 사용이 가능한 상태로 준비할 수 있습니다.

> **Note**
>
> 다음 단계에서는 사용자가 Linux를 실행하는 컴퓨터에서 Amazon EC2 인스턴스로 연결되어 있다고 가정합니다. 다른 연결 방법은 Linux 인스턴스용 Amazon EC2 사용 설명서의 Linux 인스턴스에 연결 단원을 참조하세요.

1. Amazon EC2 인스턴스에 대한 인바운드 SSH 트래픽을 승인해야 합니다. 이를 위해 새로운 EC2 보안 그룹을 생성하여 EC2 인스턴스에 할당합니다.

   a. 탐색 창에서 보안 그룹을 선택합니다.

   b. 보안 그룹 생성을 선택합니다. 보안 그룹 생성 창에서 다음을 수행하세요.

   - Security group name(보안 그룹 이름) - 보안 그룹의 이름을 입력합니다. 예: `my-ssh-access`
   - Description(설명) - 보안 그룹에 대한 간단한 설명을 입력합니다.
   - VPC - 기본 VPC를 선택합니다.
   - [Security group rules] 단원에서 [Add Rule]을 선택하고 다음을 수행합니다.
     - Type(유형) - SSH를 선택합니다.
     - Source(소스) - 내 IP를 선택합니다.

   원하는 대로 설정되었으면 [Create]를 선택합니다.

c. 탐색 창에서 인스턴스를 선택합니다.

d. 1단계: Amazon EC2 인스턴스 시작에서 시작한 Amazon EC2 인스턴스를 선택합니다.

e. Actions(작업) --> 네트워킹 --> 보안 그룹 변경을 차례로 선택합니다.

f. [보안 그룹 변경]에서 이 절차에서 앞서 생성한 보안 그룹을 선택합니다(예: `my-ssh-access`). 기존 `default` 보안 그룹도 선택되어야 합니다. 원하는 대로 설정되었으면 [보안 그룹 할당]을 선택합니다.

2. `ssh` 명령을 사용하여 Amazon EC2 인스턴스에 로그인합니다(아래 예 참조).

```
ssh -i my-keypair.pem ec2-user@public-dns-name
```

프라이빗 키 파일(.pem 파일)과 인스턴스의 퍼블릭 DNS 이름을 지정해야 합니다. (1단계: Amazon EC2 인스턴스 시작 단원을 참조하세요.)

로그인 ID는 `ec2-user`입니다. 암호는 필요하지 않습니다.

3. 다음 예제와 같이 AWS 자격 증명을 구성합니다. 메시지가 나타나면 AWS 액세스 키 ID, 비밀 키 및 기본 리전 이름을 입력합니다.

```
aws configure
```

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]:
```

이제 DynamoDB에 대한 VPC 엔드포인트를 생성할 준비가 되었습니다.

3단계: DynamoDB에 대한 VPC 엔드포인트 생성

이 단계에서는 DynamoDB에 대한 VPC 엔드포인트를 생성하고 이 엔드포인트가 작동하는지 테스트합니다.

1. 시작하기 전에 퍼블릭 엔드포인트를 사용하여 DynamoDB와 통신할 수 있는지 확인합니다.

```
aws dynamodb list-tables
```

출력에는 현재 보유하고 있는 DynamoDB 테이블 목록이 표시됩니다. (보유 중인 테이블이 없는 경우 목록이 비어있음).

2. DynamoDB가 현재 AWS 리전에서 VPC 엔드포인트를 생성하기 위해 사용할 수 있는 서비스인지 확인합니다. (명령은 예제 출력에 이어 굵은 텍스트로 표시됩니다.)

```
aws ec2 describe-vpc-endpoint-services
```

```
{
    "ServiceNames": [
        "com.amazonaws.us-east-1.s3",
        "com.amazonaws.us-east-1.dynamodb"
    ]
}
```

예제 출력에서 DynamoDB는 사용할 수 있는 서비스 중 하나이므로 이에 대한 VPC 엔트포인트를 생성할 수 있습니다.

3. VPC 식별자 확인

```
aws ec2 describe-vpcs
```

```
{
    "Vpcs": [
        {
            "VpcId": "vpc-0bbc736e",
            "InstanceTenancy": "default",
            "State": "available",
            "DhcpOptionsId": "dopt-8454b7e1",
            "CidrBlock": "172.31.0.0/16",
            "IsDefault": true
        }
    ]
}
```

예제 출력에서 VPC ID는 `vpc-0bbc736e`입니다.

4. VPC 엔드포인트를 생성합니다. `--vpc-id` 파라미터에 대해 이전 단계에서 VPC ID를 지정합니다. `--route-table-ids` 파라미터를 사용하여 엔드포인트를 라우팅 테이블과 연결합니다.

```
aws ec2 create-vpc-endpoint --vpc-id vpc-0bbc736e --service-name com.amazonaws.us-
east-1.dynamodb --route-table-ids rtb-11aa22bb
```

```
{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [
            "rtb-11aa22bb"
        ],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

5. VPC 엔드포인트를 통해 DynamoDB에 액세스할 수 있는지 확인합니다.

```
aws dynamodb list-tables
```

원하는 경우 DynamoDB에 대한 다른 몇 가지 AWS CLI 명령을 시도할 수 있습니다. 자세한 내용은 AWS CLI 명령 참조설명서를 참조하세요.

4단계: (선택 사항) 정리

이 자습서에서 생성한 리소스를 삭제하려면 다음 절차를 따릅니다.

DynamoDB에 대한 VPC 엔드포인트를 제거하려면 다음을 수행합니다.

1. Amazon EC2 인스턴스에 로그인합니다.
2. VPC 종단점 ID를 결정합니다.

```
aws ec2 describe-vpc-endpoints
```

```
{
    "VpcEndpoint": {
```

```
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

예제 출력에서 VPC 엔드포인트 ID는 `vpce-9b15e2f2`입니다.

3. VPC 엔드포인트를 삭제합니다.

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2
```

```
{
    "Unsuccessful": []
}
```

빈 어레이 `[]`는 성공을 나타냅니다(실패한 요청 없음).

Amazon EC2 인스턴스를 종료하려면

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 탐색 창에서 인스턴스를 선택합니다.
3. Amazon EC2 인스턴스를 선택합니다.
4. Actions(작업), Instance State(인스턴스 상태), Terminate(해지)를 차례로 선택합니다.
5. 확인 창에서 예, 종료합니다.를 선택합니다.

# AWS PrivateLink for DynamoDB

AWS PrivateLink for DynamoDB를 사용하면 Virtual Private Cloud(Amazon VPC)에서 인터페이스 Amazon VPC 엔드포인트(인터페이스 엔드포인트)를 프로비저닝할 수 있습니다. 이러한 엔드포인트는 VPN 및 AWS Direct Connect를 통해 온프레미스에 있는 애플리케이션에서 또는 Amazon VPC 피어링을 통해 다른 AWS 리전에 있는 애플리케이션에서 직접 액세스할 수 있습니다. AWS PrivateLink 및

인터페이스 엔드포인트를 사용하면 애플리케이션에서 DynamoDB로의 프라이빗 네트워크 연결을 단순화할 수 있습니다.

VPC의 애플리케이션은 DynamoDB 작업을 위해 VPC 인터페이스 엔드포인트를 사용하여 DynamoDB와 통신하는 데 퍼블릭 IP 주소가 필요하지 않습니다. 인터페이스 엔드포인트는 Amazon VPC의 서브넷에서 프라이빗 IP 주소가 할당된 하나 이상의 탄력적 네트워크 인터페이스(ENI)로 표시됩니다. 인터페이스 엔드포인트를 통한 DynamoDB에 대한 요청은 Amazon 네트워크에 유지됩니다. 또한, AWS Direct Connect 또는 AWS Virtual Private Network(AWS VPN)을 통해 온프레미스 애플리케이션에서 Amazon VPC의 인터페이스 엔드포인트에 액세스할 수 있습니다. Amazon VPC를 온프레미스 네트워크에 연결하는 방법에 대한 자세한 내용은 AWS Direct Connect 사용 설명서 및 AWS Site-to-Site VPN 사용 설명서를 참조하세요.

인터페이스 엔드포인트에 대한 일반적인 내용은 AWS PrivateLink 설명서의 Interface Amazon VPC endpoints (AWS PrivateLink)를 참조하세요. AWS PrivateLink는 Amazon DynamoDB Streams 엔드포인트에도 지원됩니다. 자세한 내용은 the section called “AWS PrivateLink for DynamoDB Streams” 섹션을 참조하세요.

주제

- Amazon DynamoDB용 Amazon VPC 엔드포인트의 유형
- AWS PrivateLink for Amazon DynamoDB 사용 시 고려 사항
- Amazon VPC 엔드포인트 생성
- Amazon DynamoDB 인터페이스 엔드포인트에 액세스
- DynamoDB 인터페이스 엔드포인트에서 DynamoDB 테이블에 및 제어 API 작업에 액세스
- 온프레미스 DNS 구성 업데이트
- DynamoDB에 대한 Amazon VPC 엔드포인트 정책 생성
- AWS PrivateLink for DynamoDB Streams

## Amazon DynamoDB용 Amazon VPC 엔드포인트의 유형

두 가지 유형의 Amazon VPC 엔드포인트, 즉 게이트웨이 엔드포인트와 인터페이스 엔드포인트(AWS PrivateLink 사용)를 사용하여 Amazon DynamoDB에 액세스할 수 있습니다. 게이트웨이 엔드포인트는 AWS 네트워크를 통해 Amazon VPC에서 DynamoDB에 액세스하기 위해 라우팅 테이블에 지정하는 게이트웨이입니다. 인터페이스 엔드포인트는 프라이빗 IP 주소를 통해 온프레미스의 Amazon VPC 내에서 또는 Amazon VPC 피어링이나 AWS Transit Gateway를 사용하여 다른 AWS 리전의 Amazon VPC에서 DynamoDB로 요청을 라우팅함으로써 게이트웨이 엔드포인트의 기능을 확장합니다. 자세한 내용은 Amazon VPC 피어링이란? 및 Transit Gateway와 Amazon VPC 피어링 비교를 참조하세요.

인터페이스 엔드포인트는 게이트웨이 엔드포인트와 호환됩니다. Amazon VPC에 기존 게이트웨이 엔드포인트가 있는 경우, 동일한 Amazon VPC에서 두 가지 유형의 엔드포인트를 모두 사용할 수 있습니다.

| DynamoDB용 게이트웨이 엔드포인트 | DynamoDB용 인터페이스 엔드포인트 |
|---|---|
| 두 경우 모두, 네트워크 트래픽은 AWS 네트워크에 남아 있습니다. | |
| Amazon DynamoDB 퍼블릭 IP 주소 사용 | Amazon VPC의 프라이빗 IP 주소를 사용하여 Amazon DynamoDB에 액세스 |
| 온프레미스에서의 액세스를 허용하지 않음 | 온프레미스에서의 액세스 허용 |
| 다른 AWS 리전에서의 액세스를 허용하지 않음 | Amazon VPC 피어링 또는 AWS Transit Gateway를 사용하여 다른 AWS 리전의 Amazon VPC 엔드포인트에서 액세스 허용 |
| 미청구 | 청구 |

게이트웨이 엔드포인트에 대한 자세한 내용은 AWS PrivateLink 안내서에서 Gateway Amazon VPC endpoints를 참조하세요.

## AWS PrivateLink for Amazon DynamoDB 사용 시 고려 사항

Amazon VPC 고려 사항이 AWS PrivateLink for Amazon DynamoDB에 적용됩니다. 자세한 내용은 AWS PrivateLink 가이드의 인터페이스 엔드포인트 고려 사항 및 AWS PrivateLink 할당량을 참조하십시오. 또한 다음과 같은 제한 사항이 적용됩니다.

AWS PrivateLink for Amazon DynamoDB는 다음을 지원하지 않습니다.

- 전송 계층 보안(TLS) 1.1
- 프라이빗 및 하이브리드 도메인 이름 시스템(DNS) 서비스

활성화한 각 AWS PrivateLink 엔드포인트에 대해 초당 최대 5만 개의 요청을 제출할 수 있습니다.

> **Note**
>
> AWS PrivateLink 엔드포인트에 대한 네트워크 연결 제한 시간은 DynamoDB 오류 응답 범위에 포함되지 않으므로 PrivateLink 엔드포인트에 연결하는 애플리케이션에서 적절하게 처리해야 합니다.

## Amazon VPC 엔드포인트 생성

Amazon VPC 인터페이스 엔드포인트를 생성하려면 AWS PrivateLink 안내서의 Create an Amazon VPC endpoint를 참조하세요.

## Amazon DynamoDB 인터페이스 엔드포인트에 액세스

인터페이스 엔드포인트를 생성하면 DynamoDB는 Regional 및 Zonal이라는 두 가지 유형의 엔드포인트별 DynamoDB DNS 이름을 생성합니다.

- 리전 DNS 이름에는 고유한 Amazon VPC 엔드포인트 ID, 서비스 식별자, AWS 리전, `vpce.amazonaws.com`이 포함됩니다. 예를 들어, Amazon VPC 엔드포인트 ID *`vpce-1a2b3c4d`*의 경우, 생성된 DNS 이름은 *`vpce-1a2b3c4d-5e6f`*`.dynamodb.us-east-1.vpce.amazonaws.com`과 비슷할 수 있습니다.
- 영역별 DNS 이름에는 가용 영역이 포함됩니다(예: *`vpce-1a2b3c4d-5e6f`*`-us-east-1a.dynamodb.us-east-1.vpce.amazonaws.com`). 아키텍처가 가용 영역을 분리하는 경우 이 옵션을 사용할 수 있습니다. 예를 들어, 오류를 제한하거나 리전별 데이터 전송 비용을 줄이는 데 사용할 수 있습니다.

> **Note**
>
> 최적의 신뢰성을 얻으려면 최소 3개의 가용 영역에 서비스를 배포하는 것이 좋습니다.

## DynamoDB 인터페이스 엔드포인트에서 DynamoDB 테이블에 및 제어 API 작업에 액세스

AWS CLI 또는 AWS SDK를 사용하여 DynamoDB 테이블에 액세스하고 DynamoDB 인터페이스 엔드포인트를 통해 API 작업을 제어할 수 있습니다.

## AWS CLI 예제

AWS CLI 명령의 DynamoDB 인터페이스 엔드포인트를 통해 DynamoDB 테이블 또는 DynamoDB 제어 API 작업에 액세스하려면 `--region` 및 `--endpoint-url` 파라미터를 사용하세요.

예: VPC 엔드포인트 생성

```
aws ec2 create-vpc-endpoint \
--region us-east-1 \
--service-name com.amazonaws.us-east-1.dynamodb \
--vpc-id client-vpc-id \
--subnet-ids client-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id
```

예: VPC 엔드포인트 수정

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
# any additional parameters needed, see Privatelink documentation for more details
```

예: 엔드포인트 URL을 사용하여 테이블 나열

다음 예시에서 리전 `us-east-1` 및 VPC 엔드포인트 ID의 DNS 이름 `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com`을 실제 정보로 바꿉니다.

```
aws dynamodb --region us-east-1 --endpoint https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com list-tables
```

## AWS SDK 예제

AWS SDK를 사용하여 DynamoDB 인터페이스 엔드포인트를 통해 DynamoDB 테이블 또는 DynamoDB 제어 API 작업에 액세스하려면 SDK를 최신 버전으로 업데이트하세요. 그런 다음, DynamoDB 인터페이스 엔드포인트를 통해 테이블 또는 DynamoDB 제어 API 작업에 액세스하기 위해 엔드포인트 URL을 사용하도록 클라이언트를 구성합니다.

SDK for Python (Boto3)

예: 엔드포인트 URL을 사용하여 DynamoDB 테이블에 액세스

다음 예에서 리전 us-east-1 및 VPC 엔드포인트 ID https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com을 실제 정보로 바꿉니다.

```
ddb_client = session.client(
service_name='dynamodb',
region_name='us-east-1',
endpoint_url='https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com'
)
```

SDK for Java 1.x

예: 엔드포인트 URL을 사용하여 DynamoDB 테이블에 액세스

다음 예에서 리전 us-east-1 및 VPC 엔드포인트 ID https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com을 실제 정보로 바꿉니다.

```
//client build with endpoint config
final AmazonDynamoDB dynamodb =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration(
                "https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com",
                Regions.DEFAULT_REGION.getName()
        )
).build();
```

SDK for Java 2.x

예: 엔드포인트 URL을 사용하여 DynamoDB 테이블에 액세스

다음 예에서 리전 us-east-1 및 VPC 엔드포인트 ID https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com을 실제 정보로 바꿉니다.

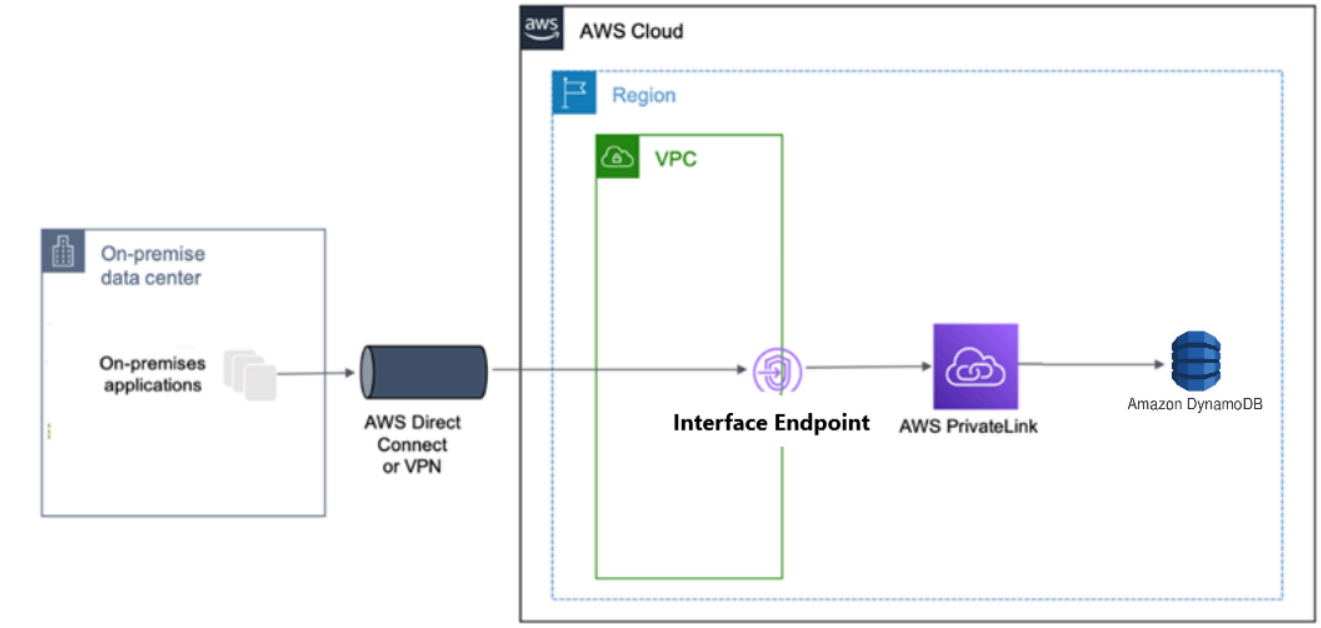
```
Region region = Region.US_EAST_1;
dynamoDbClient = DynamoDbClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

# 온프레미스 DNS 구성 업데이트

엔드포인트별 DNS 이름을 사용하여 DynamoDB의 인터페이스 엔드포인트에 액세스할 때는 온프레미스 DNS 리졸버를 업데이트할 필요가 없습니다. 퍼블릭 DynamoDB DNS 도메인의 인터페이스 엔드포인트의 프라이빗 IP 주소로 엔드포인트별 DNS 이름을 확인할 수 있습니다.

## 인터페이스 엔드포인트를 사용하여 Amazon VPC의 게이트웨이 엔드포인트 또는 인터넷 게이트웨이 없이 DynamoDB에 액세스

다음 다이어그램과 같이 Amazon VPC의 인터페이스 엔드포인트는 Amazon 네트워크를 통해 Amazon VPC 내 애플리케이션과 온프레미스 애플리케이션을 모두 DynamoDB로 라우팅할 수 있습니다.



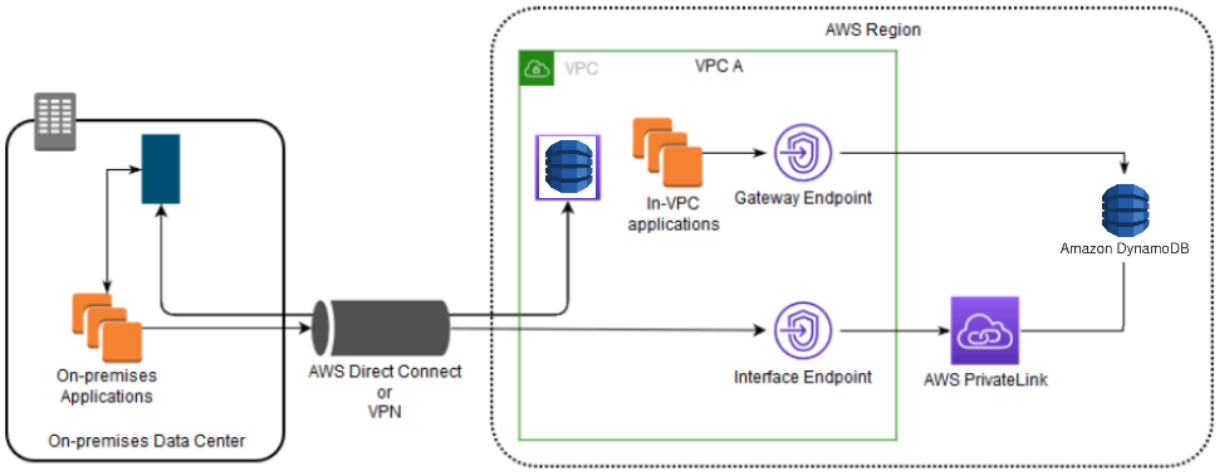

다이어그램은 다음을 보여 줍니다.

- 온프레미스 네트워크에서는 AWS Direct Connect 또는 AWS VPN을 사용하여 Amazon VPC A에 연결합니다.
- 온프레미스 및 Amazon VPC A의 애플리케이션은 엔드포인트별 DNS 이름을 사용하여 DynamoDB 인터페이스 엔드포인트를 통해 DynamoDB에 액세스합니다.
- 온프레미스 애플리케이션은 AWS Direct Connect(또는 AWS VPN)를 통해 Amazon VPC에서 인터페이스 엔드포인트로 데이터를 전송합니다. AWS PrivateLink는 AWS 네트워크를 통해 인터페이스 엔드포인트에서 DynamoDB로 데이터를 이동합니다.

- Amazon VPC 내 애플리케이션도 인터페이스 엔드포인트로 데이터를 전송합니다. AWS PrivateLink는 AWS 네트워크를 통해 인터페이스 엔드포인트에서 DynamoDB로 데이터를 이동합니다.

## 동일한 Amazon VPC에서 게이트웨이 엔드포인트와 인터페이스 엔드포인트를 함께 사용하여 DynamoDB에 액세스

다음 다이어그램과 같이 인터페이스 엔드포인트를 생성하고 기존 게이트웨이 엔드포인트를 동일한 Amazon VPC에 유지할 수 있습니다. 이 접근법을 사용하면 Amazon VPC 내 애플리케이션이 게이트웨이 엔드포인트를 통해 DynamoDB에 계속 액세스할 수 있으며, 요금은 청구되지 않습니다. 그런 다음, 온프레미스 애플리케이션만 인터페이스 엔드포인트를 사용하여 DynamoDB에 액세스합니다. 이러한 방식으로 DynamoDB에 액세스하려면 DynamoDB에 대한 엔드포인트별 DNS 이름을 사용하도록 온프레미스 애플리케이션을 업데이트해야 합니다.




다이어그램은 다음을 보여 줍니다.

- 온프레미스 애플리케이션은 엔드포인트별 DNS 이름을 사용하여 AWS Direct Connect(또는 AWS VPN)를 통해 Amazon VPC에서 인터페이스 엔드포인트로 데이터를 전송합니다. AWS PrivateLink는 AWS 네트워크를 통해 인터페이스 엔드포인트에서 DynamoDB로 데이터를 이동합니다.
- Amazon VPC 내 애플리케이션은 기본 리전 DynamoDB 이름을 사용하여 AWS 네트워크를 통해 DynamoDB에 연결하는 게이트웨이 엔드포인트로 데이터를 전송합니다.

게이트웨이 엔드포인트에 대한 자세한 내용은 Amazon VPC 사용 설명서의 Gateway Amazon VPC endpoints를 참조하세요.

# DynamoDB에 대한 Amazon VPC 엔드포인트 정책 생성

DynamoDB에 대한 액세스를 제어하는 Amazon VPC 엔드포인트에 엔드포인트 정책을 연결할 수 있습니다. 이 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 AWS Identity and Access Management(IAM) 보안 주체.
- 수행할 수 있는 작업
- 작업을 수행할 수 있는 리소스

주제


- 예: Amazon VPC 엔드포인트에서 특정 테이블로만 액세스 제한


## 예: Amazon VPC 엔드포인트에서 특정 테이블로만 액세스 제한

특정 DynamoDB 테이블로만 액세스를 제한하는 엔드포인트 정책을 생성할 수 있습니다. 이 정책 유형은 Amazon VPC에 테이블을 사용하는 다른 AWS 서비스가 있을 경우 유용합니다. 다음 테이블 정책은 *DOC-EXAMPLE-TABLE*로만 액세스를 제한합니다. 이 엔드포인트 정책을 사용하려면 *DOC-EXAMPLE-TABLE*을 실제 테이블의 이름으로 대체합니다.

JSON

```
{
"Version": "2012-10-17",
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-table-only",
      "Principal": "*",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE",
                   "arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE/*"]
    }
  ]
}
```

# AWS PrivateLink for DynamoDB Streams

AWS PrivateLink for Amazon DynamoDB Streams를 사용하면 Virtual Private Cloud(Amazon VPC)에서 인터페이스 Amazon VPC 엔드포인트(인터페이스 엔드포인트)를 프로비저닝할 수 있습니다. 이러한 엔드포인트는 VPN 및 AWS Direct Connect를 통해 온프레미스에 있는 애플리케이션에서 또는 Amazon VPC 피어링을 통해 다른 AWS 리전에 있는 애플리케이션에서 직접 액세스할 수 있습니다. AWS PrivateLink 및 인터페이스 엔드포인트를 사용하면 애플리케이션에서 DynamoDB Streams로의 프라이빗 네트워크 연결을 단순화할 수 있습니다.

Amazon VPC의 애플리케이션은 DynamoDB Streams 작업을 위해 Amazon VPC 인터페이스 엔드포인트를 사용하여 DynamoDB Streams와 통신하는 데 퍼블릭 IP 주소가 필요하지 않습니다. 인터페이스 엔드포인트는 Amazon VPC의 서브넷에서 프라이빗 IP 주소가 할당된 하나 이상의 탄력적 네트워크 인터페이스(ENI)로 표시됩니다. 인터페이스 엔드포인트를 통한 DynamoDB Streams에 대한 요청은 Amazon 네트워크에 유지됩니다. 또한, AWS Direct Connect 또는 AWS Virtual Private Network(AWS VPN)을 통해 온프레미스 애플리케이션에서 Amazon VPC의 인터페이스 엔드포인트에 액세스할 수 있습니다. AWS Virtual Private Network를 온프레미스 네트워크에 연결하는 방법에 대한 자세한 내용은 AWS Direct Connect 사용 설명서 및 AWS Site-to-Site VPN 사용 설명서를 참조하세요.

인터페이스 엔드포인트에 대한 일반적인 정보는 Interface Amazon VPC endpoints (AWS PrivateLink)를 참조하세요.

> **Note**
>
> DynamoDB Streams에는 인터페이스 엔드포인트만 지원됩니다. 게이트웨이 엔드포인트는 지원되지 않습니다.

주제

- AWS PrivateLink for Amazon DynamoDB Streams 사용 시 고려 사항
- Amazon VPC 엔드포인트 생성
- Amazon DynamoDB Streams 인터페이스 엔드포인트에 액세스
- DynamoDB Streams 인터페이스 엔드포인트에서 DynamoDB Streams API 작업에 액세스
- AWS SDK 예제
- DynamoDB Streams에 대한 Amazon VPC 엔드포인트 정책 만들기

## AWS PrivateLink for Amazon DynamoDB Streams 사용 시 고려 사항

Amazon VPC 고려 사항이 AWS PrivateLink for Amazon DynamoDB Streams에 적용됩니다. 자세한 내용은 interface endpoint considerations 및 AWS PrivateLink quotas를 참조하세요. 다음과 같은 제한 사항이 적용됩니다.

AWS PrivateLink for Amazon DynamoDB Streams는 다음을 지원하지 않습니다.

- 전송 계층 보안(TLS) 1.1
- 프라이빗 및 하이브리드 도메인 이름 시스템(DNS) 서비스

> **Note**
>
> AWS PrivateLink 엔드포인트에 대한 네트워크 연결 제한 시간은 DynamoDB Streams 오류 응답 범위에 포함되지 않으므로 AWS PrivateLink 엔드포인트에 연결하는 애플리케이션에서 적절하게 처리해야 합니다.

## Amazon VPC 엔드포인트 생성

Amazon VPC 인터페이스 엔드포인트를 생성하려면 AWS PrivateLink 안내서의 Create an Amazon VPC endpoint를 참조하세요.

## Amazon DynamoDB Streams 인터페이스 엔드포인트에 액세스

인터페이스 엔드포인트를 생성하면 DynamoDB는 Regional 및 Zonal이라는 두 가지 유형의 엔드포인트별 DynamoDB Streams DNS 이름을 생성합니다.

- 리전 DNS 이름에는 고유한 Amazon VPC 엔드포인트 ID, 서비스 식별자, AWS 리전, `vpce.amazonaws.com`이 포함됩니다. 예를 들어, Amazon VPC 엔드포인트 ID *`vpce-1a2b3c4d`*의 경우, 생성된 DNS 이름은 *`vpce-1a2b3c4d-5e6f`*`.streams.dynamodb.us-east-1.vpce.amazonaws.com`과 비슷할 수 있습니다.
- 영역별 DNS 이름에는 가용 영역이 포함됩니다(예: *`vpce-1a2b3c4d-5e6f`***`-us-east-1a`**`.streams.dynamodb.us-east-1.vpce.amazonaws.com`). 아키텍처가 가용 영역을 분리하는 경우 이 옵션을 사용할 수 있습니다. 예를 들어, 오류를 제한하거나 리전별 데이터 전송 비용을 줄이는 데 사용할 수 있습니다.

# DynamoDB Streams 인터페이스 엔드포인트에서 DynamoDB Streams API 작업에 액세스

AWS CLI 또는 AWS SDK를 사용하여 DynamoDB Streams 인터페이스 엔드포인트를 통해 DynamoDB Streams API 작업에 액세스할 수 있습니다.

AWS CLI 예제

AWS CLI 명령의 DynamoDB Streams 인터페이스 엔드포인트를 통해 DynamoDB Streams 또는 API 작업에 액세스하려면 `--region` 및 `--endpoint-url` 파라미터를 사용하세요.

예: VPC 엔드포인트 생성

```
aws ec2 create-vpc-endpoint \
--region us-east-1 \
--service-name com.amazonaws.us-east-1.dynamodb-streams \
--vpc-id client-vpc-id \
--subnet-ids client-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id
```

예: VPC 엔드포인트 수정

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
# any additional parameters needed, see Privatelink documentation for more details
```

예: 엔드포인트 URL을 사용하여 스트림 나열

다음 예시에서 리전 `us-east-1` 및 VPC 엔드포인트 ID의 DNS 이름 `vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com`을 실제 정보로 바꿉니다.

```
aws dynamodbstreams --region us-east-1 –endpoint https://
vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com list-streams
```

## AWS SDK 예제

AWS SDK를 사용하여 DynamoDB Streams 인터페이스 엔드포인트를 통해 Amazon DynamoDB Streams API 작업에 액세스하려면 SDK를 최신 버전으로 업데이트하세요. 그런 다음, DynamoDB Streams 인터페이스 엔드포인트를 통해 DynamoDB Streams API 작업에 대해 엔드포인트 URL을 사용하도록 클라이언트를 구성합니다.

SDK for Python (Boto3)

예: 엔드포인트 URL을 사용하여 DynamoDB 스트림에 액세스

다음 예에서 리전 `us-east-1` 및 VPC 엔드포인트 ID `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com`을 실제 정보로 바꿉니다.

```
ddb_streams_client = session.client(
service_name='dynamodbstreams',
region_name='us-east-1',
endpoint_url='https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com'
)
```

SDK for Java 1.x

예: 엔드포인트 URL을 사용하여 DynamoDB 스트림에 액세스

다음 예에서 리전 `us-east-1` 및 VPC 엔드포인트 ID `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com`을 실제 정보로 바꿉니다.

```
//client build with endpoint config
final AmazonDynamoDBStreams dynamodbstreams =
 AmazonDynamoDBStreamsClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration(
                "https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com",
                Regions.DEFAULT_REGION.getName()
        )
).build();
```

SDK for Java 2.x

예: 엔드포인트 URL을 사용하여 DynamoDB 스트림에 액세스

다음 예에서 리전 `us-east-1` 및 VPC 엔드포인트 ID `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com`을 실제 정보로 바꿉니다.

```
Region region = Region.US_EAST_1;
dynamoDbStreamsClient = DynamoDbStreamsClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

## DynamoDB Streams에 대한 Amazon VPC 엔드포인트 정책 만들기

DynamoDB Streams에 대한 액세스를 제어하는 Amazon VPC 엔드포인트에 엔드포인트 정책을 연결할 수 있습니다. 이 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 AWS Identity and Access Management(IAM) 보안 주체.
- 수행할 수 있는 작업
- 작업을 수행할 수 있는 리소스

주제

- 예: Amazon VPC 엔드포인트에서 특정 스트림으로만 액세스 제한

예: Amazon VPC 엔드포인트에서 특정 스트림으로만 액세스 제한

특정 DynamoDB Streams로만 액세스를 제한하는 엔드포인트 정책을 만들 수 있습니다. 이 정책 유형은 Amazon VPC에 DynamoDB Streams를 사용하는 다른 AWS 서비스가 있을 경우 유용합니다. 다음 스트림 정책은 *DOC-EXAMPLE-TABLE*에 연결된 스트림 *2025-02-20T11:22:33.444*로만 액세스를 제한합니다. 이 엔드포인트 정책을 사용하려면 *DOC-EXAMPLE-TABLE*을 테이블의 이름으로, *2025-02-20T11:22:33.444*를 스트림 레이블로 대체합니다.

JSON

```
{
```

```
"Version": "2012-10-17",
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-stream-only",
      "Principal": "*",
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/table-name/
stream/2025-02-20T11:22:33.444"]
    }
  ]
}
```

> **Note**
>
> 게이트웨이 엔드포인트는 DynamoDB Streams에서 지원되지 않습니다.

# Amazon DynamoDB의 구성 및 취약성 분석

AWS가 게스트 운영 체제(OS), 데이터베이스 패치, 방화벽 구성, 재해 복구 등의 기본 보안 작업을 처리합니다. 적합한 제3자가 이 절차를 검토하고 인증하였습니다. 자세한 내용은 다음 리소스를 참조하세요.

- Amazon DynamoDB의 규정 준수 확인
- 공동 책임 모델
- Amazon Web Services: 보안 프로세스의 개요(백서)

다음 보안 모범 사례에서도 Amazon DynamoDB의 구성 및 취약성 분석을 다룹니다.

- AWS Config 규칙을 이용한 DynamoDB 규정 준수 모니터링
- AWS Config를 이용한 DynamoDB 구성 모니터링

# Amazon DynamoDB 보안 모범 사례

Amazon DynamoDB는 자체 보안 정책을 개발하고 구현할 때 고려해야 할 여러 보안 기능을 제공합니다. 다음 모범 사례는 일반적인 지침이며 완벽한 보안 솔루션을 나타내지는 않습니다. 이러한 모범 사례는 환경에 적절하지 않거나 충분하지 않을 수 있으므로 참고용으로만 사용해 주세요.

주제


- DynamoDB 예방 보안 모범 사례
- DynamoDB 탐지 보안 모범 사례


## DynamoDB 예방 보안 모범 사례

다음과 같은 모범 사례를 통해 Amazon DynamoDB에서 보안 사고를 예측하고 방지할 수 있습니다.

유휴 시 암호화

DynamoDB는 유휴 시 테이블, 인덱스, 스트림 및 백업에 저장된 모든 사용자 데이터를 AWS Key Management Service(AWS KMS)에 저장된 암호화 키를 사용해 암호화합니다. 이를 통해 기본 스토리지에 대한 무단 액세스로부터 데이터를 보호하여 데이터 보호 계층을 추가로 제공합니다.

DynamoDB가 AWS 소유 키(기본 암호화 유형)와 AWS 관리형 키 또는 고객 관리형 키 중에서 어떤 것을 사용해 사용자 데이터를 암호화하도록 할지 지정할 수 있습니다. 자세한 내용은 미사용 Amazon DynamoDB 암호화를 참조하세요.

IAM 역할을 사용해 DynamoDB에 대한 액세스 인증하기

사용자, 애플리케이션, 기타 AWS 서비스는 DynamoDB에 액세스하려면 AWS API 요청에 유효한 AWS 자격 증명이 있어야 합니다. AWS 자격 증명을 애플리케이션이나 EC2 인스턴스에 직접 저장해서는 안 됩니다. 이러한 보안 인증은 자동으로 교체되지 않기 때문에 손상된 경우 비즈니스에 큰 영향을 줄 수 있는 장기 보안 인증입니다. IAM 역할을 사용하면 AWS 서비스 및 리소스에 액세스하는 데 사용할 수 있는 임시 액세스 키를 얻을 수 있습니다.

자세한 내용은 Amazon DynamoDB의 Identity and Access Management 섹션을 참조하세요.

IAM 정책을 사용한 DynamoDB 기본 인증

권한을 부여하려면 권한을 부여받을 사용자, 권한을 행사할 수 있는 대상이 되는 DynamoDB API, 해당 리소스에 허용하고자 하는 특정 작업을 결정합니다. 최소 권한을 구현하는 것이 오류 또는 악의적인 의도로 인해 발생할 수 있는 보안 위험과 영향을 줄일 수 있는 비결입니다.

IAM 자격 증명(즉 사용자, 그룹 및 역할)에 권한 정책을 연결함으로써 DynamoDB 리소스에서 작업을 수행할 수 있는 권한을 부여합니다.

이를 위해 다음 정책을 사용할 수 있습니다.

- AWS 관리형(미리 정의된) 정책
- 고객 관리형 정책

IAM 정책 조건을 사용하여 세분화된 액세스 제어 구현

DynamoDB에서 권한을 부여할 때 권한 정책이 적용되는 방식을 결정하는 조건을 지정할 수 있습니다. 최소 권한을 구현하는 것이 오류 또는 악의적인 의도로 인해 발생할 수 있는 보안 위험과 영향을 줄일 수 있는 비결입니다.

IAM 정책을 사용해 권한을 부여할 때 조건을 지정할 수 있습니다. 예를 들어 다음을 수행할 수 있습니다.

- 사용자에게 테이블 또는 보조 인덱스의 특정 항목 및 속성에 대한 읽기 전용 액세스를 허용하는 권한을 부여할 수 있습니다.
- 사용자 ID를 기준으로 테이블의 특정 속성에 대한 쓰기 전용 액세스 권한을 해당 사용자에게 부여할 수 있습니다.

자세한 내용은 세분화된 액세스 제어를 위한 IAM 정책을 참조하세요.

VPC 엔드포인트 및 정책을 사용해 DynamoDB에 액세스

Virtual Private Cloud(VPC)에서 DynamoDB에 액세스하는 권한만 필요한 경우에는 VPC 엔드포인트를 사용해 필요한 VPC에서만 액세스할 수 있도록 제한해야 합니다. 이를 통해 해당 트래픽이 개방형 인터넷을 통과하여 이 환경에 종속되는 것을 방지할 수 있습니다.

DynamoDB에 VPC 엔드포인트를 사용하면 다음 정책을 사용해 액세스를 제어 및 제한할 수 있습니다.

- VPC 엔드포인트 정책 - 이 정책은 DynamoDB VPC 종단점에서 적용됩니다. 이 정책을 통해 DynamoDB 테이블에 대한 API 액세스를 제어 및 제한할 수 있습니다.
- IAM 정책 - 사용자, 그룹 또는 역할에 연결된 정책의 `aws:sourceVpce` 조건을 사용해 DynamoDB 테이블에 대한 모든 액세스가 지정된 VPC 엔드포인트를 통해 이루어지도록 강제할 수 있습니다.

자세한 내용은 Amazon DynamoDB용 엔드포인트를 참조하세요.

클라이언트 측 암호화 참조

DynamoDB에서 테이블을 구현하기 전에 암호화 전략을 계획하는 것이 좋습니다. 민감한 데이터나 기밀 데이터를 DynamoDB에 저장하는 경우 계획에 클라이언트측 암호화를 포함하는 것을 고려해 보세요. 이렇게 하면 데이터를 원본에 최대한 가깝게 암호화하고 전체 수명 주기 동안 데이터를 보호할 수 있습니다. 전송 중 및 유휴 상태의 중요 데이터를 암호화하면 일반 텍스트 데이터를 제3자가 사용할 수 없게 하는 데 도움이 됩니다.

AWS Database Encryption SDK for DynamoDB는 DynamoDB에 보낼 테이블 데이터를 보호하는 데 도움이 되는 소프트웨어 라이브러리입니다. DynamoDB 테이블 항목을 암호화, 서명, 확인 및 복호화합니다. 암호화 및 서명되는 속성을 제어할 수 있습니다.

프라이머리 키 관련 고려 사항

테이블과 글로벌 보조 인덱스의 프라이머리 키에 민감한 이름이나 민감한 일반 텍스트 데이터를 사용하지 마세요. 키 이름은 테이블 정의에 표시됩니다. 예를 들어 DescribeTable을 직접적으로 호출할 권한이 있는 사람은 누구나 프라이머리 키 이름에 액세스할 수 있습니다. 키 값은 AWS CloudTrail 및 기타 로그에 표시될 수 있습니다. 또한 DynamoDB는 키 값을 사용하여 데이터를 배포하고 요청을 라우팅하며 AWS 관리자는 이 값을 관찰하여 서비스 상태를 유지할 수 있습니다.

테이블 또는 GSI 키 값의 민감한 데이터를 사용해야 하는 경우 종단 간 클라이언트 암호화를 사용하는 것이 좋습니다. 이렇게 하면 데이터에 대한 키-값 참조를 수행할 수 있으며 데이터가 DynamoDB 관련 로그에 암호화되지 않은 상태로 표시되지 않습니다. 이 작업을 수행하는 한 가지 방법은 AWS Database Encryption SDK for DynamoDB를 사용하는 것이지만 이것이 필수는 아닙니다. 자체 솔루션을 사용하는 경우 항상 충분히 안전한 암호화 알고리즘을 사용해야 합니다. 해시와 같은 비암호화 옵션은 대부분의 상황에서 충분히 안전하지 않은 것으로 간주되므로 사용해서는 안 됩니다.

프라이머리 키의 키 이름이 민감한 경우 `pk` 및 `sk`를 대신 사용하는 것이 좋습니다. 이는 파티션 키 설계를 유연하게 만드는 일반적인 모범 사례입니다.

무엇이 올바른 선택인지 고민된다면 언제든지 보안 전문가나 AWS 계정 팀에 문의하세요.

## DynamoDB 탐지 보안 모범 사례

다음과 같은 Amazon DynamoDB 모범 사례는 잠재적 보안 약점과 사고를 탐지하는 데 도움이 됩니다.

AWS CloudTrail을 사용해 AWS 관리형 KMS 키 사용량 모니터링하기

유휴 중 암호화를 위해 AWS 관리형 키를 사용하고 있는 경우 이 키의 사용량은 AWS CloudTrail로 로깅됩니다. CloudTrail은 계정에서 수행한 작업을 기록함으로써 사용자 활동에 대한 가시성을 제

공합니다. CloudTrail은 요청한 사용자, 사용한 서비스, 수행한 작업, 작업에 대한 파라미터, AWS 서비스가 반환하는 응답 요소 등 각 작업에 대한 중요 정보를 기록합니다. 이러한 정보는 AWS 리소스의 변경 사항을 추적하고 운영 관련 문제를 해결하는 데 도움이 됩니다. CloudTrail을 이용하면 내부 정책 및 규제 표준 준수를 더 쉽게 보장할 수 있습니다.

CloudTrail을 사용해 키 사용을 감사할 수 있습니다. CloudTrail은 계정의 AWS API 호출 및 관련 이벤트 내역이 포함된 로그 파일을 생성합니다. 이 로그 파일은 AWS Management Console, AWS SDK, 명령줄 도구를 사용해 이루어진 모든 AWS KMS API 요청 외에도 통합된 AWS 서비스를 통해 이루어진 요청도 포함합니다. 이러한 로그 파일을 사용해 키가 사용된 시간, 요청되었던 작업, 요청자의 ID, 요청이 시작된 IP 주소 등에 대한 정보를 얻을 수 있습니다. 자세한 내용은 AWS CloudTrail을 사용하여 AWS KMS API 호출 및 AWS CloudTrail 사용 설명서를 참조하세요.

CloudTrail 사용하여 DynamoDB 작업 모니터링

CloudTrail은 제어 플레인 이벤트와 데이터 영역 이벤트를 모두 모니터링할 수 있습니다. 제어 플레인 작업은 DynamoDB 테이블을 생성하고 관리하도록 해 줍니다. 또한 인덱스, 스트림 및 테이블에 따라 다른 다양한 객체를 사용하도록 해 줍니다. 데이터 영역 작업은 테이블의 데이터에 대해 생성, 읽기, 업데이트 및 삭제(CRUD) 작업을 수행하도록 해 줍니다. 데이터 영역 작업의 일부는 보조 인덱스로부터 데이터를 읽도록 해주기도 합니다. CloudTrail에서 데이터 영역 이벤트의 로깅을 활성화하려면 CloudTrail에서 데이터 영역 API 활동의 로깅을 활성화해야 합니다. 자세한 내용은 추적을 위한 데이터 이벤트 로깅 단원을 참조하세요.

DynamoDB에서 활동이 수행되면 해당 활동은 이벤트 기록에 있는 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. 자세한 내용은 AWS CloudTrail을 사용하여 DynamoDB 작업 로깅을 참조하세요. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서에서 CloudTrail 이벤트 기록을 사용하여 이벤트 보기를 참조하세요.

DynamoDB에 대한 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon Simple Storage Service(Amazon S3) 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 리전의 이벤트를 로깅하고 지정된 S3 버킷으로 로그 파일을 전송합니다. 이에 더해, CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다.

DynamoDB Streams를 이용한 데이터 영역 작업 모니터링

DynamoDB는 DynamoDB Streams의 이벤트에 자동으로 응답하는 코드 조각인 트리거를 만들 수 있도록 AWS Lambda와 통합되어 있습니다. 트리거를 사용하면 DynamoDB 테이블의 데이터 수정에 응답하는 애플리케이션을 구축할 수 있습니다.

테이블에 DynamoDB Streams를 활성화할 경우 스트림 Amazon 리소스 이름(ARN)을 사용자가 작성하는 Lambda 함수에 연결할 수 있습니다. 테이블의 항목이 수정되는 즉시 새로운 레코드가 테이블의 스트림에 표시됩니다. AWS Lambda는 새로운 스트림 레코드가 감지될 때마다 스트림을 폴링하고 Lambda 함수를 동기식으로 호출합니다. Lambda 함수는 알림 보내기, 워크플로 시작과 같이 사용자가 지정하는 모든 작업을 수행할 수 있습니다.

자세한 내용은 자습서: Amazon DynamoDB Streams와 함께 AWS Lambda 사용하기를 참조하세요. 이 예에서는 DynamoDB 이벤트 입력을 받아들여 이 입력에 포함된 메시지를 처리한 후 수신 이벤트 데이터 중 일부를 Amazon CloudWatch Logs에 씁니다.

AWS Config를 이용한 DynamoDB 구성 모니터링

AWS Config를 통해 AWS 리소스의 구성 변경 사항을 지속적으로 모니터링하고 기록할 수 있습니다. 또한 AWS Config를 이용해 AWS 리소스의 목록을 만들 수도 있습니다. 이전 상태에서 변경된 사항이 탐지되면 사용자가 검토하여 조치를 취할 수 있도록 Amazon Simple Notification Service(Amazon SNS) 알림이 전송됩니다. 콘솔을 이용한 AWS Config 설정의 지침에 따라 DynamoDB 리소스 유형이 포함되도록 합니다.

Amazon SNS 주제로 구성 변경 및 알림을 스트리밍하도록 AWS Config를 구성할 수 있습니다. 예를 들어, 리소스가 업데이트되면 변경 사항을 볼 수 있도록 이메일로 전송되는 알림을 받을 수 있습니다. AWS Config가 리소스와 대조하여 사용자 지정 규칙 또는 관리형 규칙을 평가하는 경우에도 알림을 받을 수 있습니다.

예제는 AWS Config개발자 안내서에서 AWS Config가 Amazon SNS 주제로 전송하는 알림을 참조하세요.

AWS Config 규칙을 이용한 DynamoDB 규정 준수 모니터링

AWS Config는 리소스에서 발생하는 구성 변경 사항을 지속적으로 추적합니다. 또한 이러한 변경으로 인해 규칙의 조건 중 위반한 것이 있는지 여부를 확인합니다. 리소스가 규칙을 위반한 경우, AWS Config는 해당 리소스와 규칙을 규칙 미준수로 표시합니다.

AWS Config를 사용해 리소스 구성을 평가함으로써 리소스 구성이 내부 관행, 업계 지침, 규정을 얼마나 잘 준수하는지 평가할 수 있습니다. AWS Config는 AWS 관리형 규칙을 제공합니다. 이 규칙은 AWS 리소스가 일반 모범 사례를 준수하는지 여부를 평가하기 위해 AWS Config가 사용하는 미리 정의된 사용자 지정 가능 규칙입니다.

식별 및 자동화를 위해 DynamoDB 리소스에 태그 지정하기

AWS 리소스에 태그 형태로 메타데이터를 지정할 수 있습니다. 각 태그는 리소스 관리, 검색 및 필터링을 더 수월하게 해줄 수 있는 옵션 값과 고객이 정의한 키로 구성된 간단한 레이블입니다.

태그를 지정하면 그룹화 제어를 구현할 수 있습니다. 고유한 태그 유형은 없지만 목적, 소유자, 환경 또는 기타 기준에 따라 리소스를 분류할 수 있습니다. 다음은 몇 가지 예입니다.

- 보안 - 암호화와 같은 요구 사항을 결정하는 데 사용됩니다.
- 기밀성 – 리소스가 지원하는 특정 데이터 기밀성 수준에 대한 식별자입니다.
- 환경 - 개발, 테스트 및 프로덕션 인프라 간 구별에 사용됩니다.

자세한 내용은 AWS 태그 지정 전략 및 DynamoDB에 태그 지정하기를 참조하세요.

AWS Security Hub을 사용하여 보안 모범 사례와 관련된 Amazon DynamoDB의 사용량을 모니터링합니다.

Security Hub는 보안 제어를 사용하여 리소스 구성 및 보안 표준을 평가하여 다양한 규정 준수 프레임워크를 준수할 수 있도록 지원합니다.

Security Hub를 사용하여 DynamoDB 리소스를 평가하는 방법에 대한 자세한 내용은 AWS Security Hub 사용 설명서의 Amazon DynamoDB 제어를 참조하세요.

# DynamoDB의 모니터링 및 로깅

모니터링은 DynamoDB 및 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. 다중 지점 실패를 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집해야 합니다.

주제


- 모니터링 계획
- 성능 기준
- 통합 서비스
- 자동 모니터링 도구
- Amazon CloudWatch로 DynamoDB에서 지표 모니터링
- AWS CloudTrail을 사용하여 DynamoDB 작업 로깅
- DynamoDB에 대한 CloudWatch Contributor Insights를 사용해 데이터 액세스 분석


## 모니터링 계획

DynamoDB에 대한 모니터링을 시작하기 전에 다음 질문에 대한 답변을 포함하는 모니터링 계획을 작성합니다

- 모니터링의 목표
- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

## 성능 기준

다양한 시간과 다양한 부하 조건에서 성능을 측정하여 환경에서 일반 DynamoDB 성능의 기준선을 설정합니다. DynamoDB를 모니터링할 때 과거 모니터링 데이터를 저장할 것을 고려해야 합니다. 이 저장된 데이터는 현재 성능 데이터와 비교하고, 일반 성능 패턴과 성능 이상을 식별하고, 문제 해결 방법을 제안하는 기준이 됩니다. 기준선을 설정하려면 최소한 다음 항목을 모니터링해야 합니다.

- 일정 시간 사용된 읽기 또는 쓰기 용량 단위의 수로서, 이를 통해 할당된 처리량이 얼마나 많이 사용되는지 추적할 수 있습니다.
- 일정 시간 테이블의 프로비저닝된 읽기 또는 쓰기 용량을 초과한 요청으로서, 이를 통해 어느 요청이 테이블의 프로비저닝된 처리량 할당량을 초과하는지 확인할 수 있습니다.
- 시스템 오류로서, 이를 통해 오류가 발생한 요청이 있는지 확인할 수 있습니다.

## 통합 서비스

DynamoDB는 사용자 대신 테이블을 자동으로 모니터링하고 Amazon CloudWatch를 통해 지표를 보고합니다. 또한 DynamoDB는 DynamoDB 리소스를 모니터링하고 문제를 해결하는 데 도움이 되도록 다음 AWS 서비스와 통합됩니다.

- AWS CloudTrail은 직접 수행하거나 AWS 계정을 대신하여 수행한 API 직접 호출 및 관련 이벤트를 캡처하고 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. 자세한 내용은 AWS CloudTrail을 사용하여 DynamoDB 작업 로깅 섹션을 참조하세요.
- Contributor Insights는 테이블 또는 인덱스에서 가장 자주 액세스하고 제한된 키를 한눈에 확인할 수 있는 진단 도구입니다. 자세한 내용은 DynamoDB에 대한 CloudWatch Contributor Insights를 사용해 데이터 액세스 분석 섹션을 참조하세요.

## 자동 모니터링 도구

AWS는 DynamoDB를 모니터링하는 데 사용할 수 있는 다양한 도구를 제공합니다. 모니터링 작업은 최대한 자동화하는 것이 좋습니다. 다음과 같은 자동 모니터링 도구를 사용하여 DynamoDB를 관찰하고 문제 발생 시 보고할 수 있습니다.

- Amazon CloudWatch 경보 – 지정한 기간 동안 단일 지표를 감시하고, 여러 기간에 대해 지정된 임계값과 관련하여 지표 값을 기준으로 하나 이상의 태스크를 수행합니다.

  이 작업은 Amazon Simple Notification Service(Amazon SNS) 주제 또는 Amazon EC2 Auto Scaling 정책에 전송되는 알림입니다. Amazon CloudWatch 경보는 특정 상태에 있다는 이유만으로는 작업을 간접적으로 호출하지 않습니다. 상태가 변경되고 지정한 기간 동안 유지되어야 합니다. 자세한 내용은 Amazon CloudWatch로 DynamoDB에서 지표 모니터링 섹션을 참조하세요.
- AWS CloudTrail 로그 모니터링 - 계정 간에 로그 파일을 공유하고, AWS CloudTrail 로그 파일을 AWS CloudTrail Logs에 전송하여 실시간으로 모니터링하며, 로그 처리 애플리케이션을 Java로 작성하고, 로그 파일이 AWS CloudTrail 전송 후 변경되지 않았는지 확인합니다. 자세한 내용은 AWS CloudTrail 사용 설명서에서 Amazon CloudWatch Logs란 무엇인가요?를 참조하세요.

# Amazon CloudWatch로 DynamoDB에서 지표 모니터링

DynamoDB의 원시 데이터를 수집하여 읽기 가능하며 실시간에 가까운 지표로 처리하는 CloudWatch를 통해 DynamoDB를 모니터링할 수 있습니다. 이러한 통계는 일정 기간 동안 유지되므로 기록 정보를 보고 웹 애플리케이션이나 서비스가 어떻게 실행되고 있는지 전체적으로 더 잘 파악할 수 있습니다. 기본적으로 DynamoDB 지표 데이터는 CloudWatch에 자동으로 전송됩니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch란 무엇인가요? 및 지표 보존 기간을 참조하세요.

주제


- DynamoDB 지표 사용 방법
- CloudWatch 콘솔에서 지표 보기
- AWS CLI에서 지표 보기
- DynamoDB 지표 및 차원
- DynamoDB에 CloudWatch 경보 생성


## DynamoDB 지표 사용 방법

DynamoDB에서 보고하는 지표는 다양한 방법으로 분석이 기능한 정보를 제공합니다. 다음 목록은 몇 가지 일반적인 지표 사용 사례를 보여 줍니다. 모든 사용 사례를 망라한 것은 아니지만 시작하는 데 참고가 될 것입니다.

DynamoDB 지표 사용 방법

| 사용 방법 | 관련 지표 |
| --- | --- |
| 내 테이블의 TTL 삭제 속도를 모니터링하려면 어떻게 해야 하나요? | 지정한 시간 동안 `TimeToLiveDeletedItemCount` 를 모니터링하여 테이블에 대한 TTL 삭제 속도를 추적할 수 있습니다. `TimeToLiveDeletedItemCount` 지표를 사용하는 서버리스 애플리케이션의 예를 보려면 Automatically archive items to S3 using DynamoDB time to live (TTL) with AWS Lambda and Amazon Data Firehose를 참조하세요. |
| 프로비저닝된 처리량이 얼마나 사용되고 있는지 어떻게 알 수 있나요? | 일정 시간 `ConsumedReadCapacityUnits` 또는 `ConsumedWriteCapacityUnits` 를 모니터링하여 할당된 처리량이 얼마나 사용되는지 추적할 수 있습니다. |

| 사용 방법 | 관련 지표 |
| --- | --- |
| 테이블의 프로비저닝된 처리량 할당량을 초과하는 요청을 어떻게 확인할 수 있나요? | 요청에 포함된 이벤트가 프로비저닝된 처리량 할당량을 초과하면 `ThrottledRequests` 가 1씩 증분됩니다. 그런 다음 요청의 병목 현상 원인이 되는 이벤트는 `ThrottledRequests` 를 테이블과 테이블 인덱스의 `ReadThrottleEvents` 및 `WriteThrottleEvents` 지표와 비교해보면 알 수 있습니다. |
| 시스템 오류가 발생했는지 어떻게 확인할 수 있나요? | `SystemErrors` 를 모니터링하여 HTTP 500(서버 오류) 코드가 발생한 요청이 있는지 확인할 수 있습니다. 일반적으로 이 지표는 0이어야 합니다. 그렇지 않다면 조사가 필요합니다. |
| 테이블 작업의 지연 시간 값을 모니터링하려면 어떻게 해야 하나요? | 백분위수 지표(p50)를 통해 평균 지연 시간 및 중앙값 지연 시간을 추적하여 `SuccessfulRequestLatency` 를 모니터링할 수 있습니다. 가끔 지연 시간이 급증하는 것은 걱정할 필요가 없습니다. 그러나 평균 지연 시간 또한 p50(중앙값)이 높으면 해결해야 할 근본적인 문제가 있을 수 있습니다. 자세한 정보는 Amazon DynamoDB의 지연 시간 문제 해결을 참조하세요. |

## CloudWatch 콘솔에서 지표 보기

지표는 먼저 서비스 네임스페이스별로 그룹화된 다음, 각 네임스페이스 내에서 다양한 차원 조합별로 그룹화됩니다.

CloudWatch 콘솔에서 지표 보기

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 지표, 모든 지표를 선택합니다.
3. DynamoDB 네임스페이스를 선택합니다. 사용량(Usage) 네임스페이스를 선택하여 DynamoDB 사용량 지표를 확인할 수도 있습니다. 사용량 지표에 대한 자세한 내용은 AWS 사용량 지표를 참조하세요.
4. 찾아보기 탭에 네임스페이스의 모든 지표가 표시됩니다.
5. (선택 사항) 이 지표 그래프를 CloudWatch 대시보드에 추가하려면 작업, 대시보드에 추가를 선택합니다.

# AWS CLI에서 지표 보기

AWS CLI를 사용하여 지표 정보를 얻으려면 CloudWatch 명령 `list-metrics`를 사용합니다. 다음 예에서는 AWS/DynamoDB 네임스페이스의 모든 지표를 나열합니다.

```
aws cloudwatch list-metrics --namespace "AWS/DynamoDB"
```

지표 통계를 얻으려면 `get-metric-statistics` 명령을 사용합니다. 다음 명령은 5분 단위로 특정 24시간 동안의 `ProductCatalog` 테이블에 대한 `ConsumedReadCapacityUnits` 통계를 가져옵니다.

```
aws cloudwatch get-metric-statistics –namespace AWS/DynamoDB \
     –metric-name ConsumedReadCapacityUnits \
     –start-time 2023-11-01T00:00:00Z \
     –end-time 2023-11-02T00:00:00Z \
     –period 360 \
     –statistics Average \
     –dimensions Name=TableName,Value=ProductCatalog
```

샘플 출력은 다음과 같이 나타납니다.

```
{
    "Datapoints": [
        {
            "Timestamp": "2023-11-01T 09:18:00+00:00",
            "Average": 20,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 04:36:00+00:00",
            "Average": 22.5,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 15:12:00+00:00",
            "Average": 20,
            "Unit": "Count"
        }, ...
        {
```

```
            "Timestamp": "2023-11-01T 17:30:00+00:00",
            "Average": 25,
            "Unit": "Count"
        }
    ],
    "Label": " ConsumedReadCapacityUnits "
}
```

## DynamoDB 지표 및 차원

DynamoDB는 사용자와 상호 작용할 때 지표와 차원을 CloudWatch로 전송합니다.

DynamoDB 출력은 1분 동안 프로비저닝된 처리량을 소비했습니다. Auto Scaling은 소비된 용량이 구성된 목표 사용률을 2분 연속으로 초과할 때 트리거됩니다. CloudWatch 경보가 Auto Scaling을 트리거하기 전에 최대 몇 분의 짧은 지연이 있을 수 있습니다. 이러한 지연을 통해 정확한 CloudWatch 지표 평가가 이루어집니다. 하지만 소비된 처리량의 급증 간격이 1분보다 크면 Auto Scaling이 트리거되지 않을 수 있습니다. 마찬가지로 15개의 연속 데이터 포인트가 목표 사용률보다 낮을 때 스케일 다운 이벤트가 발생할 수 있습니다. 두 경우 모두 Auto Scaling이 트리거된 후 UpdateTable가 간접적으로 호출됩니다. 그런 다음 테이블 또는 인덱스의 프로비저닝된 용량을 업데이트하는 데 몇 분 정도 걸립니다. 이 기간 동안 테이블의 이전 프로비저닝된 용량을 초과하는 모든 요청은 제한됩니다.

### 지표 및 차원 보기

CloudWatch에 DynamoDB에 대한 다음 지표가 표시됩니다.

DynamoDB 지표

> **Note**
>
> Amazon CloudWatch에서는 1분 간격으로 다음과 같은 지표를 집계합니다.
>
> - `ConditionalCheckFailedRequests`
> - `ConsumedReadCapacityUnits`
> - `ConsumedWriteCapacityUnits`
> - `ReadThrottleEvents`
> - `ReturnedBytes`
> - `ReturnedItemCount`
> - `ReturnedRecordsCount`

- SuccessfulRequestLatency
- SystemErrors
- TimeToLiveDeletedItemCount
- ThrottledRequests
- TransactionConflict
- UserErrors
- WriteThrottleEvents

다른 모든 DynamoDB 지표의 경우 집계 단위는 5분입니다.

Average나 Sum처럼 모든 지표에 적용되지 않는 통계도 있습니다. 하지만 이 값은 모두 Amazon DynamoDB 콘솔, CloudWatch 콘솔, AWS CLI 또는 AWS SDK(모든 지표에 대해)를 통해 사용할 수 있습니다.

다음 목록에는 각 지표에 적용되는 유효한 통계 집합이 있습니다.

사용 가능한 지표 목록

- AccountMaxReads
- AccountMaxTableLevelReads
- AccountMaxTableLevelWrites
- AccountMaxWrites
- AccountProvisionedReadCapacityUtilization
- AccountProvisionedWriteCapacityUtilization
- AgeOfOldestUnreplicatedRecord
- ConditionalCheckFailedRequests
- ConsumedChangeDataCaptureUnits
- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits
- FailedToReplicateRecordCount
- MaxProvisionedTableReadCapacityUtilization
- MaxProvisionedTableWriteCapacityUtilization

- OnDemandMaxReadRequestUnits
- OnDemandMaxWriteRequestUnits
- OnlineIndexConsumedWriteCapacity
- OnlineIndexPercentageProgress
- OnlineIndexThrottleEvents
- PendingReplicationCount
- ProvisionedReadCapacityUnits
- ProvisionedWriteCapacityUnits
- ReadThrottleEvents
- ReplicationLatency
- ReturnedBytes
- ReturnedItemCount
- ReturnedRecordsCount
- SuccessfulRequestLatency
- SystemErrors
- TimeToLiveDeletedItemCount
- ThrottledPutRecordCount
- ThrottledRequests
- TransactionConflict
- UserErrors
- WriteThrottleEvents

AccountMaxReads

계정에서 사용할 수 있는 읽기 용량 유닛의 최대 수입니다. 이 제한은 온디맨드 테이블 또는 글로벌 보조 인덱스에는 적용되지 않습니다.

단위: `Count`

유효한 통계:

- `Maximum` - 계정에서 사용할 수 있는 읽기 용량 단위의 최대 수입니다.

AccountMaxTableLevelReads

계정의 테이블 또는 글로벌 보조 인덱스에서 사용할 수 있는 읽기 용량 유닛의 최대 수입니다. 온디맨드 테이블의 경우 이 제한이 테이블 또는 글로벌 보조 인덱스에서 사용할 수 있는 읽기 요청 유닛의 최대 수에 영향을 미칩니다.

단위: `Count`

유효한 통계:

- `Maximum` - 계정의 테이블 또는 글로벌 보조 인덱스에서 사용할 수 있는 읽기 용량 단위의 최대 수입니다.

AccountMaxTableLevelWrites

계정의 테이블 또는 글로벌 보조 인덱스에서 사용할 수 있는 쓰기 용량 유닛의 최대 수입니다. 온디맨드 테이블의 경우 이 제한이 테이블 또는 글로벌 보조 인덱스에서 사용할 수 있는 쓰기 요청 유닛의 최대 수에 영향을 미칩니다.

단위: `Count`

유효한 통계:

- `Maximum` - 계정의 테이블 또는 글로벌 보조 인덱스에서 사용할 수 있는 쓰기 용량 단위의 최대 수입니다.

AccountMaxWrites

계정에서 사용할 수 있는 쓰기 용량 유닛의 최대 수입니다. 이 제한은 온디맨드 테이블 또는 글로벌 보조 인덱스에는 적용되지 않습니다.

단위: `Count`

유효한 통계:

- `Maximum` - 계정에서 사용할 수 있는 쓰기 용량 단위의 최대 수입니다.

AccountProvisionedReadCapacityUtilization

계정에서 사용하는 프로비저닝된 읽기 용량 단위의 백분율입니다.

단위: Percent

유효한 통계:

- Maximum - 계정에서 사용하는 프로비저닝된 읽기 용량 단위의 최대 백분율입니다.
- Minimum - 계정에서 사용하는 프로비저닝된 읽기 용량 단위의 최소 백분율입니다.
- Average - 계정에서 사용하는 프로비저닝된 읽기 용량 단위의 평균 백분율입니다. 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 읽기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

### AccountProvisionedWriteCapacityUtilization

계정에서 사용하는 프로비저닝된 쓰기 용량 단위의 백분율입니다.

단위: Percent

유효한 통계:

- Maximum - 계정에서 사용하는 프로비저닝된 쓰기 용량 단위의 최대 백분율입니다.
- Minimum - 계정에서 사용하는 프로비저닝된 쓰기 용량 단위의 최소 백분율입니다.
- Average - 계정에서 사용하는 프로비저닝된 쓰기 용량 단위의 평균 백분율입니다. 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 쓰기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

### AgeOfOldestUnreplicatedRecord

아직 Kinesis 데이터 스트림에 복제되지 않은 레코드가 DynamoDB 테이블에 처음 나타난 이후의 경과 시간입니다.

단위: Milliseconds

차원: TableName, DelegatedOperation

유효한 통계:

- Maximum.
- Minimum.
- Average.

### ConditionalCheckFailedRequests

조건부 쓰기 실패 횟수입니다. `PutItem`, `UpdateItem` 및 `DeleteItem` 작업에서는 해당 작업이 진행하려면 먼저 true로 평가되어야 하는 논리적 조건을 사용자가 제공하도록 합니다. 이 조건이 false로 평가되면 `ConditionalCheckFailedRequests`가 1씩 증분됩니다. `ConditionalCheckFailedRequests`도 논리적 조건이 제공되고 해당 조건이 false로 평가되는 PartiQL Update 및 Delete 문에 대해 1씩 증분됩니다.

> **Note**
>
> 조건부 쓰기가 실패하면 HTTP 400 오류(잘못된 요청)가 발생합니다. 이러한 이벤트는 `ConditionalCheckFailedRequests` 지표에만 반영되고, `UserErrors` 지표에는 반영되지 않습니다.

단위: `Count`

차원: `TableName`

유효한 통계:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

### ConsumedChangeDataCaptureUnits

사용된 변경 데이터 캡처 단위의 수입니다.

단위: `Count`

차원: `TableName, DelegatedOperation`

유효한 통계:

- `Minimum`
- `Maximum`

- `Average`

ConsumedReadCapacityUnits

일정 기간 동안 사용된 프로비저닝 용량과 온디맨드 용량의 읽기 용량 단위 수로, 이를 통해 처리량이 얼마나 많이 사용되는지 추적할 수 있습니다. 또한 테이블과 테이블의 모든 글로벌 보조 인덱스 또는 특정 글로벌 보조 인덱스에 대해 소비된 총 읽기 용량을 가져올 수 있습니다. 자세한 내용은 읽기/쓰기 용량 모드를 참조하세요.

테이블일 때는 `TableName` 차원이 `ConsumedReadCapacityUnits`를 반환하지만, 글로벌 보조 인덱스일 때는 그렇지 않습니다. 글로벌 보조 인덱스일 때 `ConsumedReadCapacityUnits`를 확인하려면 `TableName`과 `GlobalSecondaryIndexName`를 모두 지정해야 합니다.

> **Note**
>
> 즉, 단 1초의 짧은 시간 동안 급증하는 용량 소비는 CloudWatch 그래프에 정확하게 반영되지 않아 해당 분 동안 표시 소비율이 낮아질 수 있습니다.
> `Sum` 통계를 사용하여 사용된 처리량을 계산합니다. 예를 들어, 1분 동안의 `Sum` 값을 가져와서 이를 1분의 초 수(60)로 나누어 초당 평균 `ConsumedReadCapacityUnits`를 계산합니다. 계산된 값을 DynamoDB에 제공하는 프로비저닝된 처리량 값과 비교할 수 있습니다.

단위: `Count`

차원: `TableName, GlobalSecondaryIndexName`

유효한 통계:

- `Minimum` - 테이블 또는 인덱스에 대한 개별 요청에 의해 사용된 읽기 용량 단위의 최소 수입니다.
- `Maximum` - 테이블 또는 인덱스에 대한 개별 요청에 의해 사용된 읽기 용량 단위의 최대 수입니다.
- `Average` - 사용된 요청당 읽기 용량 평균입니다.

  > **Note**
  >
  > `Average` 값은 샘플 값이 0이 될 비활동 기간의 영향을 받습니다.

- `Sum` - 사용된 총 읽기 용량 단위입니다. `ConsumedReadCapacityUnits` 지표에 가장 유용한 통계입니다.

- SampleCount – 지표를 내보내는 빈도를 나타냅니다. 트래픽이 없는 테이블에서도 정기적으로 SampleCount를 내보내지만 샘플 값은 항상 0입니다.

> **Note**
>
> SampleCount 값은 샘플 값이 0이 될 비활동 기간의 영향을 받습니다.

ConsumedWriteCapacityUnits

일정 기간 동안 사용된 프로비저닝 용량과 온디맨드 용량의 쓰기 용량 단위 수로, 이를 통해 처리량이 얼마나 많이 사용되는지 추적할 수 있습니다. 또한 테이블과 테이블의 모든 글로벌 보조 인덱스 또는 특정 글로벌 보조 인덱스에 대해 소비된 총 쓰기 용량을 가져올 수 있습니다. 자세한 내용은 읽기/쓰기 용량 모드를 참조하세요.

테이블일 때는 TableName 차원이 ConsumedWriteCapacityUnits를 반환하지만, 글로벌 보조 인덱스일 때는 그렇지 않습니다. 글로벌 보조 인덱스일 때 ConsumedWriteCapacityUnits를 확인하려면 TableName과 GlobalSecondaryIndexName를 모두 지정해야 합니다.

> **Note**
>
> Sum 통계를 사용하여 사용된 처리량을 계산합니다. 예를 들어 1분간 Sum 값을 가져와 60초로 나눠 초당 평균 ConsumedWriteCapacityUnits를 계산합니다(이 평균값은 1분간 발생한 쓰기 작업이 많지만 짧은 순간에 급증한다는 것을 의미하지는 않음). 계산된 값을 DynamoDB에 제공하는 프로비저닝된 처리량 값과 비교할 수 있습니다.

단위: Count

차원: TableName, GlobalSecondaryIndexName

유효한 통계:

- Minimum - 테이블 또는 인덱스에 대한 개별 요청에 의해 사용된 쓰기 용량 단위의 최소 수입니다.
- Maximum - 테이블 또는 인덱스에 대한 개별 요청에 의해 사용된 쓰기 용량 단위의 최대 수입니다.
- Average - 사용된 요청당 쓰기 용량 평균입니다.

> **Note**
>
> Average 값은 샘플 값이 0이 될 비활동 기간의 영향을 받습니다.

- Sum - 사용된 총 쓰기 용량 단위입니다. ConsumedWriteCapacityUnits 지표에 가장 유용한 통계입니다.
- SampleCount – 지표를 내보내는 빈도를 나타냅니다. 트래픽이 없는 테이블에서도 정기적으로 SampleCount를 내보내지만 샘플 값은 항상 0입니다.

> **Note**
>
> SampleCount 값은 샘플 값이 0이 될 비활동 기간의 영향을 받습니다.

FailedToReplicateRecordCount

DynamoDB가 Kinesis 데이터 스트림으로 복제하지 못한 레코드 수입니다.

단위: Count

Dimensions: TableName, DelegatedOperation

유효한 통계:

- Sum

MaxProvisionedTableReadCapacityUtilization

계정의 가장 높은 프로비저닝된 읽기 테이블 또는 글로벌 보조 인덱스에서 사용하는 프로비저닝된 읽기 용량의 백분율입니다.

단위: Percent

유효한 통계:

- Maximum – 계정의 가장 높은 프로비저닝된 읽기 테이블 또는 글로벌 보조 인덱스에서 사용하는 프로비저닝된 읽기 용량 단위의 최대 백분율입니다.
- Minimum – 계정의 가장 높은 프로비저닝된 읽기 테이블 또는 글로벌 보조 인덱스에서 사용하는 프로비저닝된 읽기 용량 단위의 최소 백분율입니다.

- Average - 계정의 가장 높은 프로비저닝된 읽기 테이블 또는 글로벌 보조 인덱스에서 사용하는 프로비저닝된 읽기 용량 단위의 평균 백분율입니다. 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 읽기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

MaxProvisionedTableWriteCapacityUtilization

계정의 가장 높은 프로비저닝된 쓰기 테이블 또는 글로벌 보조 인덱스에서 사용하는 프로비저닝된 쓰기 용량의 백분율입니다.

단위: Percent

유효한 통계:

- Maximum - 계정의 가장 높은 프로비저닝된 쓰기 테이블 또는 글로벌 보조 인덱스에서 사용하는 프로비저닝된 쓰기 용량 단위의 최대 백분율입니다.
- Minimum - 계정의 가장 높은 프로비저닝된 쓰기 테이블 또는 글로벌 보조 인덱스에서 사용하는 프로비저닝된 쓰기 용량 단위의 최소 백분율입니다.
- Average - 계정의 가장 높은 프로비저닝된 쓰기 테이블 또는 글로벌 보조 인덱스에서 사용하는 프로비저닝된 쓰기 용량 단위의 평균 백분율입니다. 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 쓰기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

OnDemandMaxReadRequestUnits

테이블 또는 글로벌 보조 인덱스에 대해 지정된 온디맨드 읽기 요청 단위 수입니다.

테이블의 OnDemandMaxReadRequestUnits를 보려면 TableName을 지정해야 합니다. 글로벌 보조 인덱스일 때 OnDemandMaxReadRequestUnits를 확인하려면 TableName과 GlobalSecondaryIndexName를 모두 지정해야 합니다.

단위: 개

Dimensions: TableName, GlobalSecondaryIndexName

유효한 통계:

- Minimum - 온디맨드 읽기 요청 단위에 대한 가장 낮은 설정입니다. UpdateTable을 사용하여 읽기 요청 단위를 늘리는 경우 이 지표는 해당 기간 동안 온디맨드 ReadRequestUnits의 최저값을 보여줍니다.

- Maximum - 온디맨드 읽기 요청 단위에 대한 가장 높은 설정입니다. UpdateTable을 사용하여 읽기 요청 단위를 줄이는 경우 이 지표는 해당 기간 동안 온디맨드 ReadRequestUnits의 최고값을 보여 줍니다.
- Average - 평균 온디맨드 읽기 요청 단위입니다. OnDemandMaxReadRequestUnits 지표는 5분 간격으로 게시됩니다. 따라서 온디맨드 읽기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

### OnDemandMaxWriteRequestUnits

테이블 또는 글로벌 보조 인덱스에 대해 지정된 온디맨드 쓰기 요청 단위 수입니다.

테이블의 OnDemandMaxWriteRequestUnits를 보려면 TableName을 지정해야 합니다. 글로벌 보조 인덱스일 때 OnDemandMaxWriteRequestUnits를 확인하려면 TableName과 GlobalSecondaryIndexName를 모두 지정해야 합니다.

단위: Count

Dimensions: TableName, GlobalSecondaryIndexName

유효한 통계:

- Minimum - 온디맨드 쓰기 요청 단위에 대한 가장 낮은 설정입니다. UpdateTable을 사용하여 쓰기 요청 단위를 늘리는 경우 이 지표는 해당 기간 동안 온디맨드 WriteRequestUnits의 최저값을 보여 줍니다.
- Maximum - 온디맨드 쓰기 요청 단위에 대한 가장 높은 설정입니다. UpdateTable을 사용하여 쓰기 요청 단위를 줄이는 경우 이 지표는 해당 기간 동안 온디맨드 WriteRequestUnits의 최고값을 보여 줍니다.
- Average - 평균 온디맨드 쓰기 요청 단위입니다. OnDemandMaxWriteRequestUnits 지표는 5분 간격으로 게시됩니다. 따라서 온디맨드 쓰기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

### OnlineIndexConsumedWriteCapacity

새 글로벌 보조 인덱스를 테이블에 추가할 때 사용되는 쓰기 용량 단위의 수입니다. 인덱스의 쓰기 용량이 너무 낮으면 채움(backfill) 단계에서 수신되는 쓰기 작업이 병목 현상을 일으켜 인덱스 생성 시간이 늘어날 수 있습니다. 따라서 인덱스 빌드 중에는 이 통계를 모니터링하여 인덱스 쓰기 용량의 언더프로비저닝 여부를 확인해야 합니다.

인덱스가 계속 빌드 중인 경우에도 UpdateTable 작업을 사용하여 인덱스의 쓰기 용량을 조정할 수 있습니다.

인덱스 생성 중 사용된 쓰기 처리량은 인덱스의 ConsumedWriteCapacityUnits 지표에 포함되지 않습니다.

> **Note**
>
> 새 글로벌 보조 인덱스의 채우기 단계가 빠르게(몇 분 미만) 완료되는 경우 이 지표가 생성되지 않을 수 있습니다. 이는 기본 테이블에 인덱스에서 채울 항목이 거의 없거나 전혀 없는 경우에 발생할 수 있습니다.

단위: Count

차원: TableName, GlobalSecondaryIndexName

유효한 통계:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

### OnlineIndexPercentageProgress

새 글로벌 보조 인덱스가 테이블에 추가되고 있는 진행률입니다. DynamoDB는 먼저 리소스를 새 인덱스에 할당한 다음 테이블의 속성을 인덱스에 채워야 합니다. 테이블 용량이 클 때는 이 프로세스 시간이 오래 걸릴 수도 있습니다. DynamoDB가 인덱스를 빌드할 때는 이 통계를 모니터링하여 상대적인 진행률을 확인해야 합니다.

단위: Count

차원: TableName, GlobalSecondaryIndexName

유효한 통계:

- Minimum

- Maximum
- Average
- SampleCount
- Sum

### OnlineIndexThrottleEvents

새 글로벌 보조 인덱스를 테이블에 추가할 때 발생하는 쓰기 병목 이벤트의 수입니다. 이러한 이벤트는 수신 쓰기 작업이 인덱스의 프로비저닝된 쓰기 처리량을 초과하기 때문에 인덱스를 생성할 때 시간이 더 걸릴 것임을 나타냅니다.

인덱스가 계속 빌드 중인 경우에도 UpdateTable 작업을 사용하여 인덱스의 쓰기 용량을 조정할 수 있습니다.

인덱스 생성 중 발생한 제한 이벤트는 인덱스의 WriteThrottleEvents 지표에 포함되지 않습니다.

단위: Count

차원: TableName, GlobalSecondaryIndexName

유효한 통계:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

### PendingReplicationCount

글로벌 테이블 버전 2017.11.29(레거시)에 대한 지표(글로벌 테이블만 해당) 한 복제본 테이블에 쓰여졌지만 전역 테이블의 다른 복제본에는 아직 쓰여지지 않은 항목 업데이트 수입니다.

단위: Count

차원: TableName, ReceivingRegion

유효한 통계:

- Average

- Sample Count
- Sum

### ProvisionedReadCapacityUnits

테이블 또는 글로벌 보조 인덱스에 대해 프로비저닝된 읽기 용량 단위의 수입니다. 테이블일 때는 TableName 차원이 ProvisionedReadCapacityUnits를 반환하지만, 글로벌 보조 인덱스일 때는 그렇지 않습니다. 글로벌 보조 인덱스일 때 ProvisionedReadCapacityUnits를 확인하려면 TableName과 GlobalSecondaryIndexName를 모두 지정해야 합니다.

단위: Count

차원: TableName, GlobalSecondaryIndexName

유효한 통계:

- Minimum - 프로비저닝된 읽기 용량에 대한 가장 낮은 설정입니다. UpdateTable을 사용하여 읽기 용량을 늘리는 경우 이 지표는 해당 기간 동안 프로비저닝된 ReadCapacityUnits의 최저값을 보여 줍니다.
- Maximum - 프로비저닝된 읽기 용량에 대한 가장 높은 설정입니다. UpdateTable을 사용하여 읽기 용량을 줄이는 경우 이 지표는 해당 기간 동안 프로비저닝된 ReadCapacityUnits의 최고값을 보여 줍니다.
- Average - 프로비저닝된 읽기 용량 평균입니다. ProvisionedReadCapacityUnits 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 읽기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

### ProvisionedWriteCapacityUnits

테이블 또는 글로벌 보조 인덱스에 대해 프로비저닝된 쓰기 용량 단위의 수입니다.

테이블일 때는 TableName 차원이 ProvisionedWriteCapacityUnits를 반환하지만, 글로벌 보조 인덱스일 때는 그렇지 않습니다. 글로벌 보조 인덱스일 때 ProvisionedWriteCapacityUnits를 확인하려면 TableName과 GlobalSecondaryIndexName를 모두 지정해야 합니다.

단위: Count

차원: TableName, GlobalSecondaryIndexName

유효한 통계:

- Minimum - 프로비저닝된 쓰기 용량에 대한 가장 낮은 설정입니다. UpdateTable을 사용하여 쓰기 용량을 늘리는 경우 이 지표는 해당 기간 동안 프로비저닝된 WriteCapacityUnits의 최저값을 보여 줍니다.
- Maximum - 프로비저닝된 쓰기 용량에 대한 가장 높은 설정입니다. UpdateTable을 사용하여 쓰기 용량을 줄이는 경우 이 지표는 해당 기간 동안 프로비저닝된 WriteCapacityUnits의 최고값을 보여 줍니다.
- Average - 프로비저닝된 쓰기 용량 평균입니다. ProvisionedWriteCapacityUnits 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 쓰기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

### ReadThrottleEvents

테이블 또는 글로벌 보조 인덱스에 대해 프로비저닝된 읽기 용량 단위를 초과하는 DynamoDB에 대한 요청입니다.

단일 요청으로 여러 이벤트가 발생할 수 있습니다. 예를 들어 10개 항목을 읽는 BatchGetItem이 10개의 GetItem 이벤트로 처리됩니다. 그리고 각 이벤트마다 병목 현상이 발생하면 ReadThrottleEvents가 1씩 증분됩니다. 모두 10개의 GetItem 이벤트에서 병목 현상이 일어나지 않는 한 전체 BatchGetItem의 ThrottledRequests 지표는 증분되지 않습니다.

테이블일 때는 TableName 차원이 ReadThrottleEvents를 반환하지만, 글로벌 보조 인덱스일 때는 그렇지 않습니다. 글로벌 보조 인덱스일 때 ReadThrottleEvents를 확인하려면 TableName과 GlobalSecondaryIndexName를 모두 지정해야 합니다.

단위: Count

차원: TableName, GlobalSecondaryIndexName

유효한 통계:

- SampleCount
- Sum

### ReplicationLatency

(이 지표는 DynamoDB 전역 테이블용입니다.) 하나의 복제본 테이블에 대한 DynamoDB 스트림에 나타나는 업데이트된 항목과 전역 테이블의 다른 복제본에 나타나는 해당 항목 사이의 경과된 시간입니다.

단위: Milliseconds

차원: TableName, ReceivingRegion

유효한 통계:

- Average
- Minimum
- Maximum

### ReturnedBytes

지정된 기간 동안 GetRecords 작업(Amazon DynamoDB Streams)에 의해 반환되는 바이트 수입니다.

단위: Bytes

차원: Operation, StreamLabel, TableName

유효한 통계:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

### ReturnedItemCount

지정된 기간 동안 Query, Scan 또는 ExecuteStatement(select) 작업에 의해 반환되는 항목 수입니다.

반환되는 항목 수가 평가된 항목 수와 반드시 일치하지는 않습니다. 예를 들어 항목이 100개인 테이블 또는 인덱스에 대해 Scan을 요청했지만 결과를 좁히는 FilterExpression을 지정하여 15개의 항목만 반환되도록 한 경우, Scan의 응답에 100개 ScanCount 및 15개 Count의 반환된 항목이 포함됩니다.

단위: Count

차원: `TableName, Operation`

유효한 통계:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedRecordsCount

지정된 기간 동안 `GetRecords` 작업(Amazon DynamoDB Streams)에 의해 반환되는 스트림 레코드 수입니다.

단위: `Count`

차원: `Operation, StreamLabel, TableName`

유효한 통계:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

SuccessfulRequestLatency

지정된 기간 동안 DynamoDB 또는 Amazon DynamoDB Streams에 대해 성공한 요청의 지연 시간입니다. `SuccessfulRequestLatency`는 다음과 같이 두 가지 종류의 정보를 제공할 수 있습니다.

- 성공한 요청의 경과 시간(`Minimum`, `Maximum`, `Sum`, `Average` 또는 `Percentile`)입니다.
- 성공한 요청의 수(`SampleCount`)

`SuccessfulRequestLatency`는 DynamoDB 또는 Amazon DynamoDB Streams 내의 활동만 반영하며, 네트워크 지연 시간이나 클라이언트 측 활동은 고려하지 않습니다.

> **Note**
>
> 사용자 지정 백분위수 값(예: p99.9)을 분석하려면 CloudWatch 지표 통계 필드에 원하는 백분위수(예: p99.9)를 수동으로 입력합니다. 이를 통해 드롭다운에 나열된 기본 백분위수를 초과하는 지연 시간 분포를 평가할 수 있습니다.

단위: `Milliseconds`

차원: `TableName, Operation, StreamLabel`

유효한 통계:

- `Minimum`
- `Maximum`
- `Sum`
- `Average`
- `Percentile`
- `SampleCount`

SystemErrors

지정된 기간 동안 HTTP 500 상태 코드를 생성하는 DynamoDB 또는 Amazon DynamoDB Streams에 대한 요청입니다. HTTP 500은 대개 내부 서비스 오류를 나타냅니다.

> **Note**
>
> DynamoDB가 시스템 오류(HTTP 500)를 반환하면 대부분의 AWS SDK가 구성 가능한 횟수의 재시도를 자동으로 수행합니다. 재시도 중에 문제가 해결되면 오류가 나타나지 않고 애플리케이션이 계속 실행되며 클라이언트 측에서 인식되는 지연 시간이 늘어날 수 있습니다. 모든 재시도 후에도 오류가 지속되면 해당 오류가 애플리케이션 코드로 전파됩니다.

단위: `Count`

차원: `TableName, Operation`

유효한 통계:

- Sum
- SampleCount

TimeToLiveDeletedItemCount

지정된 기간 동안 TTL(Time To Live)에서 삭제된 항목 수입니다. 이 지표는 테이블에서 TTL 삭제 비율을 모니터링하는 데 효과적입니다.

단위: Count

측정 기준: TableName

유효한 통계:

- Sum

ThrottledPutRecordCount

Kinesis Data Streams 용량이 부족하여 Kinesis Data Streams에 복제하지 못한 레코드 수입니다.

단위: Count

차원: TableName, DelegatedOperation

유효한 통계:

- Minimum
- Maximum
- Average
- SampleCount

ThrottledRequests

리소스(테이블 또는 인덱스 등)에 대해 프로비저닝된 처리량 제한을 초과하는 DynamoDB에 대한 요청입니다.

요청에 포함된 이벤트가 프로비저닝된 처리량 제한을 초과하면 ThrottledRequests가 1씩 증분됩니다. 예를 들어 테이블의 항목을 글로벌 보조 인덱스로 업데이트하는 경우 테이블에 쓰기, 각 인덱스

에 쓰기 등과 같은 여러 이벤트가 발생합니다. 이때 이 이벤트 중 하나 이상에서 병목 현상이 발생하면 ThrottledRequests가 1씩 증분됩니다.

> **Note**
>
> 배치 요청(BatchGetItem 또는 BatchWriteItem)에서는 배치의 모든 요청에서 병목 현상이 발생하는 경우에만 ThrottledRequests가 증분됩니다.
> 배치 내의 개별 요청에서 병목 현상이 발생하면 다음 지표 중 하나가 증분됩니다.
>
> - ReadThrottleEvents - BatchGetItem 내의 GetItem 이벤트에서 병목 현상이 발생한 경우
> - WriteThrottleEvents - BatchWriteItem 내의 PutItem 또는 DeleteItem 이벤트에서 병목 현상이 발생한 경우

요청의 병목 현상 원인이 되는 이벤트는 ThrottledRequests를 테이블과 테이블 인덱스의 ReadThrottleEvents 및 WriteThrottleEvents와 비교해보면 알 수 있습니다.

> **Note**
>
> 병목 현상이 발생한 요청에 의해 HTTP 400 상태 코드가 생성됩니다. 이러한 이벤트는 모두 ThrottledRequests 지표에만 반영되고 UserErrors 지표에는 반영되지 않습니다.

단위: Count

차원: TableName, Operation

유효한 통계:

- Sum
- SampleCount

TransactionConflict

동일한 항목의 동시 요청 간 트랜잭션 충돌로 인해 거부된 항목 수준의 요청입니다. 자세한 내용은 DynamoDB의 트랜잭션 충돌 처리를 참조하세요.

단위: Count

차원: TableName

유효한 통계:

- Sum - 트랜잭션 충돌로 인해 거부된 항목 수준의 요청 수입니다.

  > **Note**
  >
  > TransactWriteItems 또는 TransactGetItems에 대한 호출 내 여러 항목 수준 요청이 거부된 경우 각 항목 수준 Put, Update, Delete 또는 Get에 대해 1씩 Sum이 증분됩니다.

- SampleCount - 트랜잭션 충돌로 인해 거부된 요청 수입니다.

  > **Note**
  >
  > TransactWriteItems 또는 TransactGetItems에 대한 호출 내 여러 항목 수준 요청이 거부된 경우 SampleCount만 1씩 증분됩니다.

- Min - TransactWriteItems, TransactGetItems, PutItem, UpdateItem 또는 DeleteItem에 대한 호출 내 거부된 항목 수준 요청의 최소 수입니다.
- Max - TransactWriteItems, TransactGetItems, PutItem, UpdateItem 또는 DeleteItem에 대한 호출 내 거부된 항목 수준 요청의 최대 수입니다.
- Average - TransactWriteItems, TransactGetItems, PutItem, UpdateItem 또는 DeleteItem에 대한 호출 내 거부된 항목 수준 요청의 평균 수입니다.

### UserErrors

지정된 기간 동안 HTTP 400 상태 코드를 생성하는 DynamoDB 또는 Amazon DynamoDB Streams에 대한 요청입니다. HTTP 400은 유효하지 않은 파라미터 조합, 존재하지 않는 테이블을 업데이트하려는 시도, 잘못된 요청 서명 등 대개 클라이언트 측 오류를 나타냅니다.

UserErrors와 관련된 지표를 로깅하는 예외의 몇 가지 예는 다음과 같습니다.

- ResourceNotFoundException
- ValidationException
- TransactionConflict

이러한 이벤트는 모두 다음을 제외하고 UserErrors 지표에 반영됩니다.

- ProvisionedThroughputExceededException - 이 단원의 ThrottledRequests 지표를 참조하세요.
- ConditionalCheckFailedException - 이 단원의 ConditionalCheckFailedRequests 지표를 참조하세요.

UserErrors는 현재 AWS 리전 및 현재 AWS 계정의 DynamoDB 또는 Amazon DynamoDB Streams 요청에 대한 HTTP 400 오류의 집계를 나타냅니다.

단위: Count

유효한 통계:

- Sum
- SampleCount

WriteThrottleEvents

테이블 또는 글로벌 보조 인덱스에 대해 프로비저닝된 쓰기 용량 단위를 초과하는 DynamoDB에 대한 요청입니다.

단일 요청으로 여러 이벤트가 발생할 수 있습니다. 예를 들어 글로벌 보조 인덱스를 사용하는 테이블에 대한 PutItem 요청으로 테이블 쓰기, 각 3개의 인덱스 쓰기와 같은 4개 이벤트가 발생합니다. 그리고 각 이벤트마다 병목 현상이 발생하면 WriteThrottleEvents 지표가 1씩 증분됩니다. 하나의 PutItem 요청에 대해 어떤 이벤트에서든지 병목 현상이 발생하면 ThrottledRequests 역시 1씩 증분됩니다. BatchWriteItem에 대해서는 모든 개별 PutItem 또는 DeleteItem 이벤트에서 병목 현상이 일어나지 않는 한 전체 BatchWriteItem의 ThrottledRequests 지표는 증분되지 않습니다.

테이블일 때는 TableName 차원이 WriteThrottleEvents를 반환하지만, 글로벌 보조 인덱스일 때는 그렇지 않습니다. 글로벌 보조 인덱스일 때 WriteThrottleEvents를 확인하려면 TableName과 GlobalSecondaryIndexName를 모두 지정해야 합니다.

단위: Count

차원: TableName, GlobalSecondaryIndexName

유효한 통계:

- Sum

- SampleCount

사용량 지표

CloudWatch의 사용량 지표를 사용하면 CloudWatch 콘솔에서 지표를 시각화하고 사용자 지정 대시보드를 생성하며 CloudWatch 이상 탐지를 통해 활동의 변화를 감지하고 사용량이 임계값에 근접할 때 이를 알려 주는 경보를 구성함으로써 사용량을 사전에 관리할 수 있습니다.

또한 DynamoDB는 이러한 사용량 지표를 Service Quotas와 통합합니다. CloudWatch를 사용하여 계정의 서비스 할당량 사용을 관리할 수 있습니다. 자세한 내용은 서비스 할당량 시각화 및 경보 설정을 참조하세요.

사용 가능한 사용량 지표 목록

- AccountProvisionedWriteCapacityUnits
- AccountProvisionedReadCapacityUnits
- TableCount

AccountProvisionedWriteCapacityUnits

계정의 모든 테이블 및 글로벌 보조 인덱스에 대해 프로비저닝된 쓰기 용량 단위의 합계입니다.

단위: Count

유효한 통계:

- Minimum - 일정 기간 동안 프로비저닝된 쓰기 용량 단위의 가장 큰 수입니다.
- Maximum - 일정 기간 동안 프로비저닝된 쓰기 용량 단위의 가장 작은 수입니다.
- Average - 일정 기간 동안 계정에 프로비저닝된 쓰기 용량 단위의 평균 수입니다.

이 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 쓰기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

AccountProvisionedReadCapacityUnits

계정의 모든 테이블 및 글로벌 보조 인덱스에 대해 프로비저닝된 읽기 용량 단위의 합계입니다.

단위: Count

유효한 통계:

- `Minimum` - 일정 기간 동안 프로비저닝된 읽기 용량 단위의 가장 작은 수입니다.
- `Maximum` - 일정 기간 동안 프로비저닝된 읽기 용량 단위의 가장 큰 수입니다.
- `Average` - 일정 기간 동안 계정에 프로비저닝된 읽기 용량 단위의 평균 수입니다.

이 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 읽기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

TableCount

계정의 활성 테이블 수입니다.

단위: `Count`

유효한 통계:

- `Minimum` - 일정 기간 동안 테이블의 가장 적은 수입니다.
- `Maximum` - 일정 기간 동안 테이블의 가장 큰 수입니다.
- `Average` - 일정 기간 동안 테이블의 평균 수입니다.

## DynamoDB의 지표 및 차원 이해

DynamoDB의 지표는 계정, 테이블 이름, 글로벌 보조 인덱스 이름 또는 작업 값으로 한정됩니다. CloudWatch 콘솔을 사용하면 아래 표의 어떤 차원이든지 함께 DynamoDB 데이터를 가져올 수 있습니다.

사용 가능한 차원 목록

- DelegatedOperation
- GlobalSecondaryIndexName
- Operation
- OperationType
- 동사
- ReceivingRegion
- StreamLabel

- TableName

DelegatedOperation

이 차원은 DynamoDB가 사용자 대신 수행하는 작업에 대한 데이터를 제한합니다. 다음 작업은 캡처됩니다.

- Kinesis Data Streams에 대한 변경 데이터 캡처

GlobalSecondaryIndexName

이 차원은 테이블의 글로벌 보조 인덱스에 대한 데이터를 제한합니다. `GlobalSecondaryIndexName`을 지정하면 `TableName`도 함께 지정해야 합니다.

Operation

이 차원은 다음의 DynamoDB 작업 중 하나에 대한 데이터를 제한합니다.

- `PutItem`
- `DeleteItem`
- `UpdateItem`
- `GetItem`
- `BatchGetItem`
- `Scan`
- `Query`
- `BatchWriteItem`
- `TransactWriteItems`
- `TransactGetItems`
- `ExecuteTransaction`
- `BatchExecuteStatement`
- `ExecuteStatement`

또한 다음의 Amazon DynamoDB Streams 작업에 대한 데이터를 제한할 수도 있습니다.

- `GetRecords`

OperationType

이 차원은 다음 작업 유형 중 하나에 대한 데이터를 제한합니다.

- `Read`
- `Write`

이 차원은 `ExecuteTransaction` 및 `BatchExecuteStatement` 요청에 대해 내보내집니다.

동사

이 차원은 다음의 DynamoDB PartiQL 동사 중 하나에 대한 데이터를 제한합니다.

- Insert: `PartiQLInsert`
- Select: `PartiQLSelect`
- Update: `PartiQLUpdate`
- Delete: `PartiQLDelete`

이 차원은 `ExecuteStatement` 작업에 대해 내보내집니다.

ReceivingRegion

이 차원은 데이터를 특정 AWS 리전으로 제한합니다. 이 지표는 DynamoDB 전역 테이블 내 복제 테이블에서 비롯된 지표와 함께 사용됩니다.

StreamLabel

이 차원은 특정 스트림 라벨에 대한 데이터를 제한합니다. 이것은 Amazon DynamoDB Streams `GetRecords` 작업에서 비롯된 지표와 함께 사용됩니다.

TableName

이 차원은 특정 테이블에 대한 데이터를 제한합니다. 이 값은 현재 리전 및 현재 AWS 계정의 테이블 이름일 수 있습니다.

## DynamoDB에 CloudWatch 경보 생성

CloudWatch 경보는 지정한 기간에 단일 지표를 감시하고 시간에 따른 임계값에 대한 지표 값을 기준으로 지정된 작업을 하나 이상 수행합니다. 이 작업은 Amazon SNS 주제 또는 Auto Scaling 정책으로 전송되는 알림입니다. 또한 대시보드에 경보를 추가할 수 있으므로 여러 리전에 걸쳐 AWS 리소스 및

애플리케이션에 대한 경보를 모니터링하고 수신할 수 있습니다. 생성할 수 있는 경보 수에는 제한이 없습니다. CloudWatch 경보는 특정 상태에 있다는 이유만으로는 작업을 호출하지 않습니다. 상태가 변경되고 지정한 기간 동안 유지되어야 합니다. 권장 DynamoDB 경보 목록은 권장 경보를 참조하세요.

> **Note**
>
> CloudWatch는 누락된 차원의 지표를 집계하지 않으므로 CloudWatch 경보를 생성할 때 필요한 차원을 모두 지정해야 합니다. 차원이 누락된 CloudWatch 경보를 생성해도 경보를 생성할 때 오류가 발생하지 않습니다.

예를 들어, 읽기 용량 단위 5개로 구성된 프로비저닝된 테이블이 있습니다. 프로비저닝된 전체 읽기 용량을 사용하기 전에 알림을 받고 싶기 때문에 소비된 용량이 테이블에 프로비저닝한 용량의 80%에 도달하면 알림을 받도록 CloudWatch 경보를 생성하기로 결정했습니다. CloudWatch 콘솔 또는 AWS CLI를 사용하여 경보를 생성할 수 있습니다.

## CloudWatch 콘솔에서 경보 생성

CloudWatch 콘솔에서 경보를 생성하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 Alarms, All alarms를 선택합니다.
3. Create alarm(경보 생성)을 선택하세요.
4. 지표 이름 열에서 모니터링하려는 테이블과 **`ConsumeReadCapacityUnits`**이 있는 행을 찾습니다. 이 행 옆의 확인란을 선택하고 지표 선택을 선택합니다.
5. 지표 및 조건 지정의 통계에서 합계를 선택합니다. 기간에서 1분을 선택합니다.
6. 조건에서 다음을 지정합니다.
   a. 임곗값 유형에서 정적을 선택합니다.
   b. **`ConsumedReadCapacityUnits`**이(가) 다음과 같은 경우에 항상에서 크거나 같음을 선택하고 임계값으로 240을 지정합니다.
7. Next(다음)를 선택합니다.
8. 알림에서 **`In alarm`**을 선택하고 경보가 `ALARM` 상태일 때 알릴 SNS 주제를 선택합니다.
9. 마친 후에는 다음을 선택합니다.
10. 경보의 이름과 설명을 입력하고 다음을 선택합니다.

11. 미리 보기 및 생성에서 정보 및 조건이 원하는 내용인지 확인한 다음 경보 생성을 선택합니다.

## AWS CLI에서 경보 생성

```
aws cloudwatch put-metric-alarm \
    -\-alarm-name ReadCapacityUnitsLimitAlarm \
    -\-alarm-description "Alarm when read capacity reaches 80% of my provisioned read
 capacity" \
    -\-namespace AWS/DynamoDB \
    -\-metric-name ConsumedReadCapacityUnits \
    -\-dimensions Name=TableName,Value=myTable \
    -\-statistic Sum \
    -\-threshold 240 \
    -\-comparison-operator GreaterThanOrEqualToThreshold \
    -\-period 60 \
    -\-evaluation-periods 1 \
    -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
```

경보를 테스트합니다.

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value OK
```

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value ALARM
```

## 추가 AWS CLI 예제

다음 절차에서는 요청이 테이블의 프로비저닝된 처리량 할당량을 초과하는 경우 알림을 받는 방법을 설명합니다.

1. Amazon SNS 주제 `arn:aws:sns:us-east-1:123456789012:requests-exceeding-throughput`을 생성합니다. 자세한 내용은 Amazon Simple Notification Service 설정을 참조하세요.
2. 경보를 만듭니다.

```
aws cloudwatch put-metric-alarm \
        -\-alarm-name ReadCapacityUnitsLimitAlarm \
        -\-alarm-description "Alarm when read capacity reaches 80% of my
 provisioned read capacity" \
```

```
        -\-namespace AWS/DynamoDB \
        -\-metric-name ConsumedReadCapacityUnits \
        -\-dimensions Name=TableName,Value=myTable \
        -\-statistic Sum \
        -\-threshold 240 \
        -\-comparison-operator GreaterThanOrEqualToThreshold \
        -\-period 60 \
        -\-evaluation-periods 1 \
        -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
```

3. 경보를 테스트합니다.

```
aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
state-reason "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
state-reason "initializing" --state-value ALARM
```

다음 절차에서는 시스템 오류가 발생할 경우 알림을 받는 방법을 설명합니다.

1. Amazon SNS 주제 `arn:aws:sns:us-east-1:123456789012:notify-on-system-errors`을 생성합니다. 자세한 내용은 Amazon Simple Notification Service 설정을 참조하세요.
2. 경보를 만듭니다.

```
aws cloudwatch put-metric-alarm \
    --alarm-name SystemErrorsAlarm \
    --alarm-description "Alarm when system errors occur" \
    --namespace AWS/DynamoDB \
    --metric-name SystemErrors \
    --dimensions Name=TableName,Value=myTable
 Name=Operation,Value=aDynamoDBOperation \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --treat-missing-data breaching \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. 경보를 테스트합니다.

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value ALARM
```

# AWS CloudTrail을 사용하여 DynamoDB 작업 로깅

DynamoDB는 DynamoDB에서 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다. CloudTrail은 DynamoDB에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 DynamoDB 콘솔의 호출과 PartiQL 및 Classic API를 모두 사용한 DynamoDB API 작업에 대한 코드 호출이 포함됩니다. 추적을 생성하면 DynamoDB 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다. 트레일을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 DynamoDB에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

강력한 모니터링 및 경고를 위해 CloudTrail 이벤트를 Amazon CloudWatch Logs와 통합할 수도 있습니다. DynamoDB 서비스 활동에 대한 분석을 향상하고 AWS 계정의 활동 변경 사항을 식별하기 위해 Amazon Athena를 사용하여 AWS CloudTrail 로그를 쿼리할 수 있습니다. 예를 들어 쿼리를 사용하여 트렌드를 식별하고 소스 IP 주소나 사용자 등의 속성별로 활동을 추가로 격리할 수 있습니다.

구성 및 사용 방법을 포함하여 CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

주제

- CloudTrail의 DynamoDB 정보
- DynamoDB 로그 파일 항목 이해

## CloudTrail의 DynamoDB 정보

CloudTrail은 계정 생성 시 AWS 계정에서 활성화됩니다. 지원되는 이벤트 활동이 DynamoDB에서 발생하면, 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 설명은 CloudTrail 이벤트 기록 작업을 참조하세요.

DynamoDB에 대한 이벤트를 포함하여 AWS 계정의 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 지역에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 지역의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 추가적으로, CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에 대한 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기

## CloudTrail의 컨트롤 플레인 이벤트

다음 API 작업은 CloudTrail 파일의 이벤트로 기록됩니다.

Amazon DynamoDB

- CreateBackup
- CreateGlobalTable
- CreateTable
- DeleteBackup
- DeleteTable
- DescribeBackup
- DescribeContinuousBackups
- DescribeGlobalTable
- DescribeLimits
- DescribeTable
- DescribeTimeToLive
- ListBackups
- ListTables
- ListTagsOfResource
- ListGlobalTables
- RestoreTableFromBackup

- RestoreTableToPointInTime
- TagResource
- UntagResource
- UpdateGlobalTable
- UpdateTable
- UpdateTimeToLive
- DescribeReservedCapacity
- DescribeReservedCapacityOfferings
- PurchaseReservedCapacityOfferings
- DescribeScalableTargets
- RegisterScalableTarget

DynamoDB Streams

- DescribeStream
- ListStreams

DynamoDB Accelerator(DAX)

- CreateCluster
- CreateParameterGroup
- CreateSubnetGroup
- DecreaseReplicationFactor
- DeleteCluster
- DeleteParameterGroup
- DeleteSubnetGroup
- DescribeClusters
- DescribeDefaultParameters
- DescribeEvents
- DescribeParameterGroups
- DescribeParameters
- DescribeSubnetGroups

- IncreaseReplicationFactor
- ListTags
- RebootNode
- TagResource
- UntagResource
- UpdateCluster
- UpdateParameterGroup
- UpdateSubnetGroup

## CloudTrail의 DynamoDB 데이터 영역 이벤트

CloudTrail 파일에서 다음 API 작업의 로깅을 활성화하려면 CloudTrail에서 데이터 영역 API 활동의 로깅을 활성화해야 합니다. 자세한 내용은 추적을 위한 데이터 이벤트 로깅 단원을 참조하세요.

데이터 영역 이벤트를 리소스 유형별로 필터링하여 CloudTrail에서 선택적으로 기록하고 지불하려는 DynamoDB API 호출을 세부적으로 제어할 수 있습니다. 예를 들어 리소스 유형으로 `AWS::DynamoDB::Stream`을 지정하면 DynamoDB 스트림 API에 대한 호출만 로깅할 수 있습니다. 스트림이 활성화된 테이블의 경우 데이터 영역 이벤트의 리소스 필드에 `AWS::DynamoDB::Stream`과 `AWS::DynamoDB::Table`이 모두 포함됩니다. 리소스 유형으로 `AWS::DynamoDB::Table`을 지정하는 경우 기본적으로 DynamoDB 테이블과 DynamoDB 스트림 이벤트가 모두 로깅됩니다. 스트림 이벤트가 로깅되지 않도록 하려면 스트림 이벤트를 제외하는 필터를 추가합니다. 자세한 내용은 AWS CloudTrail API 참조의 DataResource를 참조하세요.

Amazon DynamoDB

- BatchExecuteStatement
- BatchGetItem
- BatchWriteItem
- DeleteItem
- ExecuteStatement
- ExecuteTransaction
- GetItem
- PutItem
- Query

- Scan
- TransactGetItems
- TransactWriteItems
- UpdateItem

> **Note**
>
> DynamoDB 유지 시간(TTL) 데이터 영역 작업은 CloudTrail에서 로깅하지 않습니다.

DynamoDB Streams

- GetRecords
- GetShardIterator

## DynamoDB 로그 파일 항목 이해

추적이란 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 대한 정보가 포함됩니다. 보안 인증 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 사용자 자격 증명으로 했는지 여부
- 역할 또는 페더레이션 사용자에 대한 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지.

> **Note**
>
> 키가 아닌 속성 값은 PartiQL API를 사용하는 작업의 CloudTrail 로그에서 삭제되며 Classic API를 사용하는 작업 로그에는 표시되지 않습니다.

자세한 내용은 CloudTrail userIdentity 요소를 참조하세요.

다음 예에서는 이러한 이벤트 유형의 CloudTrail 로그를 보여줍니다.

Amazon DynamoDB

- UpdateTable
- DeleteTable
- CreateCluster
- PutItem(성공)
- UpdateItem(실패)
- TransactWriteItems(성공)
- TransactWriteItems(TransactionCanceledException 있음)
- ExecuteStatement
- BatchExecuteStatement

DynamoDB Streams

- GetRecords

## UpdateTable


```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
```

```
                "arn": "arn:aws:iam::444455556666:role/admin-role",
                "accountId": "444455556666",
                "userName": "bob"
            }
        }
    },
    "eventTime": "2015-05-04T02:14:52Z",
    "eventSource": "dynamodb.amazonaws.com",
    "eventName": "UpdateTable",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "console.aws.amazon.com",
    "requestParameters": {
        "provisionedThroughput": {
            "writeCapacityUnits": 25,
            "readCapacityUnits": 25
        }
    },
    "responseElements": {
        "tableDescription": {
            "tableName": "Music",
            "attributeDefinitions": [
                {
                    "attributeType": "S",
                    "attributeName": "Artist"
                },
                {
                    "attributeType": "S",
                    "attributeName": "SongTitle"
                }
            ],
            "itemCount": 0,
            "provisionedThroughput": {
                "writeCapacityUnits": 10,
                "numberOfDecreasesToday": 0,
                "readCapacityUnits": 10,
                "lastIncreaseDateTime": "May 3, 2015 11:34:14 PM"
            },
            "creationDateTime": "May 3, 2015 11:34:14 PM",
            "keySchema": [
                {
                    "attributeName": "Artist",
                    "keyType": "HASH"
                },
```

```
                        {
                            "attributeName": "SongTitle",
                            "keyType": "RANGE"
                        }
                    ],
                    "tableStatus": "UPDATING",
                    "tableSizeBytes": 0
                }
            },
            "requestID": "AALNP0J2L244N5O15PKISJ1KUFVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "eb834e01-f168-435f-92c0-c36278378b6e",
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

## DeleteTable

```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
```

```
            },
            "eventTime": "2015-05-04T13:38:20Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "DeleteTable",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "console.aws.amazon.com",
            "requestParameters": {
                "tableName": "Music"
            },
            "responseElements": {
                "tableDescription": {
                    "tableName": "Music",
                    "itemCount": 0,
                    "provisionedThroughput": {
                        "writeCapacityUnits": 25,
                        "numberOfDecreasesToday": 0,
                        "readCapacityUnits": 25
                    },
                    "tableStatus": "DELETING",
                    "tableSizeBytes": 0
                }
            },
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

## CreateCluster

```
{
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
```

```
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "bob"
            },
            "eventTime": "2019-12-17T23:17:34Z",
            "eventSource": "dax.amazonaws.com",
            "eventName": "CreateCluster",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.16.304 Python/3.6.9
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.13.40",
            "requestParameters": {
                "sSESpecification": {
                    "enabled": true
                },
                "clusterName": "daxcluster",
                "nodeType": "dax.r4.large",
                "replicationFactor": 3,
                "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess"
            },
            "responseElements": {
                "cluster": {
                    "securityGroups": [
                        {
                            "securityGroupIdentifier": "sg-1af6e36e",
                            "status": "active"
                        }
                    ],
                    "parameterGroup": {
                        "nodeIdsToReboot": [],
                        "parameterGroupName": "default.dax1.0",
                        "parameterApplyStatus": "in-sync"
                    },
                    "clusterDiscoveryEndpoint": {
                        "port": 8111
                    },
                    "clusterArn": "arn:aws:dax:us-west-2:111122223333:cache/
daxcluster",
                    "status": "creating",
                    "subnetGroup": "default",
                    "sSEDescription": {
                        "status": "ENABLED",
                        "kMSMasterKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/764898e4-adb1-46d6-a762-e2f4225b4fc4"
```

```
                    },
                    "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess",
                    "clusterName": "daxcluster",
                    "activeNodes": 0,
                    "totalNodes": 3,
                    "preferredMaintenanceWindow": "thu:13:00-thu:14:00",
                    "nodeType": "dax.r4.large"
                }
            },
            "requestID": "585adc5f-ad05-4e27-8804-70ba1315f8fd",
            "eventID": "29158945-28da-4e32-88e1-56d1b90c1a0c",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

## **PutItem**(성공)

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
```

```
            },
            "eventTime": "2019-01-19T15:41:54Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "PutItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
 botocore/1.10.63",
            "requestParameters": {
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Scared of My Shadow"
                },
                "item": [
                    "Artist",
                    "SongTitle",
                    "AlbumTitle"
                ],
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

## **UpdateItem**(실패)


```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2020-09-03T22:27:15Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "UpdateItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
 botocore/1.10.63",
            "errorCode": "ConditionalCheckFailedException",
            "errorMessage": "The conditional request failed",
            "requestParameters": {
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Call Me Today"
                },
                "updateExpression": "SET #Y = :y, #AT = :t",
                "expressionAttributeNames": {
```

```
                    "#Y": "Year",
                    "#AT": "AlbumTitle"
                },
                "conditionExpression": "attribute_not_exists(#Y)",
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

## **TransactWriteItems**(성공)

```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
```

```
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2020-09-03T21:48:12Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-1",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
botocore/1.10.63",
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Today"
                        },
                        "items": [
                            "Artist",
                            "SongTitle",
                            "AlbumTitle"
                        ],
                        "conditionExpression": "#AT = :A",
                        "expressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Update",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Tomorrow"
                        },
                        "updateExpression": "SET #AT = :newval",
```

```
                    "ConditionExpression": "attribute_not_exists(Rating)",
                    "ExpressionAttributeNames": {
                        "#AT": "AlbumTitle"
                    },
                    "returnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "Delete",
                    "TableName": "Music",
                    "key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Yesterday"
                    },
                    "conditionExpression": "#P between :lo and :hi",
                    "expressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "ConditionCheck",
                    "TableName": "Music",
                    "Key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Now"
                    },
                    "ConditionExpression": "#P between :lo and :hi",
                    "ExpressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                }
            ],
            "returnConsumedCapacity": "TOTAL",
            "returnItemCollectionMetrics": "SIZE"
        },
        "responseElements": null,
        "requestID": "45EN32OM6TQSMV2MI65O4L5TNFVV4KQNSO5AEMVJF66Q9ASUAAJG",
        "eventID": "4f1cc78b-5c94-4174-a6ad-3ee78605381c",
        "readOnly": false,
        "resources": [
            {
                "accountId": "111122223333",
                "type": "AWS::DynamoDB::Table",
```

```
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

## `TransactWriteItems`(`TransactionCanceledException` 있음)

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2019-02-01T00:42:34Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
```

```
            "userAgent": "aws-cli/1.16.93 Python/3.4.7
Linux/4.9.119-0.1.ac.277.71.329.metal1.x86_64 botocore/1.12.83",
            "errorCode": "TransactionCanceledException",
            "errorMessage": "Transaction cancelled, please refer cancellation reasons
for specific reasons [ConditionalCheckFailed, None]",
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Today"
                        },
                        "items": [
                            "Artist",
                            "SongTitle",
                            "AlbumTitle"
                        ],
                        "conditionExpression": "#AT = :A",
                        "expressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Update",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Tomorrow"
                        },
                        "updateExpression": "SET #AT = :newval",
                        "ConditionExpression": "attribute_not_exists(Rating)",
                        "ExpressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Delete",
                        "TableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
```

```
                        "SongTitle": "Call Me Yesterday"
                    },
                    "conditionExpression": "#P between :lo and :hi",
                    "expressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "ConditionCheck",
                    "TableName": "Music",
                    "Key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Now"
                    },
                    "ConditionExpression": "#P between :lo and :hi",
                    "ExpressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                }
            ],
            "returnConsumedCapacity": "TOTAL",
            "returnItemCollectionMetrics": "SIZE"
        },
        "responseElements": null,
        "requestID": "A0GTQEKLBB9VD8E05REA5A3E1VVV4KQNSO5AEMVJF66Q9ASUAAJG",
        "eventID": "43e437b5-908a-46af-84e6-e27fffb9c5cd",
        "readOnly": false,
        "resources": [
            {
                "accountId": "111122223333",
                "type": "AWS::DynamoDB::Table",
                "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
            }
        ],
        "eventType": "AwsApiCall",
        "apiVersion": "2012-08-10",
        "managementEvent": false,
        "recipientAccountId": "111122223333",
        "eventCategory": "Data"
    }
]
```

```
}
```

## ExecuteStatement

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:06:45Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "ExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "statement": "SELECT * FROM Music WHERE Artist = 'No One You Know' AND
 SongTitle = 'Call Me Today' AND nonKeyAttr = ***(Redacted)"
            },
            "responseElements": null,
            "requestID": "V7G2KCSFLP83ORB7MMFG6RIAD3VV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "0b5c4779-e169-4227-a1de-6ed01dd18ac7",
            "readOnly": false,
```

```
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

## BatchExecuteStatement

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:24:48Z",
```

```
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "BatchExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "requestItems": [
                    {
                        "statement": "UPDATE Music SET Album = ***(Redacted) WHERE
 Artist = 'No One You Know' AND SongTitle = 'Call Me Today'"
                    },
                    {
                        "statement": "INSERT INTO Music VALUE {'Artist' :
 ***(Redacted), 'SongTitle' : ***(Redacted), 'Album' : ***(Redacted)}"
                    }
                ]
            },
            "responseElements": null,
            "requestID": "23PE7ED291UD65P9SMS6TISNVBVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "f863f966-b741-4c36-b15e-f867e829035a",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

## GetRecords

```
{
    "Records": [
        {
```

```
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-04-15T04:15:02Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "GetRecords",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.50 Python/3.6.13
 Linux/4.9.230-0.1.ac.224.84.332.metal1.x86_64 botocore/1.20.50",
            "requestParameters": {
                "shardIterator": "arn:aws:dynamodb:us-west-2:123456789012:table/
Music/stream/2021-04-15T04:02:47.428|1|AAAAAAAAAAH7HF3xwDQHBrvk2UBZ1PKh8bX3F
+JeH0rFwHCE7dz4VGV1ZoJ5bMxQwkmerA3wzCTL+zSseGLdSXNJP14EwrjLNvDNoZeRSJ/
n6xc3I4NYOptR4zR8d7VrjMAD6h5nR12NtxGIgJ/
dVsUpluWsHyCW3PPbKsMlJSruVRWoitRhSd3S6slEWEPB0bDC7+
+ISH5mXrCHOnvyezQKlqNshTSPZ5jWwqRj2VNSXCMTGXv9P01/
U0bpOUI2cuRTchgUpPSe3ur2sQrRj3KlbmIyCz7P
+H3CYlugafi8fQ5kipDSkESkIWS6O5ejzibWKg/3izms1eVIm/
zLFdEeihCYJ7G8fpHUSLX5JAk3ab68aUXGSFEZLONntgNIhQkcMo00/
mJlaIgkEdBUyqvZO1vtKUBH5YonIrZqSUhv8Coc+mh24vOg1YI+SPIXlr
+Lnl54BG6AjrmaScjHACVXoPDxPsXSJXC4c9HjoC3YSskCPV7uWi0f65/
n7JAT3cskcX2ISaLHwYzJPaMBSftxOgeRLm3BnisL32nT8uTj2gF/
PUrEjdyoqTX7EerQpcaekXmOgay5Kh8n4T2uPdM83f356vRpar/
DDp8pLFD0ddb6Yvz7zU2zGdAvTod3IScC1GpTqcjRxaMhlBVZy1TnI9Cs
+7fXMdUF6xYScjR2725icFBNLojSFVDmsfHabXaCEpmeuXZsLbp5CjcPAHa66R8mQ5tSoFjrzOEzeB4uconEXAMPLE=="
```

```
            },
            "responseElements": null,
            "requestID": "1M0U1Q80P4LDPT7A7N1A758N2VVV4KQNSO5AEMVJF66Q9EXAMPLE",
            "eventID": "09a634f2-da7d-4c9e-a259-54aceexample",
            "readOnly": true,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```


# DynamoDB에 대한 CloudWatch Contributor Insights를 사용해 데이터 액세스 분석

Amazon DynamoDB에 대한 Amazon CloudWatch Contributor Insights는 테이블 또는 인덱스에서 가장 자주 액세스하고 제한된 키를 한눈에 확인할 수 있는 진단 도구입니다. 이 도구는 CloudWatch Contributor Insights를 사용합니다.

테이블 또는 글로벌 보조 인덱스에서 DynamoDB에 대해 CloudWatch Contributor Insights를 활성화함으로써 이 리소스에서 가장 많이 액세스하고 제한된 항목을 확인할 수 있습니다.

> **Note**
>
> DynamoDB에 대한 Contributor Insights에는 CloudWatch 비용이 부과됩니다. 요금에 대한 자세한 정보는 Amazon CloudWatch 비용을 참조하세요.

주제

- DynamoDB에 대한 CloudWatch Contributor Insights: 작동 방식

- DynamoDB용 CloudWatch Contributor Insights 사용 시작
- DynamoDB용 CloudWatch Contributor Insights와 함께 IAM 사용

## DynamoDB에 대한 CloudWatch Contributor Insights: 작동 방식

Amazon DynamoDB는 테이블 또는 글로벌 보조 인덱스에서 가장 많이 액세스되고 제한된 항목에 대한 정보를 제공하기 위해 CloudWatch Contributor Insights와 통합됩니다. DynamoDB는 CloudWatch Contributor Insights 규칙, 보고서 및 보고서 데이터 그래프를 통해 정보를 제공합니다.

DynamoDB용 CloudWatch Contributor Insights는 DynamoDB 테이블에 성능 영향을 주지 않도록 설계되었습니다.

CloudWatch Contributor Insights에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 Contributor Insights를 사용하여 카디널리티가 높은 데이터 분석을 참조하세요.

다음 단원에서는 DynamoDB에 대한 CloudWatch Contributor Insights의 핵심 개념과 동작에 대해 설명합니다.

주제

- DynamoDB 규칙에 대한 CloudWatch Contributor Insights
- DynamoDB 그래프에 대한 CloudWatch Contributor Insights 이해
- 다른 DynamoDB 기능과 상호 작용
- DynamoDB 결제에 대한 CloudWatch Contributor Insights

### DynamoDB 규칙에 대한 CloudWatch Contributor Insights

테이블 또는 글로벌 보조 인덱스에서 DynamoDB에 대한 CloudWatch Contributor Insights를 활성화하면, DynamoDB는 다음과 같은 규칙을 대신해서 생성합니다.

- 가장 많이 액세스된 항목(파티션 키) - 테이블 또는 글로벌 보조 인덱스에서 가장 많이 액세스된 항목의 파티션 키를 의미합니다.

  CloudWatch 규칙 이름 형식: `DynamoDBContributorInsights-PKC-[resource_name]-[creationtimestamp]`
- 가장 많이 제한된 키(파티션 키) - 테이블 또는 글로벌 보조 인덱스에서 가장 많이 제한된 항목의 파티션 키를 의미합니다.

CloudWatch 규칙 이름 형식: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

> **Note**
>
> DynamoDB 테이블에서 Contributor Insights를 활성화해도 Contributor Insights 규칙 제한이 계속 적용됩니다. 자세한 내용은 CloudWatch Logs 서비스 할당량을 참조하세요.

테이블이나 글로벌 보조 인덱스에 정렬 키가 있는 경우 DynamoDB는 정렬 키에 대한 다음의 상세 규칙을 생성합니다.

- 가장 많이 액세스된 키(파티션 및 정렬 키) - 테이블 또는 글로벌 보조 인덱스에서 가장 많이 액세스된 항목의 파티션과 정렬 키를 의미합니다.

  CloudWatch 규칙 이름 형식: `DynamoDBContributorInsights-SKC-[resource_name]-[creationtimestamp]`
- 가장 많이 제한된 키(파티션 및 정렬 키) - 테이블 또는 글로벌 보조 인덱스에서 가장 많이 제한된 항목의 파티션 및 정렬 키를 의미합니다.

  CloudWatch 규칙 이름 형식: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

> **Note**
>
> - DynamoDB에 대한 CloudWatch Contributor Insights에 의해 생성된 규칙을 직접 수정하거나 삭제하기 위해 CloudWatch 콘솔 또는 API를 시용할 수 없습니다. 테이블 또는 글로벌 보조 인덱스에 있는 DynamoDB에 대한 CloudWatch Contributor Insights를 비활성화하는 것은 테이블 또는 글로벌 보조 인덱스에 생성된 규칙을 자동적으로 삭제합니다.
> - DynamoDB에서 생성한 CloudWatch Contributor Insights 규칙과 함께 GetInsightRuleReport 작업을 사용하는 경우 `MaxContributorValue` 및 `Maximum`만 유용한 통계를 반환합니다. 이 목록에 있는 다른 통계는 의미 있는 값을 반환하지 않습니다.
> - DynamoDB에 대한 CloudWatch Contributor Insights에서는 기여자가 25개로 제한됩니다. 25개가 넘는 기여자를 요청하면 오류가 반환됩니다.

DynamoDB 규칙에 대한 CloudWatch Contributor Insights를 사용하여 CloudWatch 경보를 생성할 수 있습니다. 이렇게 하면 항목이 `ConsumedThroughputUnits` 또는 `ThrottleCount`에 대한 특정 임계값을 초과하거나 충족하는 경우 알림을 받을 수 있습니다. 자세한 내용은 Contributor Insights 지표 데이터에 대한 경보 설정을 참조하세요.

## DynamoDB 그래프에 대한 CloudWatch Contributor Insights 이해

DynamoDB에 대한 CloudWatch Contributor Insights는 DynamoDB 및 CloudWatch 콘솔 모두에 두 가지 유형의 그래프, 즉 가장 많이 액세스된 항목 및 가장 많이 제한된 항목 그래프를 표시합니다.

가장 많이 액세스한 항목

이 그래프를 이용하여 테이블 또는 글로벌 보조 인덱스에서 가장 많이 액세스된 아이템을 확인할 수 있습니다. 그래프의 Y축은 `ConsumedThroughputUnits`를 나타내고 X축은 시간을 나타냅니다. 각 상위 N 키는 X축 아래에 있는 legend와 함께 특정 색상으로 표시됩니다.

DynamoDB는 결합된 읽기 및 쓰기 트래픽을 측정하는 `ConsumedThroughputUnits`를 사용하여 키 액세스 빈도를 측정합니다. `ConsumedThroughputUnits`는 다음과 같이 정의됩니다.

- 프로비저닝 방식 - (3 x 사용된 쓰기 용량 단위) + 사용된 읽기 용량 단위
- 온디맨드 방식 - (3 x 쓰기 요청 단위) + 읽기 요청 단위

DynamoDB 콘솔에서 그래프의 각 데이터 요소는 1분 주기 동안 최대 `ConsumedThroughputUnits`를 나타냅니다. 예를 들어 그래프 값이 180,000 `ConsumedThroughputUnits`이면 1분 주기(3,000 x 60초) 내에 60초의 시간 범위 동안 항목별 최대 처리량(1,000개의 쓰기 요청 유닛 또는 3,000개의 읽기 요청 유닛)으로 항목이 지속적으로 액세스되었음을 나타냅니다. 다시 말해 그래프 값은 각 1분 주기에서 가장 높은 트래픽을 보인 시간을 나타냅니다. CloudWatch 콘솔에서 `ConsumedThroughputUnits` 지표의 시간 세분성을 변경할 수 있습니다(예: 1분 대신 5분 지표를 보기 위해).

특이 사항이 없는데 여러 개의 군집 그래프 선이 보일 경우, 이는 해당 기간의 아이템에 대해 워크로드가 상대적으로 균형 잡혀 있음을 의미합니다. 그래프에서 연결된 선 외에 따로 떨어진 점이 있는 경우, 이는 짧은 기간 동안 자주 액세스된 아이템을 의미합니다.

테이블 또는 글로벌 보조 인덱스에 정렬 키가 있는 경우 DynamoDB는 2개의 그래프를 생성하며, 하나는 가장 많이 액세스된 파티션 키를 나타내고 다른 하나는 가장 많이 액세스된 파티션과 정렬 키 페어를 의미합니다. 파티션 키 전용 그래프에서 파티션 키 수준의 트래픽을 확인할 수 있습니다. 파티션 및 정렬 키 그래프에서 항목 수준의 트래픽을 확인할 수 있습니다.

### 가장 제한된 항목

이 그래프를 이용하여 테이블 또는 글로벌 보조 인덱스에서 가장 많이 제한된 아이템을 확인할 수 있습니다. 그래프의 Y축은 `ThrottleCount`를 나타내고 X축은 시간을 나타냅니다. 각 상위 N 키는 X축 아래에 있는 legend와 함께 특정 색상으로 표시됩니다.

DynamoDB는 `ThrottleCount`를 이용하여 제한 빈도를 측정하며, 이는 `ProvisionedThroughputExceededException`, `ThrottlingException`, `RequestLimitExceeded` 에러를 계산한 것입니다.

글로벌 보조 인덱스의 쓰기 용량이 부족하여 발생하는 쓰기 제한은 측정되지 않습니다. 글로벌 보조 인덱스의 가장 많이 액세스된 항목 그래프를 사용하여 쓰기 제한을 일으킬 수 있는 불균형한 액세스 패턴을 식별할 수 있습니다. 자세한 내용은 글로벌 보조 인덱스에서 프로비저닝된 처리량 고려 사항을 참조하세요.

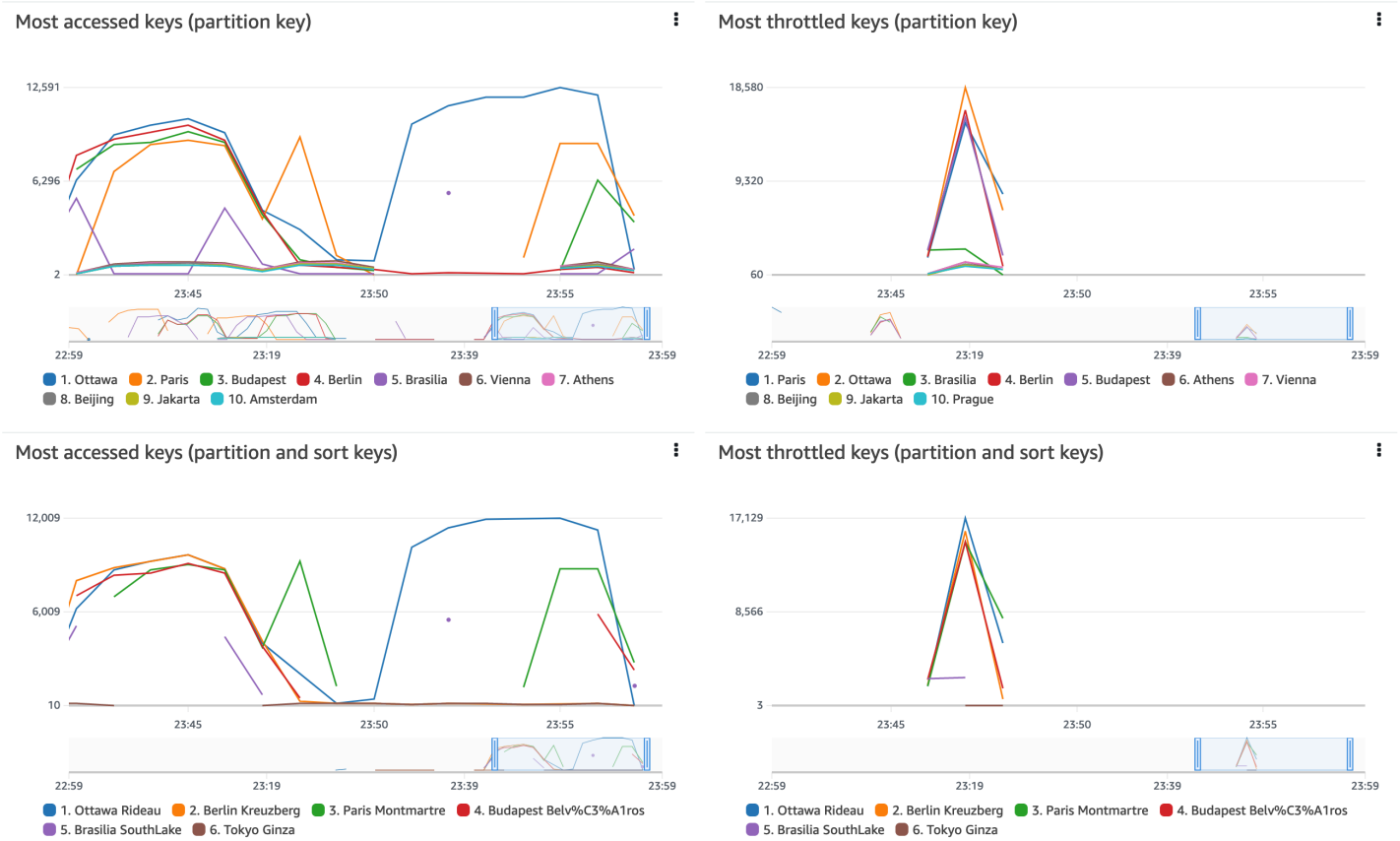
DynamoDB 콘솔에서 그래프의 각 데이터 요소는 1분 주기 동안의 스로틀 이벤트 수를 나타냅니다.

이 그래프에서 데이터가 나타나지 않는다면 요청이 제한되지 않고 있음을 의미합니다. 그래프에서 연결된 선 외에 따로 떨어진 점이 있는 경우, 이는 짧은 기간 동안 자주 스로틀된 항목을 의미합니다.

테이블 또는 글로벌 보조 인덱스에 정렬 키가 있는 경우 DynamoDB는 2개의 그래프를 생성하며, 하나는 가장 많이 제한된 파티션 키를 나타내고 다른 하나는 가장 많이 제한된 파티션과 정렬 키 페어를 의미합니다. 파티션 키만 있는 그래프에서 파티션 키 수준에 관한 제한 횟수를 확인할 수 있으며, 파티션과 정렬 키 조합 그래프에서 아이템 수준의 제한 횟수를 확인할 수 있습니다.

### 보고서 예

다음은 파티션 키와 정렬 키가 있는 테이블에서 생성된 보고서의 예제입니다.




## 다른 DynamoDB 기능과 상호 작용

다음 단원에서는 DynamoDB에 대한 CloudWatch Contributor Insights가 어떻게 동작하고 DynamoDB의 다른 여러 기능과 어떻게 상호 작용하는가에 대해 설명합니다.

### 전역 테이블

DynamoDB에 대한 CloudWatch Contributor Insights는 글로벌 테이블 복제본을 별개의 테이블로서 모니터링합니다. 한 AWS 리전의 복제본에 대한 Contributor Insights 그래프는 다른 리전과 동일한 패턴을 표시하지 않을 수 있습니다. 이것은 쓰기 데이터가 전역 테이블에 있는 모든 복제본에 복사되었기 때문입니다. 하지만 각 복제본은 지역 기반 읽기 트래픽만 수행할 수 있습니다.

### DynamoDB Accelerator(DAX)

DynamoDB에 대한 CloudWatch Contributor Insights는 DAX 캐시 응답을 표시하지 않습니다. 테이블 또는 글로벌 보조 인덱스의 액세스에 대한 응답만 표시합니다.

> **Note**
>
> DynamoDB CloudWatch Contributor Insights는 PartiQL 요청을 지원하지 않습니다.

저장 시 암호화

DynamoDB에 대한 CloudWatch Contributor Insights는 DynamoDB에서 암호화가 작동하는 방식에 영향을 미치지 않습니다. CloudWatch에 게시되는 기본 키 데이터는 AWS 소유 키로 암호화됩니다. 그러나 DynamoDB에서는 AWS 관리형 키 및 고객 관리형 키도 지원합니다.

DynamoDB에 대한 CloudWatch Contributor Insights는 자주 액세스하는 항목과 자주 스로틀링되는 항목의 파티션 키와 정렬 키(해당하는 경우)를 표시합니다. CloudWatch Contributor Insights는 암호화된 DynamoDB 테이블에서 작동하지만 테이블의 구성된 암호화와는 별개인 자체 Amazon 소유 암호화 컨텍스트를 사용한다는 점에 유의해야 합니다.

DynamoDB 테이블의 프라이머리 키에 민감한 정보가 포함되어 있고 조직의 보안 정책이 암호화 프로세스를 완전히 제어할 것을 요구하는 경우 CloudWatch Contributor Insights를 활성화하는 것이 적합하지 않을 수 있습니다.

세분화된 액세스 제어

DynamoDB에 대한 CloudWatch Contributor Insights는 세분화된 액세스 제어(FGAC)가 제공되는 테이블에서 다르게 작동하지 않습니다. 다시 말하면 적절한 CloudWatch 권한이 있는 사용자는 CloudWatch Contributor Insights 그래프에서 FGAC로 보호된 기본 키를 볼 수 있습니다.

테이블에 있는 기본 키에 CloudWatch에 발행하고 싶지 않은 FGAC로 보호된 데이터가 있는 경우, 해당 테이블에서 DynamoDB에 대한 CloudWatch Contributor Insights를 활성화하지 않아야 합니다.

액세스 제어

DynamoDB 제어 영역 권한 및 CloudWatch 데이터 영역 권한을 제한해 AWS Identity and Access Management(IAM)를 사용하는 DynamoDB에 대한 CloudWatch Contributor Insights에 대한 액세스를 제어합니다. 자세한 정보는 DynamoDB에 대한 CloudWatch Contributor Insights와 함께 IAM 사용하기를 참조하세요.

## DynamoDB 결제에 대한 CloudWatch Contributor Insights

DynamoDB에 대한 CloudWatch Contributor Insights 요금은 월별 청구서의 CloudWatch 섹션에 표시됩니다. 이러한 요금은 처리되는 DynamoDB 이벤트 수를 기준으로 계산됩니다. 활성화된 DynamoDB

에 대한 CloudWatch Contributor Insights가 있는 테이블 및 글로벌 보조 인덱스에서 데이터 영역 작업을 통해 쓰기 및 읽기가 된 각 항목은 하나의 이벤트를 의미합니다.

테이블 또는 글로벌 보조 인덱스에 정렬 키가 포함되어 있는 경우, 읽기 또는 쓰기가 수행된 각 항목이 두 개의 이벤트를 나타냅니다. 왜냐하면 DynamoDB가 별도의 시계열에서 최상위 기여자를 식별하기 때문입니다(하나는 파티션 키만을 위한 것이고 다른 하나는 파티션 및 정렬 키 페어를 위한 것).

예를 들어 애플리케이션에서 5개의 항목을 투입하는 `GetItem`, `PutItem` 및 `BatchWriteItem`이라는 DynamoDB 작업을 수행한다고 가정합니다.

- 테이블 또는 글로벌 보조 인덱스에 파티션 키만 있는 경우, 7개의 이벤트가 발생합니다(`GetItem`에 대해 1개, `PutItem`에 대해 1개, `BatchWriteItem`에 대해 5개).
- 테이블 또는 글로벌 보조 인덱스에 파티션 키와 정렬 키가 있는 경우에는 14개의 이벤트가 발생합니다(`GetItem`에 대해 2개, `PutItem`에 대해 2개, `BatchWriteItem`에 대해 10개).
- `Query` 작업에서는 반환된 항목의 수에 관계없이 항상 1개의 이벤트가 발생합니다.

다른 DynamoDB 기능과 달리 DynamoDB 결제에 대한 CloudWatch Contributor Insights는 다음에 따라 달라지지 않습니다.

- 용량 모드(프로비저닝 방식 대 온디맨드 방식)
- 읽기 또는 쓰기 요청 수행 여부
- 읽기 또는 쓰기가 수행된 항목의 크기 (KB)

## DynamoDB용 CloudWatch Contributor Insights 사용 시작

이 단원에서는 Amazon DynamoDB 콘솔 또는 AWS Command Line Interface(AWS CLI)와 함께 Amazon CloudWatch Contributor Insights를 이용하는 방법을 설명합니다.

다음 예에서는 DynamoDB 시작하기 자습서에서 정의한 DynamoDB 테이블을 사용합니다.

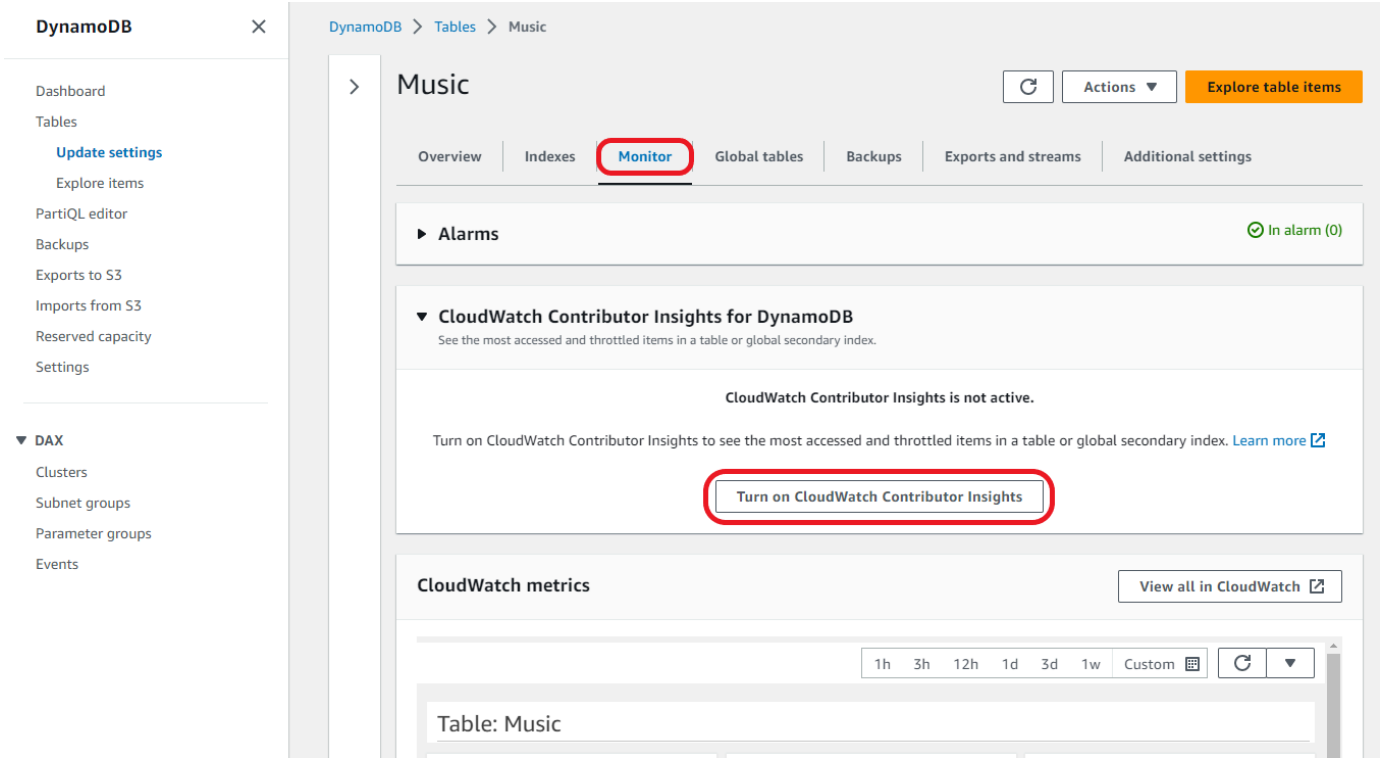
주제

- Contributor Insights 사용(콘솔)
- Contributor Insights(AWS CLI) 사용

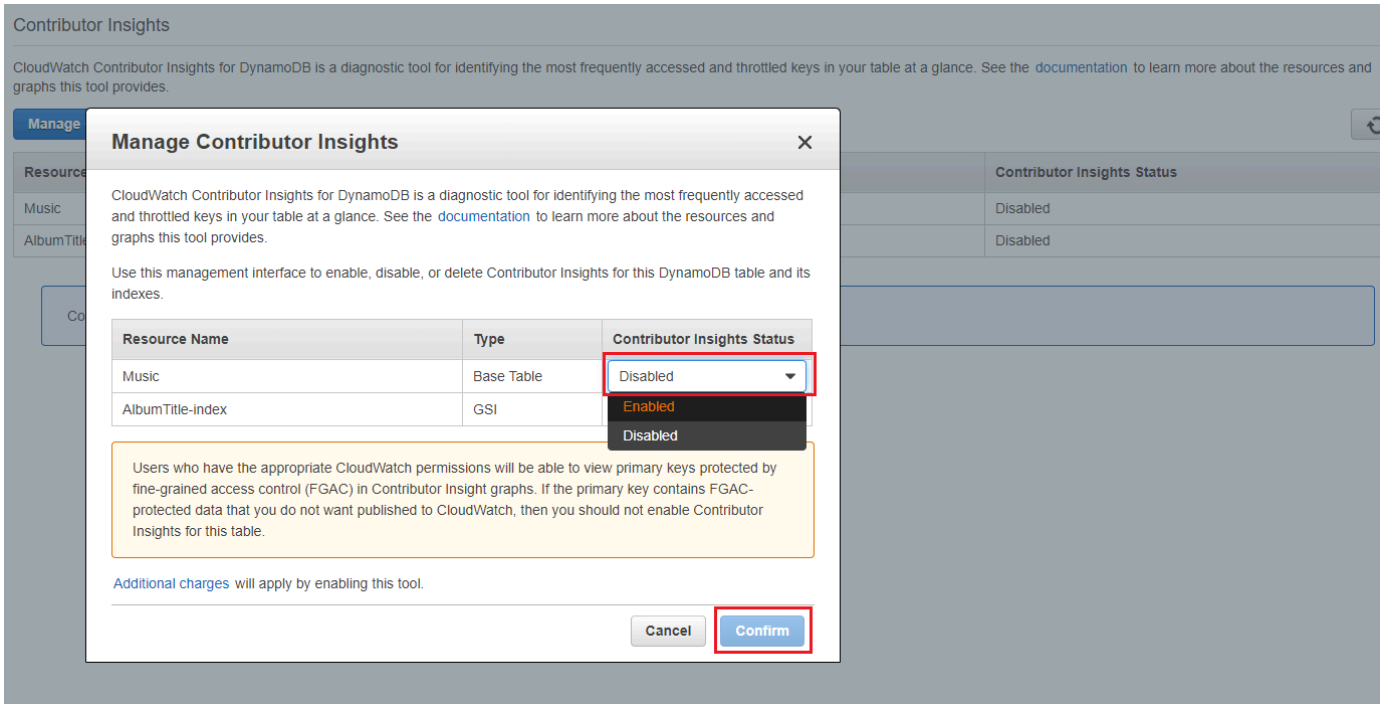
# Contributor Insights 사용(콘솔)

콘솔에서 Contributor Insights를 사용하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/vpc/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 테이블(Tables)을 선택합니다.
3. `Music` 테이블을 선택합니다.
4. 모니터링 탭을 선택합니다.
5. CloudWatch Contributor Insights 켜기를 선택합니다.

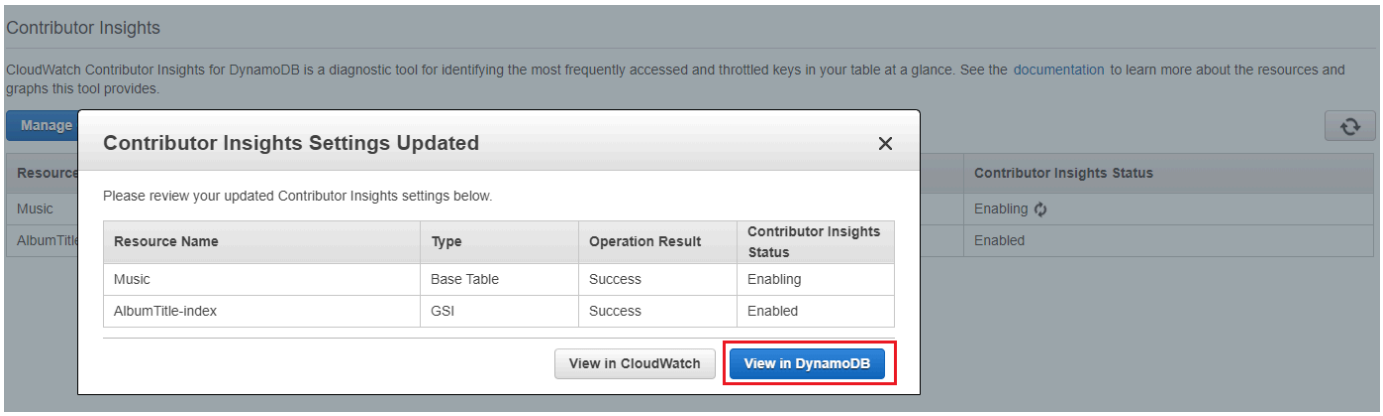



6. Contributor Insights 관리(Manage Contributor Insights) 대화 상자의 Contributor Insights 상태(Contributor Insights Status)에서 `Music` 기반 테이블 및 `AlbumTitle-index` 글로벌 보조 인덱스 모두에 대해 활성(Enabled)을 선택합니다. 그 다음 확인을 선택합니다.

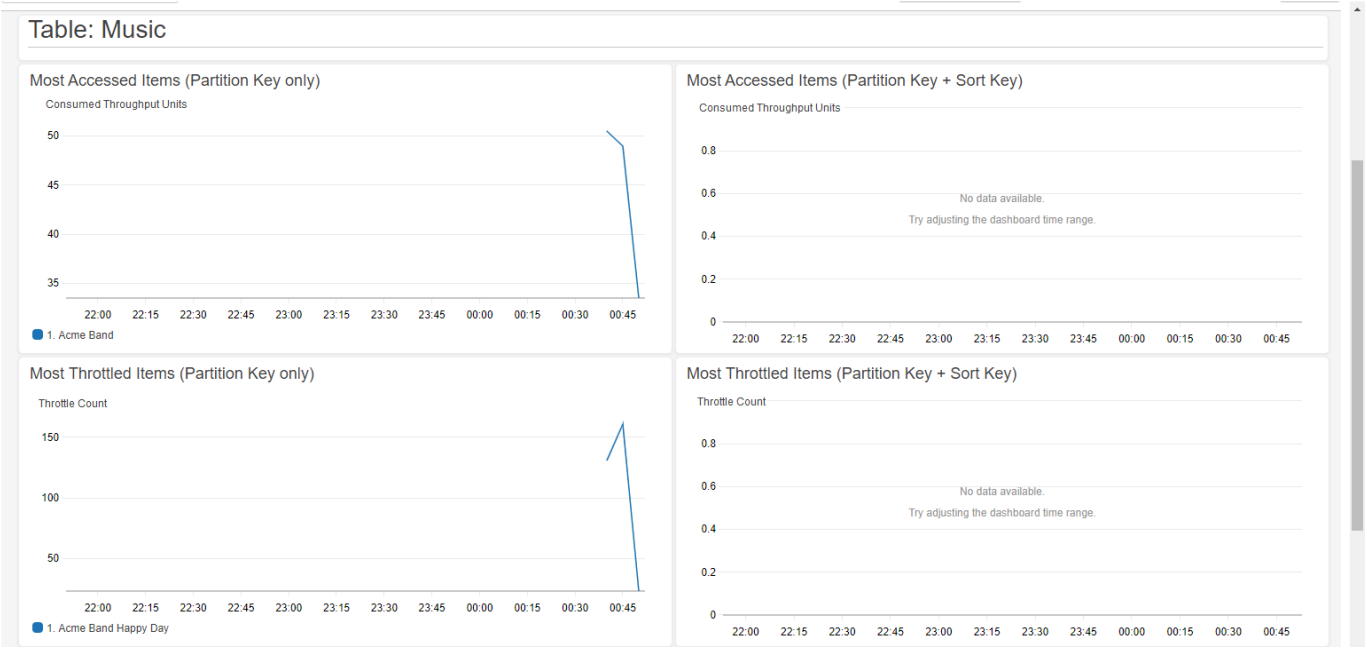



작업이 실패할 경우 Amazon DynamoDB API 참조의 DescribeContributorInsights FailureException에서 가능한 이유을 확인하세요.

7. DynamoDB에서 보기를 선택합니다.




8. 이제 Contributor Insights 그래프가 `Music` 테이블의 Contributor Insights 탭에 표시됩니다.

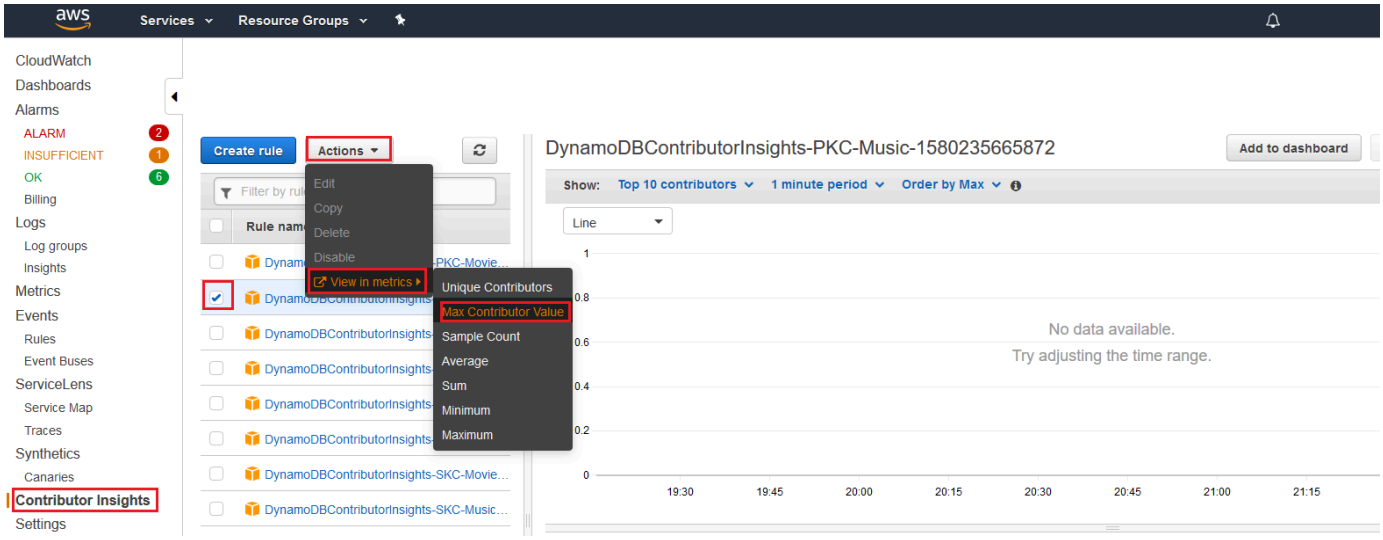



## CloudWatch 경보 생성

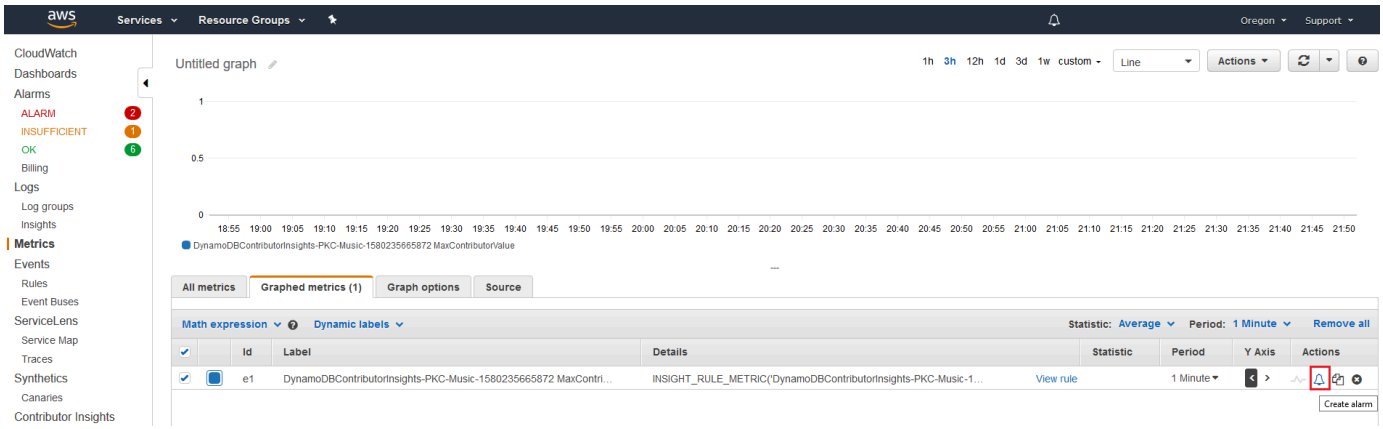
CloudWatch 경보를 생성하고 파티션 키가 50,000개 이상의 ConsumedThroughputUnits를 소비하는 경우 알림을 받으려면 다음 단계를 따르세요.

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 Contributor Insights를 선택합니다.
3. DynamoDBContributorInsights-PKC-Music 규칙을 선택합니다.
4. 작업 드롭다운 메뉴를 선택합니다.
5. 지표에서 보기를 선택합니다.
6. 최대 기고자 값을 선택합니다.

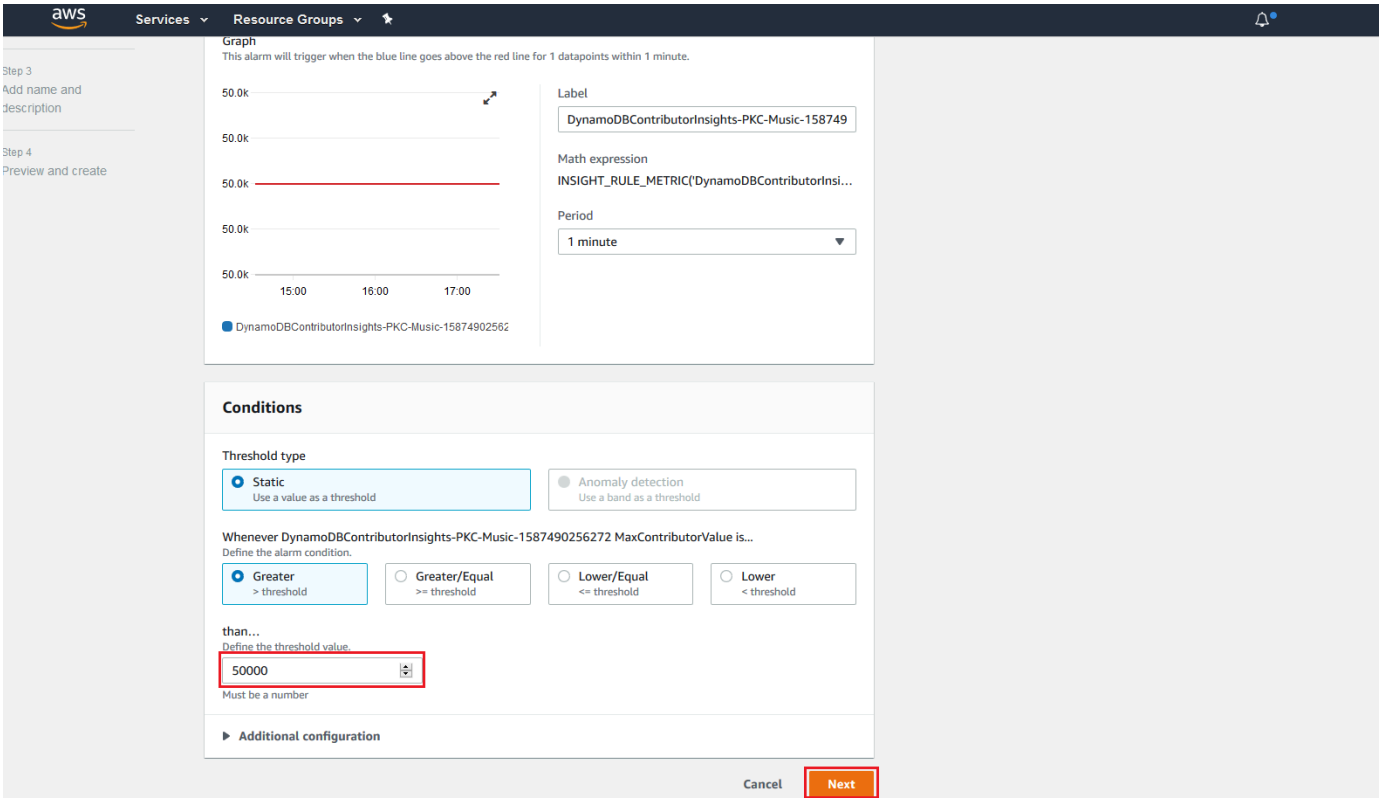
> **Note**
>
> `Max Contributor Value` 및 `Maximum`만 유용한 통계를 반환합니다. 이 목록에 있는 다른 통계는 의미 있는 값을 반환하지 않습니다.




7. 작업 열에서 경보 생성을 선택합니다.




8. 임계값에 50000을 입력하고 다음을 선택합니다.




9. 경보에 대한 알림을 구성하는 방법에 대한 자세한 내용은 Amazon CloudWatch 경보 사용을 참조하세요.

## Contributor Insights(AWS CLI) 사용

AWS CLI에서 Contributor Insights를 사용하려면

1. `Music` 베이스 테이블에 있는 DynamoDB에 대한 CloudWatch Contributor Insights를 활성화합니다.

```
aws dynamodb update-contributor-insights --table-name Music --contributor-insights-action=ENABLE
```

2. `AlbumTitle-index` 글로벌 보조 인덱스에 있는 DynamoDB에 대한 Contributor Insights를 활성화합니다.

```
aws dynamodb update-contributor-insights --table-name Music --index-name
 AlbumTitle-index --contributor-insights-action=ENABLE
```

3. `Music` 테이블과 모든 인덱스에 대한 상태와 규칙을 확인합니다.

```
aws dynamodb describe-contributor-insights --table-name Music
```

4. `AlbumTitle-index` 글로벌 보조 인덱스에 있는 DynamoDB에 대한 CloudWatch Contributor Insights를 비활성화합니다.

```
aws dynamodb update-contributor-insights --table-name Music --index-name
 AlbumTitle-index --contributor-insights-action=DISABLE
```

5. `Music` 테이블과 모든 인덱스에 대한 상태를 확인합니다.

```
aws dynamodb list-contributor-insights --table-name Music
```

## DynamoDB용 CloudWatch Contributor Insights와 함께 IAM 사용

Amazon DynamoDB에 대한 Amazon CloudWatch Contributor Insights를 처음 활성화하면, DynamoDB는 자동적으로 고객님을 위한 AWS Identity and Access Management(IAM) 서비스 링크 역할을 생성합니다. 이 역할, `AWSServiceRoleForDynamoDBCloudWatchContributorInsights`는 DynamoDB가 고객님을 대신해 CloudWatch Contributor Insights 규칙을 관리할 수 있도록 합니다. 이 서비스 연결 역할을 삭제하지 마세요. 이를 삭제하면 모든 관리 규칙은 테이블 또는 글로벌 보조 인덱스를 삭제할 때 더 이상 정리되지 않습니다.

서비스 연결 역할에 대한 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 사용을 참조하세요

다음 권한이 필요합니다.

- DynamoDB에 대한 CloudWatch Contributor Insights를 활성화하거나 비활성화하려면, 테이블 또는 인덱스에 대한 `dynamodb:UpdateContributorInsights` 권한이 반드시 필요합니다.
- DynamoDB 그래프에 대한 CloudWatch Contributor Insights를 보려면, `cloudwatch:GetInsightRuleReport` 권한이 반드시 필요합니다.
- 주어진 DynamoDB 테이블 또는 인덱스에 대한 DynamoDB용 CloudWatch Contributor Insights를 설명하려면 `dynamodb:DescribeContributorInsights` 권한이 있어야 합니다.
- 각 테이블 및 글로벌 보조 인덱스에 대한 DynamoDB CloudWatch Contributor Insights 상태를 작성하려면, `dynamodb:ListContributorInsights` 권한이 반드시 필요합니다.

## 예: DynamoDB용 CloudWatch Contributor Insights 활성화 또는 비활성화

다음 IAM 정책은 DynamoDB용 CloudWatch Contributor Insights를 활성화 또는 비활성화할 권한을 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        }
    ]
}
```

KMS 키로 암호화된 테이블의 경우 사용자에게 Contributor Insights를 업데이트하려면 kms:Decrypt 권한이 있어야 합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
```

```
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        },
        {
            "Effect": "Allow",
            "Resource": "arn:aws:kms:*:*:key/*",
            "Action": [
                "kms:Decrypt"
            ]
        }
    ]
}
```

## 예: CloudWatch Contributor Insights 규칙 보고서 검색

다음 IAM 정책은 CloudWatch Contributor Insights 규칙 보고서를 검색할 수 있는 권한을 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport"
            ],
```

```
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/
DynamoDBContributorInsights*"
        }
    ]
}
```

## 예: 리소스를 기반으로 DynamoDB용 CloudWatch Contributor Insights 권한을 선택적으로 적용

다음 IAM 정책은 특정 글로벌 보조 인덱스에 대해 `ListContributorInsights` 및 `DescribeContributorInsights` 작업에 대한 권한을 부여하고, `UpdateContributorInsights` 작업을 거부합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListContributorInsights",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books/
index/Author-index"
        }
    ]
}
```

# DynamoDB용 CloudWatch Contributor Insights의 서비스 연결 역할 사용

DynamoDB용 CloudWatch Contributor Insights는 AWS Identity and Access Management(IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 DynamoDB용 CloudWatch Contributor Insights에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 DynamoDB용 CloudWatch Contributor Insights에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 DynamoDB용 CloudWatch Contributor Insights를 더 쉽게 설정할 수 있습니다. DynamoDB용 CloudWatch Contributor Insights는 서비스 연결 역할의 권한을 정의하며, 달리 정의하지 않은 한 DynamoDB용 CloudWatch Contributor Insights만 해당 역할을 가질 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 IAM으로 작업하는 AWS 서비스를 참조해 서비스 연결 역할(Service-Linked Role) 열이 예(Yes)인 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 링크가 있는 예를 선택합니다.

DynamoDB용 CloudWatch Contributor Insights의 서비스 연결 역할 권한

DynamoDB용 CloudWatch Contributor Insights는 AWSServiceRoleForDynamoDBCloudWatchContributorInsights라는 서비스 연결 역할을 사용합니다. 서비스 연결 역할의 목적은 Amazon DynamoDB가 사용자 대신 DynamoDB 테이블 및 글로벌 보조 인덱스에 대해 생성된 Amazon CloudWatch Contributor Insights 규칙을 관리할 수 있도록 하는 것입니다.

`AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `contributorinsights.dynamodb.amazonaws.com`

역할 권한 정책은 DynamoDB용 CloudWatch Contributor Insights가 지정된 리소스에서 다음 작업을 완료하도록 허용합니다.

- 작업: `DynamoDBContributorInsights`에 대한 `Create and manage Insight Rules`

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한 섹션을 참조하세요.

DynamoDB용 CloudWatch Contributor Insights의 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 Contributor Insights를 활성화하면 DynamoDB용 CloudWatch Contributor Insights는 서비스 연결 역할을 생성합니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. Contributor Insights를 활성화하면 DynamoDB용 CloudWatch Contributor Insights는 서비스 연결 역할을 다시 생성합니다.

DynamoDB용 CloudWatch Contributor Insights의 서비스 연결 역할 편집

DynamoDB용 CloudWatch Contributor Insights는 `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 서비스 연결 역할 편집을 허용하지 않습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

DynamoDB용 CloudWatch Contributor Insights의 서비스 연결 역할 삭제

`AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 역할은 수동으로 삭제할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 Contributor Insights를 비활성화하면 DynamoDB용 CloudWatch Contributor Insights는 리소스를 정리합니다.

또한 IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 서비스 연결 역할을 수동으로 삭제할 수 있습니다. 단, 서비스 연결 역할에 대한 리소스를 먼저 정리해야 수동으로 삭제할 수 있습니다.

> **Note**
>
> 리소스를 삭제하려 할 때 DynamoDB용 CloudWatch Contributor Insights 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

IAM을 사용하여 수동으로 서비스 연결 역할을 삭제하려면 다음을 수행하세요.

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하십시오.

# DynamoDB를 사용한 설계 및 아키텍처 설계 모범 사례

이 섹션을 통해 DynamoDB를 이용할 때 성능을 극대화하고 처리량 비용을 최소화할 수 있는 방법을 신속히 찾으세요.

주제

- DynamoDB를 위한 NoSQL 설계
- DynamoDB Well-Architected 렌즈를 사용하여 DynamoDB 워크로드 최적화
- DynamoDB에서 효과적으로 파티션 키를 설계해 사용하는 모범 사례입니다.
- 정렬 키를 사용하여 DynamoDB의 데이터를 정리하는 모범 사례
- DynamoDB의 보조 인덱스 사용에 대한 모범 사례
- DynamoDB에 큰 항목과 속성 저장 모범 사례
- DynamoDB의 시계열 데이터 처리 모범 사례
- DynamoDB 테이블의 다대다 관계 관리 모범 사례
- DynamoDB의 데이터 쿼리 및 스캔 모범 사례
- DynamoDB 테이블 설계 모범 사례
- DynamoDB 글로벌 테이블 설계 모범 사례
- DynamoDB의 컨트롤 플레인 관리 모범 사례
- DynamoDB에서 대량 데이터 작업을 사용하는 모범 사례
- DynamoDB에서 버전 관리를 구현하는 모범 사례
- DynamoDB의 AWS 결제 및 사용량 보고서 이해에 관한 모범 사례
- DynamoDB 테이블을 한 계정에서 다른 계정으로 마이그레이션
- DAX를 DynamoDB 애플리케이션과 통합하기 위한 권장 가이드

- AWS PrivateLink for Amazon DynamoDB 사용 시 고려 사항

## DynamoDB를 위한 NoSQL 설계

Amazon DynamoDB와 같은 NoSQL 데이터베이스 시스템은 데이터 관리에 키-값 페어 또는 문서 스토리지와 같은 대체 모델을 사용합니다. 관계형 데이터베이스 관리 시스템을 DynamoDB와 같은 NoSQL

데이터베이스 시스템으로 교체할 때 주요 차이점과 설계에 있어 특정 접근법을 이해하는 것이 중요합니다.

주제


- 관계형 데이터 설계와 NoSQL의 차이점
- NoSQL 설계의 두 가지 주요 개념
- NoSQL 설계에 접근
- DynamoDB용 NoSQL Workbench


## 관계형 데이터 설계와 NoSQL의 차이점

관계형 데이터베이스 시스템(RDBMS)과 NoSQL 데이터베이스는 각기 다른 장단점을 갖고 있습니다.

- RDBMS에서는 데이터를 유연하게 쿼리할 수 있지만, 쿼리 비용이 상대적으로 높으며 트래픽이 많은 상황에서는 확장성이 떨어집니다(DynamoDB의 관계형 데이터를 모델링하는 첫 번째 단계 참조).
- DynamoDB와 같은 NoSQL 데이터베이스에서는 몇 가지 방법으로 데이터를 효율적으로 쿼리할 수 있지만, 그 외에는 쿼리 비용이 높고 속도가 느립니다.

이런 차이가 두 시스템의 데이터베이스 설계를 다르게 만듭니다.

- RDBMS의 경우, 세부적인 구현이나 성능을 걱정하지 않고 유연성을 목적으로 설계합니다. 일반적으로 쿼리 최적화가 스키마 설계에 영향을 미치지 않지만, 정규화가 중요합니다.
- DynamoDB의 경우, 가장 중요하고 범용적인 쿼리를 가능한 빠르고 저렴하게 수행할 수 있도록 스키마를 설계합니다. 사용자의 데이터 구조는 사용자 비즈니스 사용 사례의 특정 요구 사항에 적합하도록 만듭니다.

## NoSQL 설계의 두 가지 주요 개념

NoSQL 설계에는 RDBMS 설계와 다른 사고 방식이 요구됩니다. RDBMS의 경우, 액세스 패턴을 생각하지 않고 정규화된 데이터 모델을 생성할 수 있습니다. 그런 후 나중에 새로운 질문과 쿼리에 대한 요구 사항이 생길 때 이를 확장할 수 있습니다. 각 데이터 유형을 고유의 테이블에 구성할 수 있습니다.

NoSQL 설계의 차이점

- 대조적으로 대답해야 할 질문을 알기 전까지는 DynamoDB에 대한 스키마 설계를 시작할 수 없습니다. 사전에 비즈니스 문제와 애플리케이션 사용 사례를 이해해야 합니다.

- DynamoDB 애플리케이션에서는 가능한 적은 수의 테이블을 유지해야 합니다. 테이블 수가 적을수록 확장성이 향상되고 권한 관리가 줄어들며 DynamoDB 애플리케이션의 오버헤드가 감소합니다. 백업 비용도 전반적으로 낮게 유지할 수 있습니다.

## NoSQL 설계에 접근

DynamoDB 애플리케이션 설계의 첫 번째 단계는 시스템이 충족해야 하는 특정 쿼리 패턴을 파악하는 것입니다.

특히 시작을 하기 앞서 애플리케이션 액세스 패턴에서 3가지 기본적인 속성을 이해하는 것이 중요합니다.

- 데이터 크기: 저장해야 할 데이터의 양과 한 번에 요청할 데이터의 양을 알면 가장 효과적으로 데이터를 파티션(분할)하는 방법을 결정할 수 있습니다.
- 데이터 모양: 쿼리를 처리할 때 데이터를 변화시키는 대신(RDBMS 시스템의 방식), NoSQL 데이터베이스는 데이터베이스의 모양이 쿼리 대상과 일치하도록 데이터를 구성합니다. 이는 속도와 확장성 향상에 중요한 요소입니다.
- 데이터 속도: DynamoDB는 프로세스 쿼리에 사용할 수 있는 물리적 파티션의 수를 늘리고, 해당 파티션에 효율적으로 데이터를 배포해 조정합니다. 사전에 피크 쿼리 로드를 알면 I/O 용량을 가장 효과적으로 사용할 수 있는 데이터 파티션(분할) 방법을 결정하는 데 도움이 될 수 있습니다.

특정 쿼리 요구 사항을 파악한 후, 성능을 결정하는 일반 원칙에 따라 데이터를 구성할 수 있습니다.

- 관련 데이터를 함께 유지합니다.  연구에 따르면 관련 데이터를 한 곳에 유지하는 '참조 지역성' 원칙은 수년 전 라우팅 테이블을 최적화하는 데 중요한 것으로 밝혀진 것처럼 NoSQL 시스템의 성능과 응답 시간을 개선하는 핵심 요소입니다.

  대체로 DynamoDB 애플리케이션에서는 가능한 적은 수의 테이블을 유지해야 합니다.

  단, 볼륨이 많은 시계열 데이터가 관여된 경우나 액세스 패턴이 아주 다른 데이터 세트는 해당되지 않습니다. 통상 반전된 인덱스의 단일 테이블로 간단한 쿼리를 활성화시켜 사용자의 애플리케이션에 필요한 복잡한 계층적 데이터 구조를 생성 및 검색할 수 있습니다.
- 정렬 순서를 사용합니다.  핵심 설계가 함께 정렬할 것을 요구하는 경우, 관련 항목을 그룹으로 묶어 효율적으로 쿼리할 수 있습니다. 이는 중요한 NoSQL 설계 전략입니다.
- 쿼리를 분산합니다.  많은 볼륨의 쿼리를 데이터베이스의 특정 부분에 집중시키지 않는 것이 중요합니다. I/O 용량을 초과할 수 있기 때문입니다. 대신 트래픽을 가능한 한 여러 파티션으로 분산시켜 핫 스팟이 방지되도록 데이터 키를 설계해야 합니다.

- 글로벌 보조 인덱스를 사용합니다. 특정 글로벌 보조 인덱스를 생성해 기본 테이블이 지원할 수 있는 것보다 다양한 쿼리를 더 빨리 저렴한 비용으로 활성화시킬 수 있습니다.

이런 일반 원칙은 DynamoDB에서 데이터를 효율적으로 모델링하기 위해 사용할 수 있는 범용 설계 패턴 가운데 일부로 전환됩니다.

## DynamoDB용 NoSQL Workbench

DynamoDB용 NoSQL Workbench는 최신 데이터베이스 개발 및 작업에 사용할 수 있는 크로스 플랫폼, 클라이언트 측 GUI 애플리케이션입니다. Windows, macOS 및 Linux에서 사용할 수 있습니다. NoSQL Workbench는 DynamoDB 테이블 설계, 생성, 쿼리 및 관리에 도움이 되는 데이터 모델링, 데이터 시각화, 샘플 데이터 생성 및 쿼리 개발 기능을 제공하는 시각적 개발 도구입니다. DynamoDB용 NoSQL Workbench를 사용하여 새로운 데이터 모델을 구축하거나 애플리케이션 데이터 액세스 패턴을 충족하는 기존 데이터 모델을 기반으로 모델을 설계할 수 있습니다. 그리고 프로세스를 종료할 때 설계된 데이터 모델을 가져오고 내보낼 수도 있습니다. 자세한 내용은 NoSQL Workbench로 데이터 모델 빌드 섹션을 참조하세요.

# DynamoDB Well-Architected 렌즈를 사용하여 DynamoDB 워크로드 최적화

이 섹션에서는 효율적으로 구조화된 DynamoDB 워크로드를 설계하기 위한 설계 원칙 및 지침 모음인 Amazon DynamoDB Well-Architected 렌즈에 대해 설명합니다.

## DynamoDB 테이블의 비용 최적화

이 섹션에서는 기존 DynamoDB 테이블의 비용을 최적화하는 방법에 대한 모범 사례를 다룹니다. 다음 전략을 검토하여 요구 사항에 가장 적합한 비용 최적화 전략을 확인하고 반복적으로 접근해야 합니다. 각 전략은 비용에 영향을 미칠 수 있는 요소에 대한 개요, 찾아야 할 징후, 비용 절감 방법에 대한 규범적 지침을 제공합니다.

주제

- 테이블 수준에서 비용 평가
- 테이블 용량 모드 평가
- DynamoDB 테이블의 오토 스케일링 설정 평가
- DynamoDB 테이블 클래스 선택 평가
- DynamoDB의 미사용 리소스 식별

- DynamoDB 테이블 사용 패턴 평가
- DynamoDB 스트림 사용량 평가
- DynamoDB 테이블에서 적절한 규모의 프로비저닝을 위해 프로비저닝된 용량 평가

## 테이블 수준에서 비용 평가

AWS Management Console 내에 있는 Cost Explorer 도구를 사용하면 읽기, 쓰기, 스토리지 및 백업 요금과 같은 비용을 유형별로 분류하여 볼 수 있습니다. 월별 또는 일과 같은 기간별로 요약된 비용도 확인할 수 있습니다.

관리자가 직면할 수 있는 한 가지 문제는 특정 테이블 하나의 비용만 검토해야 하는 경우입니다. 이 데이터 중 일부는 DynamoDB 콘솔이나 `DescribeTable` API 호출을 통해 사용할 수 있지만 Cost Explorer 탐색기에서는 기본적으로 특정 테이블과 관련된 비용을 기준으로 필터링하거나 그룹화할 수 없습니다. 이 섹션에서는 태그를 사용하여 Cost Explorer에서 개별 테이블 비용을 분석하는 방법을 보여줍니다.

주제

- 단일 DynamoDB 테이블의 비용을 확인하는 방법
- Cost Explorer의 기본 보기
- Cost Explorer에서 테이블 태그를 사용하고 적용하는 방법

### 단일 DynamoDB 테이블의 비용을 확인하는 방법

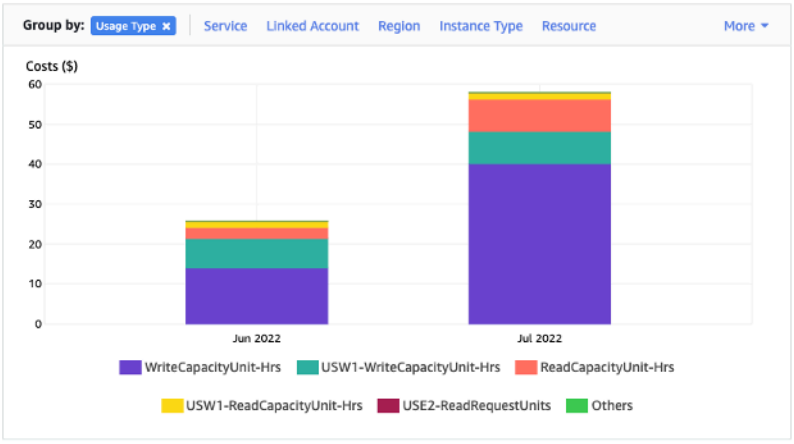
Amazon DynamoDB AWS Management Console 및 `DescribeTable` API 모두 프라이머리 키 스키마, 테이블의 인덱스, 테이블과 인덱스의 크기 및 항목 수 등 단일 테이블에 대한 정보를 표시합니다. 테이블 크기와 인덱스 크기를 사용하여 테이블의 월별 스토리지 비용을 계산할 수 있습니다. 예를 들어, us-east-1 리전에서는 GB당 0.25 USD입니다.

테이블이 프로비저닝된 용량 모드인 경우 현재 RCU 및 WCU 설정도 반환됩니다. 이를 사용하여 테이블의 현재 읽기 및 쓰기 비용을 계산할 수 있지만, 특히 테이블이 Auto Scaling으로 구성된 경우 이러한 비용이 달라질 수 있습니다.

> **Note**
>
> 테이블이 온디맨드 용량 모드인 경우 한 기간의 프로비저닝되지 않은 실제 사용량을 기준으로 청구되므로 `DescribeTable`은 처리량(throughput) 비용을 추정하는 데 도움이 되지 않습니다.

### Cost Explorer의 기본 보기

Cost Explorer의 기본 보기는 처리량(throughput) 및 스토리지와 같은 사용된 리소스 비용을 보여주는 차트를 제공합니다. 월별 또는 일별 합계와 같이 기간별로 비용을 그룹화하도록 선택할 수 있습니다. 스토리지, 읽기, 쓰기 및 기타 기능의 비용도 세분화하여 비교할 수 있습니다.




### Cost Explorer에서 테이블 태그를 사용하고 적용하는 방법

기본적으로 Cost Explorer는 여러 테이블의 비용을 합산하므로 특정 테이블에 대한 비용 요약을 제공하지 않습니다. 하지만 AWS 리소스 태깅을 사용하여 메타데이터 태그로 각 테이블을 식별할 수 있습니다. 태그는 프로젝트 또는 부서에 속한 모든 리소스를 식별하는 등 다양한 용도로 사용할 수 있는 키-값 쌍입니다. 이 예제에서는 MyTable이라는 테이블이 있다고 가정합니다.

1. table_name의 키와 MyTable의 값으로 태그를 설정합니다.
2. Cost Explorer 탐색기에서 태그를 활성화한 후 태그 값을 필터링하여 각 테이블의 비용을 더 잘 파악할 수 있습니다.

> **Note**
>
> Cost Explorer에 태그가 표시되려면 하루나 이틀이 걸릴 수 있습니다.

콘솔에서 직접 또는AWS CLI 또는AWS SDK와 같은 자동화를 통해 메타데이터 태그를 설정할 수 있습니다. 조직의 새 테이블 생성 프로세스의 일부로 table_name 태그를 설정하도록 요구하는 것이 좋습니다. 기존 테이블의 경우 Python 유틸리티를 사용하여 이러한 태그를 찾아 계정의 특정 리전에 있는 모든 기존 테이블에 적용할 수 있습니다. 자세한 내용은 GitHub의 Eponymous Table Tagger를 참조하세요.

# 테이블 용량 모드 평가

이 섹션에서는 DynamoDB 테이블에 적절한 용량 모드를 선택하는 방법을 간략히 살펴봅니다. 각 모드는 처리량(throughput) 변화에 대한 대응력 및 사용량 청구 방식 측면에서 다양한 워크로드의 요구 사항을 충족하도록 조정됩니다. 결정을 내릴 때 이러한 요소들의 균형을 맞춰야 합니다.

주제


- 사용할 수 있는 테이블 용량 모드
- 온디맨드 용량 모드를 선택하는 경우
- 프로비저닝된 용량 모드를 선택하는 경우
- 테이블 용량 모드를 선택할 때 고려해야 할 추가 요소


## 사용할 수 있는 테이블 용량 모드

DynamoDB 테이블을 생성할 때 온디맨드 또는 프로비저닝된 용량 모드를 선택해야 합니다. 24시간마다 한 번 용량 모드를 전환할 수 있습니다. 유일한 예외는 프로비저닝된 모드 테이블을 온디맨드 모드로 전환하는 경우 동일한 24시간 내에 프로비저닝된 모드로 다시 전환할 수 있다는 것입니다.

### 온디맨드 용량 모드

온디맨드 용량 모드는 DynamoDB 테이블의 용량을 계획하거나 프로비저닝할 필요가 없도록 설계되었습니다. 이 모드에서 테이블은 리소스를 확장하거나 축소할 필요 없이 테이블에 대한 요청을 즉시 수용합니다(테이블의 이전 최대 처리량(throughput)의 최대 2배).

DynamoDB 온디맨드는 읽기 및 쓰기 요청에 대해 요청당 지불 요금이 적용되므로 사용하는 만큼에 대해서만 비용을 지불하면 됩니다.

### 프로비저닝된 용량 모드

프로비저닝된 용량 모드는 테이블이 요청에 사용할 수 있는 용량을 직접 또는 오토 스케일링의 도움으로 정의할 수 있는 보다 전통적인 모델입니다. 지정된 시간에 테이블에 대해 특정 용량이 프로비저닝되기 때문에 비용은 사용된 요청 수가 아닌 프로비저닝된 용량을 기준으로 청구됩니다. 할당된 용량을 초과하면 테이블이 요청을 거부하고 애플리케이션 사용자의 경험을 감소시킬 수도 있습니다.

프로비저닝된 용량 모드에서는 스로틀링을 낮게 유지하고 비용을 조정하기 위해 테이블을 과다 프로비저닝 또는 과소 프로비저닝하지 않도록 지속적인 모니터링을 통해 균형을 찾아야 합니다.

온디맨드 용량 모드를 선택하는 경우

비용에 맞게 최적화할 때는 다음 그래프와 유사한 워크로드가 있는 경우 온디맨드 모드가 최선의 선택입니다.

이러한 유형의 워크로드에 영향을 미치는 요소는 다음과 같습니다.

- 시간이 지남에 따라 변화하는 트래픽 패턴
- 다양한 볼륨의 요청(배치 워크로드로 인한 요청)
- 예측할 수 없는 요청 타이밍(트래픽 급증 초래)
- 지정된 시간 동안 0 또는 피크의 30% 미만으로 떨어짐




위의 요소가 있는 워크로드의 경우 Auto Scaling을 사용하여 테이블에서 트래픽 급증에 대응할 수 있는 충분한 용량을 유지하면 테이블이 과도하게 프로비저닝되고 필요 이상으로 비용이 많이 들거나 테이블이 과소 프로비저닝되고 요청이 불필요하게 제한될 수 있습니다. 온디맨드 용량 모드는 용량을 예측하거나 조정할 필요 없이 변동하는 트래픽을 처리할 수 있으므로 더 나은 선택입니다.

온디맨드 모드의 요청당 지불 요금 모델을 사용하면 실제로 사용하는 처리량에 대해서만 지불하기 때문에 유휴 용량에 대해 걱정할 필요가 없습니다. 사용된 읽기 또는 쓰기 요청당 요금이 청구되므로 요금에 실제 사용량이 직접 반영되어 비용과 성능의 균형을 쉽게 맞출 수 있습니다. 선택적으로, 개별 온디맨드 테이블 및 글로벌 보조 인덱스의 초당 최대 읽기 또는 쓰기(또는 둘 다) 처리량을 구성하여 비용과 사용량을 제한할 수도 있습니다. 자세한 내용은 온디맨드 테이블의 최대 처리량을 참조하세요.

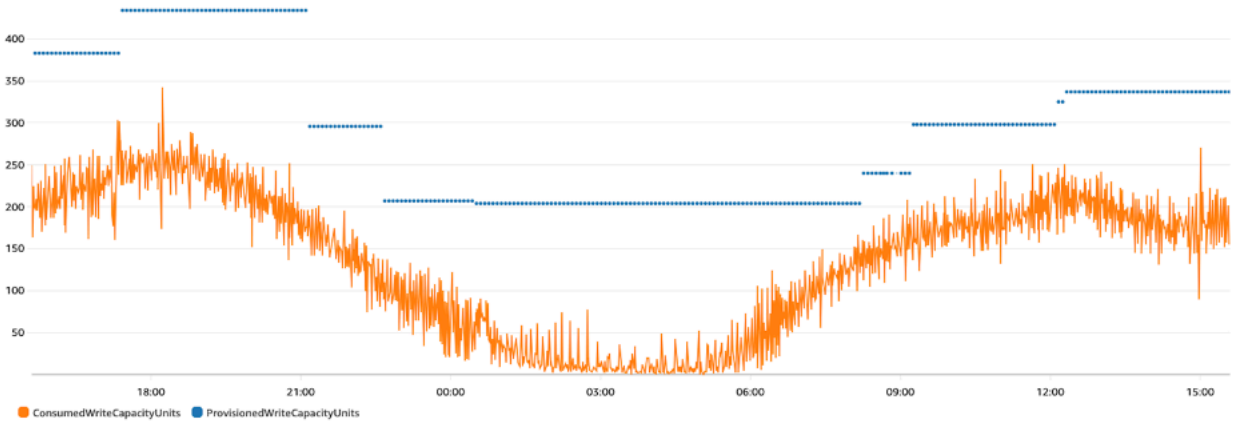
프로비저닝된 용량 모드를 선택하는 경우

프로비저닝된 용량 모드에 대한 이상적인 워크로드는 아래 그래프와 같이 사용 패턴이 더 안정적이고 예측 가능한 워크로드입니다.

> **Note**
>
> 프로비저닝된 용량에 대해 조치를 취하기 전에 14일 등 세분화된 기간으로 지표를 검토하는 것이 좋습니다.

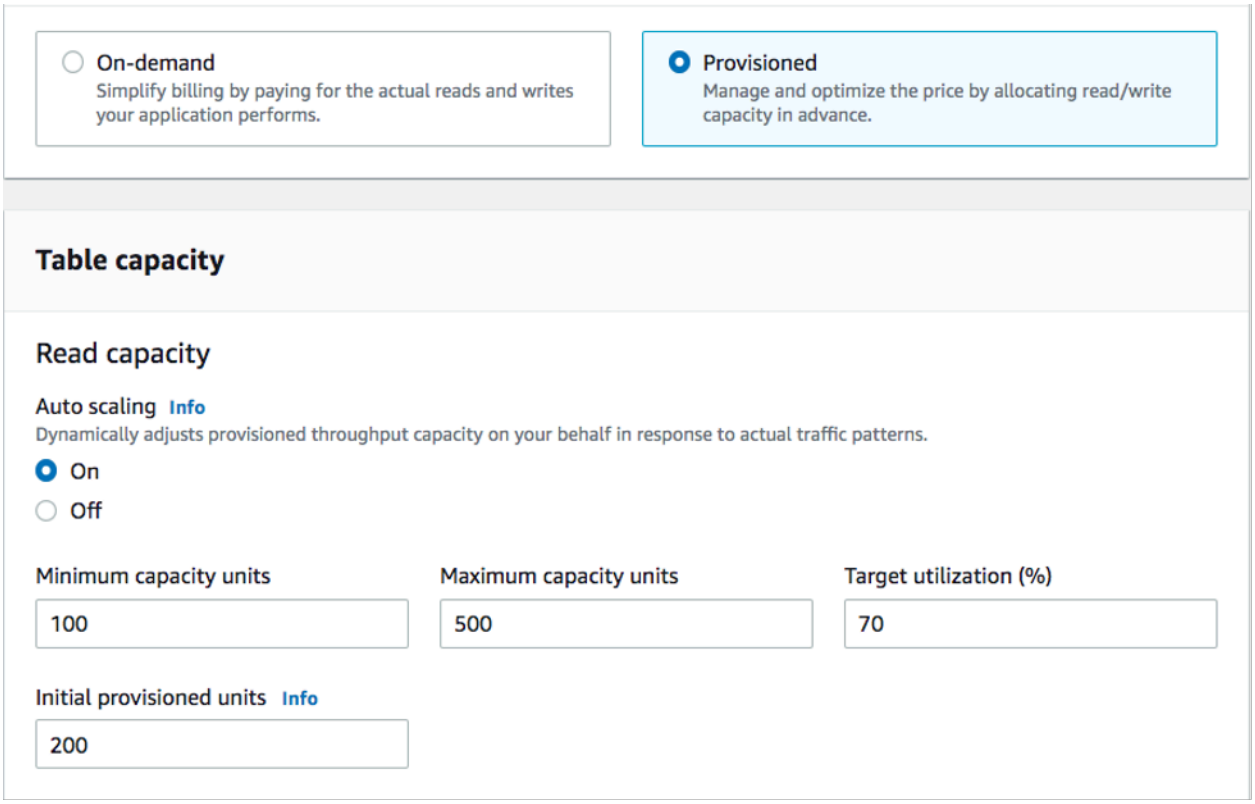
이러한 유형의 워크로드에 영향을 미치는 요소는 다음과 같습니다.

- 지정된 시간 또는 날짜에 대한 안정적이고 예측 가능한 주기적 트래픽
- 제한적인 단기 트래픽 폭주




지정된 시간 또는 일의 트래픽 볼륨이 더 안정적이기 때문에 테이블의 프로비저닝된 용량을 테이블의 실제 사용된 용량에 비교적 가깝게 설정할 수 있습니다. 프로비저닝된 용량 테이블의 비용 최적화는 궁극적으로 테이블에서 `ThrottledRequests`를 늘리지 않고 할당된 용량(파란색 선)을 사용된 용량(주황색 선)에 최대한 가깝게 만드는 연습입니다. 두 선 사이의 공간은 낭비된 용량뿐만 아니라 제한으로 인해 발생할 수 있는 나쁜 사용자 경험에 대비한 용량을 나타냅니다. 애플리케이션의 처리량 요구 사항을 예측할 수 있고 읽기 및 쓰기 용량 제어의 비용 예측성을 선호하는 경우 프로비저닝된 테이블을 계속 사용할 수 있습니다.

DynamoDB는 프로비저닝된 용량 테이블에 대한 Auto Scaling을 제공하므로 사용자를 대신하여 자동으로 균형을 조정할 수 있습니다. 이를 통해 하루 종일 사용된 용량을 추적하고 몇 가지 변수를 기반으로 테이블의 용량을 설정할 수 있습니다. 오토 스케일링을 사용하면 테이블이 과다 프로비저닝되므로 워크로드 요구 사항에 맞게 스로틀 수와 과다 프로비저닝된 용량 단위 간의 비율을 미세 조정해야 합니다.




최소 용량 단위

제한을 설정하지 않도록 테이블의 최소 용량을 설정할 수 있지만 테이블 비용이 줄어들지는 않습니다. 테이블 사용량이 낮은 기간이 있다가 갑자기 사용량이 급증한 경우 최소값으로 설정하면 Auto Scaling이 테이블 용량을 너무 낮게 설정하는 것을 방지할 수 있습니다.

최대 용량 단위

테이블의 최대 용량을 설정하여 테이블 확장을 의도한 것보다 높게 제한할 수 있습니다. 대규모 로드 테스트가 필요하지 않은 Dev 또는 Test 테이블에는 최대값을 적용하는 것이 좋습니다. 모든 테이블에 대해 최대값을 설정할 수 있지만 실수로 제한되지 않도록 프로덕션에서 사용할 때 테이블 기준선에 대해 이 설정을 정기적으로 평가해야 합니다.

목표 사용률

테이블의 목표 사용률을 설정하는 것은 프로비저닝된 용량 테이블에 대한 비용 최적화의 기본 수단입니다. 여기서 백분율 값을 낮게 설정하면 테이블이 오버프로비저닝되는 양이 늘어나 비용이 증가하지만 제한의 위험은 줄어듭니다. 백분율 값을 높게 설정하면 테이블이 오버프로비저닝되는 양이 줄어들지만 제한의 위험은 높아집니다.

### 테이블 용량 모드를 선택할 때 고려해야 할 추가 요소

두 모드 중 하나를 결정할 때 고려해야 할 몇 가지 추가 요소가 있습니다.

#### 프로비저닝된 용량 사용률

온디맨드 모드의 비용이 프로비저닝된 용량보다 낮은 시기를 확인하려면 할당된(또는 '프로비저닝된') 리소스가 얼마나 효율적으로 사용되고 있는지를 나타내는 프로비저닝된 용량 사용률을 살펴보는 것이 좋습니다. 온디맨드 모드는 프로비저닝된 평균 용량 사용률이 35% 미만인 워크로드에 비해 비용이 저렴합니다. 대부분의 경우 프로비저닝된 용량 사용률이 35%보다 높은 워크로드의 경우에도 특히 워크로드의 활동량이 적을 때가 있고 가끔 피크가 발생하는 경우 온디맨드 모드를 사용하는 것이 비용 효과적일 수 있습니다.

#### 예약 용량

프로비저닝된 용량 테이블의 경우 DynamoDB는 읽기 및 쓰기 용량에 대한 예약 용량을 구매할 수 있는 기능을 제공합니다(복제된 쓰기 용량 단위(rWCU) 및 Standard-IA 테이블은 현재 해당 없음). 예약 용량은 표준 프로비저닝 용량 요금보다 상당한 할인을 제공합니다.

두 테이블 모드 중 하나를 결정할 때는 이러한 추가 할인이 테이블 비용에 얼마나 영향을 미치는지 고려합니다. 대부분의 경우 비교적 예측하기 어려운 워크로드라도 예약 용량이 있는 과다 프로비저닝된 용량 테이블에서 실행하는 것이 더 저렴할 수 있습니다.

#### 워크로드의 예측 가능성 향상

경우에 따라 워크로드에 예측 가능한 패턴과 예측할 수 없는 패턴이 모두 있는 것처럼 보일 수 있습니다. 온디맨드 테이블에서는 이러한 워크로드도 쉽게 지원할 수 있지만, 워크로드에 있는 예측할 수 없는 패턴을 개선할 수 있다면 비용을 더 줄일 수 있을 것입니다.

이러한 패턴의 가장 일반적인 원인 중 하나는 일괄 가져오기입니다. 이러한 유형의 트래픽은 테이블을 실행할 때 제한이 발생할 정도로 테이블의 기준 용량을 초과하는 경우가 많습니다. 프로비저닝된 용량 테이블에서 이와 같은 워크로드를 계속 실행하려면 다음 옵션을 사용해봅니다.

- 예약된 시간에 일괄 처리가 발생하는 경우 실행 전에 Auto Scaling 용량을 늘리도록 예약할 수 있습니다.
- 일괄 처리가 무작위로 발생하는 경우 최대한 빨리 실행하는 대신 실행 시간을 연장하는 것이 좋습니다.
- 가져오기 속도가 처음에는 느린 속도에서 시작해서 Auto Scaling이 테이블 용량 조정을 시작할 수 있을 때까지 몇 분 동안 서서히 오르도록 증가 기간을 가져오기에 추가합니다.

# DynamoDB 테이블의 오토 스케일링 설정 평가

이 섹션에서는 DynamoDB 테이블의 Auto Scaling 설정을 평가하는 방법을 간략히 살펴봅니다. Amazon DynamoDB Auto Scaling은 애플리케이션 트래픽과 대상 사용률 지표를 기반으로 테이블 및 글로벌 보조 인덱스(GSI) 처리량을 관리하는 기능입니다. 이렇게 하면 테이블 또는 GSI가 애플리케이션 패턴에 필요한 용량을 확보할 수 있습니다.

AWS Auto Scaling 서비스는 현재 테이블 사용률을 모니터링하고 이를 목표 사용률 값(`TargetValue`)과 비교합니다. 할당된 용량을 늘리거나 줄여야 할 시기가 되면 알려 줍니다.

주제

- Auto Scaling 설정 이해하기
- 목표 사용률이 낮은(<= 50%) 테이블을 식별하는 방법
- 계절적 변동이 있는 워크로드를 해결하는 방법
- 알 수 없는 패턴으로 급증하는 워크로드를 해결하는 방법
- 연결된 애플리케이션으로 워크로드를 해결하는 방법

## Auto Scaling 설정 이해하기

목표 사용률, 초기 단계 및 최종 값에 대한 올바른 값을 정의하려면 운영 팀의 참여가 필요합니다. 이렇게 하면 AWS Auto Scaling 정책을 트리거하는 데 사용될 값을 과거 애플리케이션 사용량을 기반으로 적절하게 정의할 수 있습니다. 사용률 목표는 Auto Scaling 규칙이 적용되기 전에 일정 기간 동안 도달해야 하는 총 용량의 백분율입니다.

높은 사용률 목표(약 90%의 목표)를 설정하면 Auto Scaling이 시작되기 전에 일정 기간 동안 트래픽이 90%보다 높아야 합니다. 애플리케이션이 매우 일정하고 트래픽 급증이 발생하지 않는 한 높은 사용률 목표를 사용해서는 안 됩니다.

매우 낮은 사용률(50% 미만의 목표)을 설정하면 애플리케이션이 Auto Scaling 정책을 실행하기 전에 프로비저닝된 용량의 50%에 도달해야 합니다. 애플리케이션 트래픽이 매우 빠른 속도로 증가하지 않는 한 이는 대개 용량 미사용 및 리소스 낭비로 이어집니다.

## 목표 사용률이 낮은(<= 50%) 테이블을 식별하는 방법

AWS CLI 또는 AWS Management Console을 사용하여 DynamoDB 리소스의 Auto Scaling 정책에 대한 `TargetValues`를 모니터링하고 식별할 수 있습니다.

AWS CLI

1. 다음 명령을 실행하여 전체 리소스 목록을 반환합니다.

```
aws application-autoscaling describe-scaling-policies --service-namespace
 dynamodb
```

이 명령은 모든 DynamoDB 리소스에 발행된 Auto Scaling 정책의 전체 목록을 반환합니다. 특정 테이블에서만 리소스를 검색하려는 경우 `–resource-id parameter`를 추가하면 됩니다. 예:

```
aws application-autoscaling describe-scaling-policies --service-namespace
 dynamodb --resource-id "table/<table-name>”
```

2. 다음 명령을 실행하여 특정 GSI에 대한 Auto Scaling 정책만 반환하세요.

```
aws application-autoscaling describe-scaling-policies --service-namespace
 dynamodb --resource-id "table/<table-name>/index/<gsi-name>”
```

Auto Scaling 정책과 관련하여 알아야 할 값은 아래에 강조 표시되어 있습니다. 과다 프로비저닝을 방지하기 위해 목표값을 50%보다 높게 설정하고자 합니다. 다음과 유사한 결과가 출력되어야 합니다.

```
{
    "ScalingPolicies": [
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName": $<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:WriteCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
                }
            },
            "Alarms": [
```

```
                ...
            ],
            "CreationTime": "2022-03-04T16:23:48.641000+10:00"
        },
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName":$<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:ReadCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "DynamoDBReadCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:47.820000+10:00"
        }
    ]
}
```

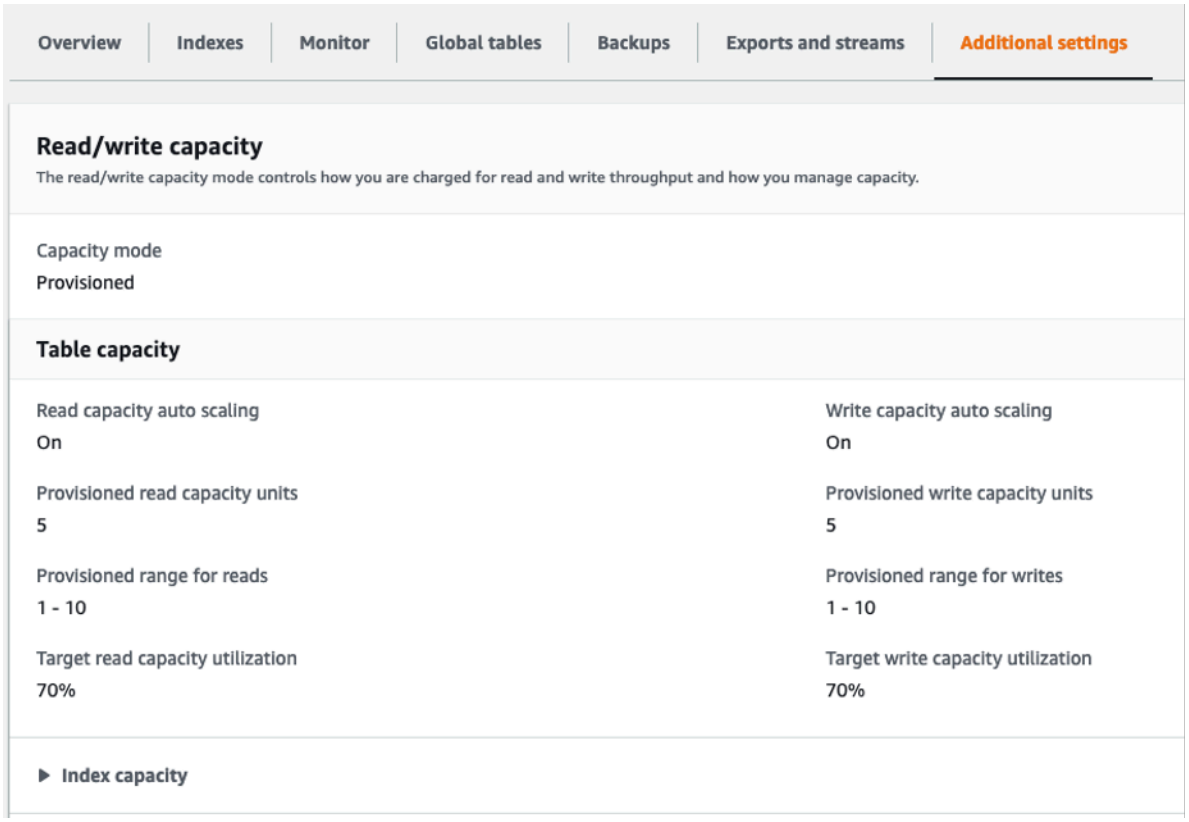
AWS Management Console

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.

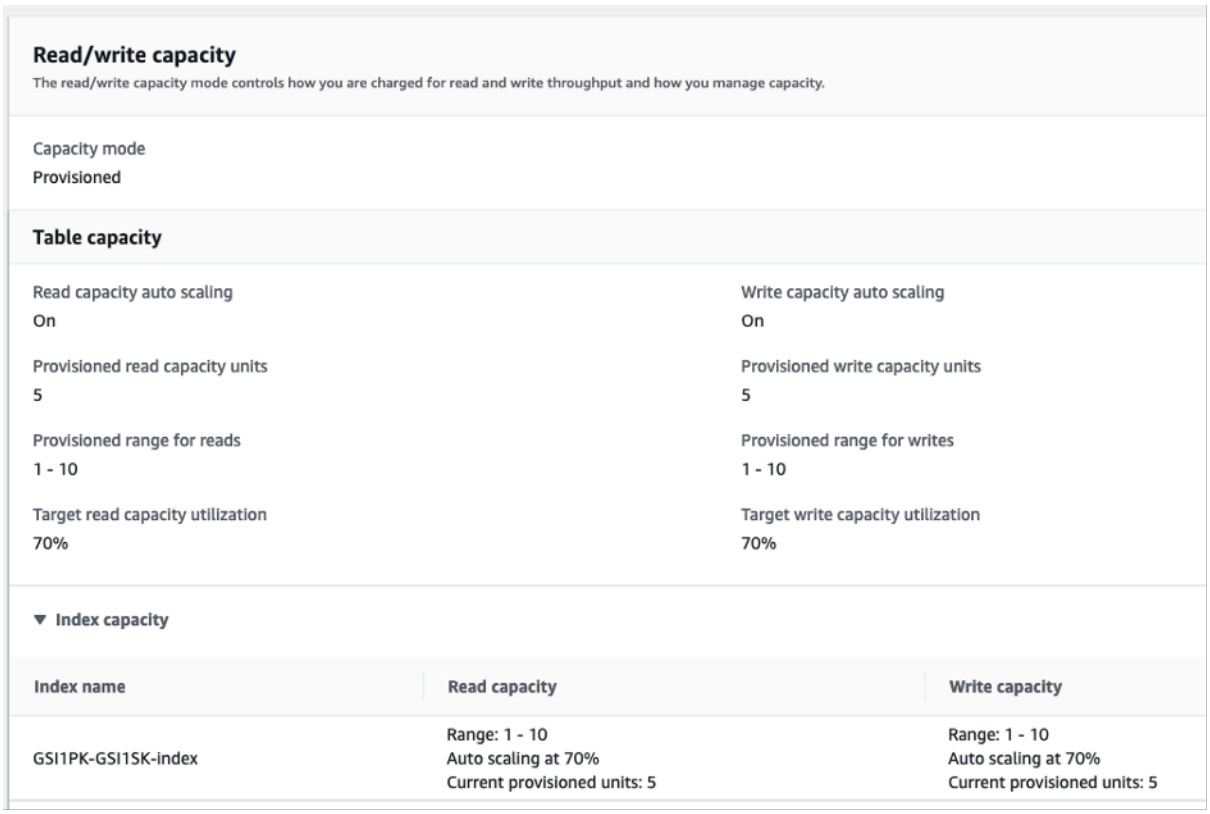
   필요한 경우 적절한 AWS 리전을 선택합니다.

2. 왼쪽 탐색 메뉴에서 Tables(테이블)를 선택합니다. Tables(테이블) 페이지에서 테이블 이름을 선택합니다.

3. 테이블 세부 정보 페이지에서 추가 설정을 선택한 다음 테이블의 Auto Scaling 설정을 검토합니다.




인덱스의 경우 인덱스 용량 섹션을 확장하여 인덱스의 Auto Scaling 설정을 검토합니다.



목표 사용률 값이 50% 이하이면 테이블 사용률 지표를 살펴보고 해당 지표가 과소 프로비저닝되었는지 과다 프로비저닝되었는지 확인해야 합니다.

계절적 변동이 있는 워크로드를 해결하는 방법

다음과 같은 시나리오를 생각해 보세요. 대부분의 경우 애플리케이션이 최소 평균값 미만으로 작동하지만 사용률 목표가 낮기 때문에 애플리케이션이 하루 중 특정 시간에 발생하는 이벤트에 신속하게 반응할 수 있고 충분한 용량이 확보되어 제한을 피할 수 있습니다. 이 시나리오는 일반적인 업무 시간(오전 9시~오후 5시)에는 사용량이 아주 많지만 업무 시간 이후에는 기본 수준에서 작동하는 애플리케이션이 있는 경우에 일반적입니다. 일부 사용자는 오전 9시 이전에 연결을 시작하므로 애플리케이션은 이 낮은 임계값을 사용하여 사용량이 많은 시간대에 필요한 용량에 빠르게 도달합니다.

이 시나리오는 다음과 같이 진행될 수 있습니다.

- 오후 5시에서 오전 9시 사이에는 `ConsumedWriteCapacity` 단위가 90에서 100 사이로 유지됩니다.
- 사용자가 오전 9시 이전에 애플리케이션에 연결하기 시작하면 용량 단위가 크게 늘어납니다(지금까지 본 최대값은 1,500WCU).
- 평균적인 애플리케이션 사용량은 근무 시간 동안 800에서 1,200 사이로 다양합니다.

이전 시나리오에 해당하는 경우 예약된 Auto Scaling을 사용하는 것을 고려해 보세요. 예약된 Auto Scaling에서는 테이블에 애플리케이션 Auto Scaling 규칙을 구성하면서도 필요한 특정 간격으로만 추가 용량을 프로비저닝하는 덜 적극적인 대상 사용률을 적용할 수 있습니다.

AWS CLI를 통해 다음 단계를 실행하여 시간과 요일을 기준으로 실행되는 예약된 Auto Scaling 규칙을 생성할 수 있습니다.

1. Application Auto Scaling을 사용하여 DynamoDB 테이블 또는 GSI를 확장 가능한 목표로 등록하세요. 확장 가능한 목표란 Application Auto Scaling이 스케일 아웃 또는 스케일 인할 수 있는 리소스입니다.

   ```
   aws application-autoscaling register-scalable-target \
       --service-namespace dynamodb \
       --scalable-dimension dynamodb:table:WriteCapacityUnits \
       --resource-id table/<table-name> \
       --min-capacity 90 \
       --max-capacity 1500
   ```

2. 요구 사항에 따라 예약된 작업 설정

시나리오를 다루려면 두 가지 규칙이 필요합니다. 하나는 스케일 업 규칙이고 다른 하나는 스케일 다운 규칙입니다. 예약된 작업을 스케일 업하기 위한 첫 번째 규칙:

```
aws application-autoscaling put-scheduled-action \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:WriteCapacityUnits \
    --resource-id table/<table-name> \
    --scheduled-action-name my-8-5-scheduled-action \
    --scalable-target-action MinCapacity=800,MaxCapacity=1500 \
    --schedule "cron(45 8 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

예약된 작업을 스케일 다운하기 위한 첫 번째 규칙:

```
aws application-autoscaling put-scheduled-action \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:WriteCapacityUnits \
    --resource-id table/<table-name> \
    --scheduled-action-name my-5-8-scheduled-down-action \
    --scalable-target-action MinCapacity=90,MaxCapacity=1500 \
    --schedule "cron(15 17 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

3. 다음 명령을 실행하여 두 규칙이 모두 활성화되었는지 확인합니다.

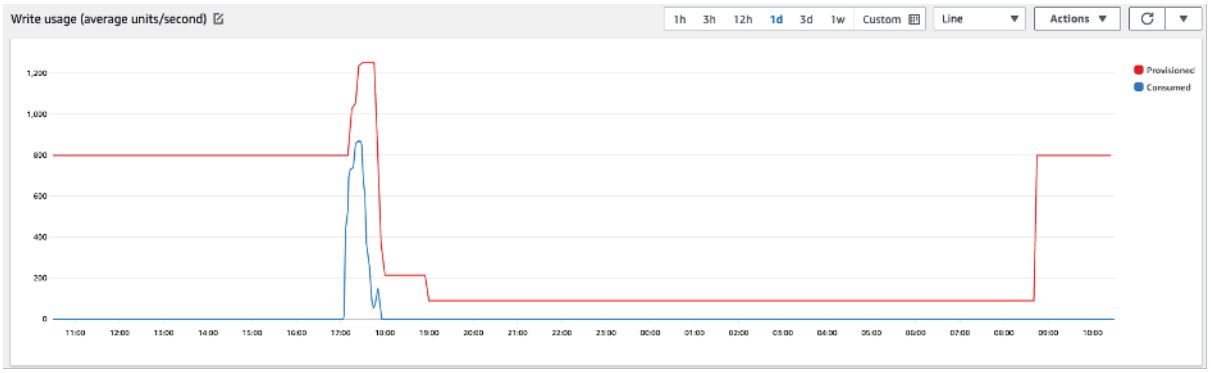
```
aws application-autoscaling describe-scheduled-actions --service-namespace dynamodb
```

다음과 같은 결과가 나와야 합니다.

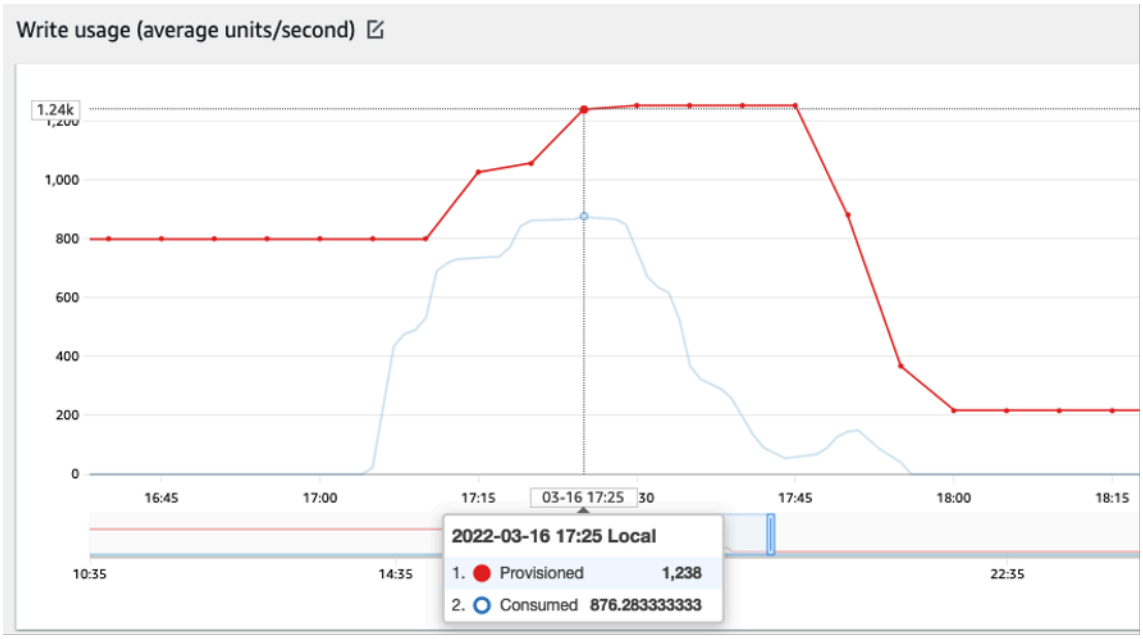
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-5-8-scheduled-down-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(15 17 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
```

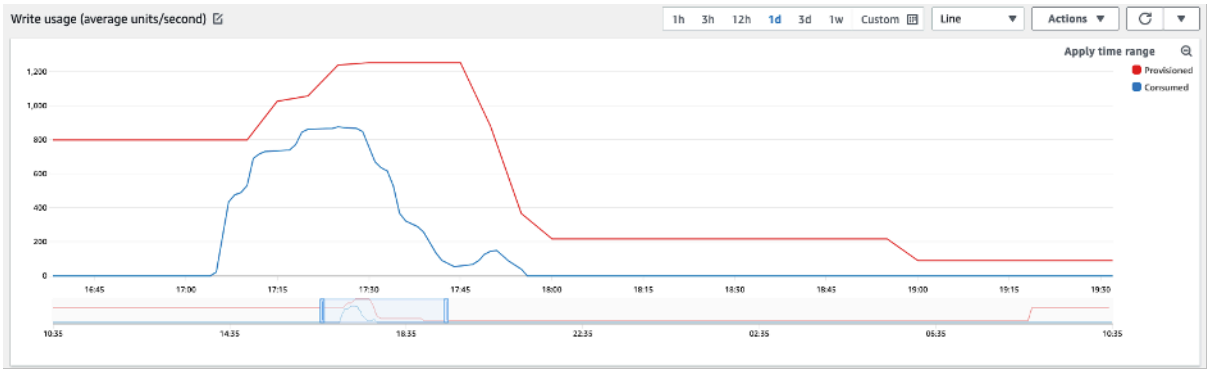
```
            "ScalableTargetAction": {
                "MinCapacity": 90,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:30:25.100000+10:00"
        },
        {
            "ScheduledActionName": "my-8-5-scheduled-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-8-5-scheduled-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(45 8 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 800,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:28:57.816000+10:00"
        }
    ]
}
```

다음 그림은 항상 70% 목표 사용률을 유지하는 샘플 워크로드를 보여 줍니다. Auto Scaling 규칙이 여전히 적용되고 있으며 처리량은 감소하지 않는다는 점에 유의하세요.




자세히 보면 애플리케이션에 급증이 발생하여 70% 오토 스케일링 임곗값이 트리거되고, 오토 스케일링이 강제로 시작되어 테이블에 필요한 추가 용량을 제공하는 것을 볼 수 있습니다. 예약된 Auto Scaling 작업은 최대값과 최소값에 영향을 미치며 이를 설정하는 것은 사용자의 책임입니다.




알 수 없는 패턴으로 급증하는 워크로드를 해결하는 방법

이 시나리오에서는 애플리케이션 패턴을 아직 모르고 워크로드에 제한이 발생하지 않도록 하기 위해 애플리케이션이 매우 낮은 사용률 목표를 사용합니다.

대신 온디맨드 용량 모드를 사용하는 것을 고려해 봅니다. 온디맨드 테이블은 트래픽 패턴을 모르는 급증하는 워크로드에 적합합니다. 온디맨드 용량 모드를 사용하면 애플리케이션이 테이블에서 수행하는 데이터 읽기 및 쓰기에 대해 요청당 요금을 지불합니다. DynamoDB는 늘어나거나 줄어드는 워크로드를 즉시 수용하기 때문에 애플리케이션에서 수행할 것으로 예상되는 읽기 및 쓰기 처리량을 지정할 필요가 없습니다.

연결된 애플리케이션으로 워크로드를 해결하는 방법

이 시나리오에서 애플리케이션은 다른 시스템에 의존합니다. 예를 들어 애플리케이션 로직의 이벤트에 따라 트래픽이 크게 급증할 수 있는 일괄 처리 시나리오가 이에 해당합니다.

이러한 이벤트에 대응하여 특정 요구 사항에 따라 테이블 용량 및 `TargetValues`를 늘릴 수 있는 사용자 지정 Auto Scaling 로직을 개발하는 것을 고려해 보세요. Amazon EventBridge를 활용하고 Lambda 및 Step Functions와 같은 AWS 서비스의 조합을 사용하여 특정 애플리케이션 요구 사항에 대응할 수 있습니다.

## DynamoDB 테이블 클래스 선택 평가

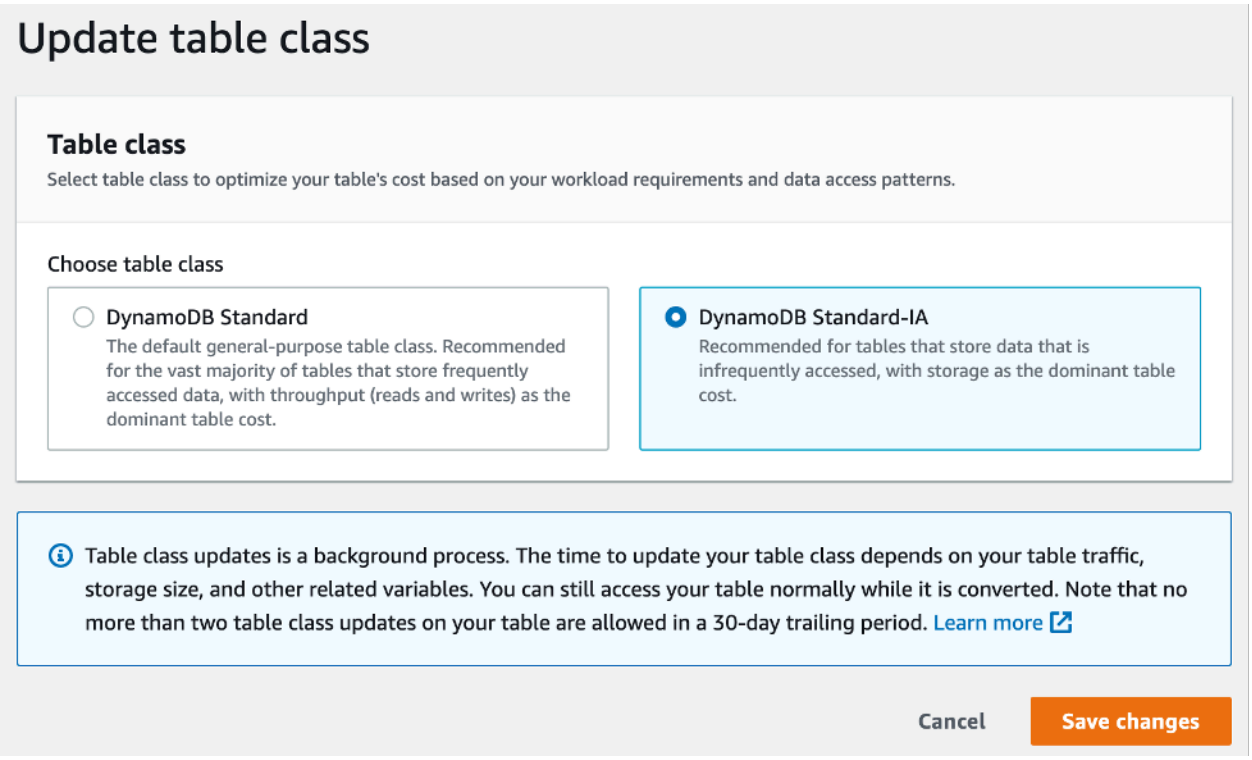
이 섹션에서는 DynamoDB 테이블에 적절한 테이블 클래스를 선택하는 방법을 간략히 살펴봅니다. Standard Infrequent-Access(Standard-IA) 테이블 클래스가 출시되면서 이제 테이블을 최적화하여 스토리지 비용을 낮추거나 처리량(throughput) 비용을 낮출 수 있습니다.

주제

- 사용 가능한 테이블 클래스
- DynamoDB Standard 테이블 클래스를 선택해야 하는 경우
- DynamoDB Standard-IA 테이블 클래스를 선택해야 하는 경우
- 테이블 클래스를 선택할 때 고려해야 할 추가 요소

### 사용 가능한 테이블 클래스

DynamoDB 테이블을 생성할 때 테이블 클래스에 대해 DynamoDB Standard 또는 DynamoDB Standard-IA를 선택해야 합니다. 테이블 클래스는 30일 동안 두 번 변경할 수 있으므로 나중에 언제든지 변경할 수 있습니다. 테이블 클래스 중 하나를 선택해도 테이블 성능, 가용성, 신뢰성 또는 내구성에는 영향을 미치지 않습니다.




Standard 테이블 클래스

Standard 테이블 클래스는 새 테이블의 기본 옵션입니다. 이 옵션은 DynamoDB의 원래 결제 잔고를 유지합니다. DynamoDB 원래 결제 잔액은 자주 액세스하는 데이터가 있는 테이블의 처리량(throughput)과 스토리지 비용의 균형을 제공합니다.

Standard-IA 테이블 클래스

Standard-IA 테이블 클래스는 업데이트 또는 읽기가 자주 발생하지 않는 데이터의 장기 저장이 필요한 워크로드에 대해 스토리지 비용 절감(~60% 더 낮음) 효과가 있습니다. 클래스는 자주 액세스하지 않도록 최적화되어 있으므로 읽기 및 쓰기 비용은 Standard 테이블 클래스보다 약간 더 높은 비용(~25% 더 높음)으로 청구됩니다.

DynamoDB Standard 테이블 클래스를 선택해야 하는 경우

DynamoDB Standard 테이블 클래스는 스토리지 비용이 전체 월별 테이블 요금의 약 50% 이하인 테이블에 가장 적합합니다. 이러한 비용 균형은 DynamoDB에 이미 저장된 항목에 정기적으로 액세스하거나 항목을 업데이트하는 워크로드를 나타냅니다.

DynamoDB Standard-IA 테이블 클래스를 선택해야 하는 경우

DynamoDB Standard-IA 테이블 클래스는 스토리지 비용이 전체 월별 테이블 요금의 약 50% 이상인 테이블에 가장 적합합니다. 이러한 비용 균형은 스토리지에 보관하는 항목보다 매월 생성하거나 읽는 항목 수가 적은 워크로드를 나타냅니다.

Standard-IA 테이블 클래스의 일반적인 용도는 액세스 빈도가 낮은 데이터를 개별 Standard-IA 테이블로 이동하는 것입니다. 자세한 내용은 Amazon DynamoDB Standard-IA 테이블 클래스를 사용한 워크로드의 스토리지 비용 최적화를 참조하세요.

테이블 클래스를 선택할 때 고려해야 할 추가 요소

두 테이블 클래스 중에서 결정할 때 결정의 일부로 고려할 가치가 있는 몇 가지 추가 요소가 있습니다.

예약 용량

Standard-IA 테이블 클래스를 사용하는 테이블의 예약 용량 구매는 현재 지원되지 않습니다. 예약 용량이 있는 Standard 테이블에서 예약 용량이 없는 Standard-IA 테이블로 전환할 때 비용 측면에서 이점이 나타나지 않을 수 있습니다.

## DynamoDB의 미사용 리소스 식별

이 섹션에서는 미사용 리소스를 정기적으로 평가하는 방법을 간략히 살펴봅니다. 애플리케이션 요구 사항이 발전함에 따라 미사용 리소스가 없고 이로 인해 불필요한 Amazon DynamoDB 비용이 발생하지 않도록 해야 합니다. 아래 설명된 절차는 미사용 리소스를 식별하는 Amazon CloudWatch 지표를 사용하여 해당 리소스를 식별하고 이에 대한 조치를 취하여 비용을 절감하는 데 도움이 됩니다.

DynamoDB의 원시 데이터를 수집하여 읽기 가능하며 실시간에 가까운 지표로 처리하는 CloudWatch를 통해 DynamoDB를 모니터링할 수 있습니다. 이러한 통계는 일정 기간 동안 유지되므로 기록 정보에 액세스하여 사용률을 더 잘 파악할 수 있습니다. 기본적으로 DynamoDB 지표 데이터는 CloudWatch에 자동으로 전송됩니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch란 무엇인가요? 및 지표 보존 기간을 참조하세요.

주제

- 미사용 리소스를 식별하는 방법
- 미사용 테이블 리소스 식별
- 미사용 테이블 리소스 정리
- 미사용 GSI 리소스 식별
- 미사용 GSI 리소스 정리
- 미사용 글로벌 테이블 정리

- 미사용 백업 또는 시점 복구(PITR) 제거

미사용 리소스를 식별하는 방법

미사용 테이블 또는 인덱스를 식별하기 위해 30일 동안 다음 CloudWatch 지표를 검토하여 테이블에 활성 읽기나 쓰기가 있는지 또는 글로벌 보조 인덱스(GSI)에 대한 읽기가 있는지 알아보겠습니다.

ConsumedReadCapacityUnits

일정 기간 동안 사용된 읽기 용량 단위의 수로, 이를 통해 사용된 용량이 얼마나 사용되었는지 추적할 수 있습니다. 또한 테이블과 테이블의 모든 글로벌 보조 인덱스 또는 특정 글로벌 보조 인덱스에 대해 소비된 총 읽기 용량을 가져올 수 있습니다.

ConsumedWriteCapacityUnits

일정 기간 동안 사용된 쓰기 용량 단위의 수로, 이를 통해 사용된 용량을 얼마나 사용했는지 추적할 수 있습니다. 또한 테이블과 테이블의 모든 글로벌 보조 인덱스 또는 특정 글로벌 보조 인덱스에 대해 소비된 총 쓰기 용량을 가져올 수 있습니다.

미사용 테이블 리소스 식별

Amazon CloudWatch는 미사용 리소스를 식별하는 데 사용할 DynamoDB 테이블 지표를 제공하는 모니터링 및 관찰 가능 서비스입니다. CloudWatch 지표는 AWS Management Console과 AWS Command Line Interface를 통해 확인할 수 있습니다

AWS Command Line Interface

AWS Command Line Interface를 통해 테이블 지표를 보려면 다음 명령을 사용할 수 있습니다.

1. 먼저 테이블의 읽기를 평가합니다.

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

테이블을 미사용 테이블로 잘못 식별하지 않으려면 더 오랜 기간 동안 지표를 평가하세요. 적절한 시작 시간 및 종료 시간 범위(예: 30일)와 적절한 기간(예: 86400)을 선택합니다.

반환된 데이터에서 0보다 큰 모든 합계는 해당 기간 동안 수신된 읽기 트래픽을 평가 중인 테이블을 나타냅니다.

다음 결과에는 평가 기간에 읽기 트래픽을 수신한 테이블이 표시됩니다.

```
        {
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
        },
```

다음 결과에는 평가 기간에 읽기 트래픽을 수신하지 않은 테이블이 표시됩니다.

```
        {
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

2. 다음으로 테이블의 쓰기를 평가하세요.

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedWriteCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

테이블을 미사용 테이블로 잘못 식별하지 않으려면 더 오랜 기간 동안 지표를 평가하는 것이 좋습니다. 적절한 시작 시간 및 종료 시간 범위(예: 30 days(30일))와 적절한 기간(예: 86400)을 선택합니다.

반환된 데이터에서 0보다 큰 모든 합계는 해당 기간 동안 수신된 읽기 트래픽을 평가 중인 테이블을 나타냅니다.

다음 결과에는 평가 기간에 쓰기 트래픽을 수신한 테이블이 표시됩니다.

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

다음 결과에는 평가 기간에 쓰기 트래픽을 수신하지 않은 테이블이 표시됩니다.

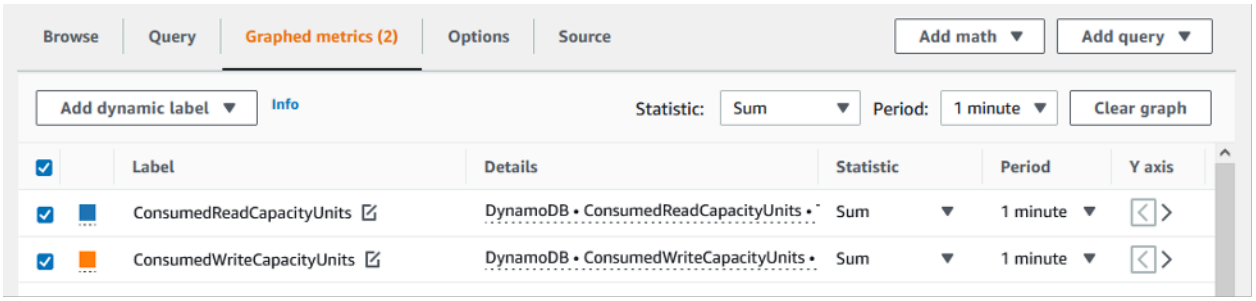
```
        {
            "Timestamp": "2022-07-31T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

### AWS Management Console

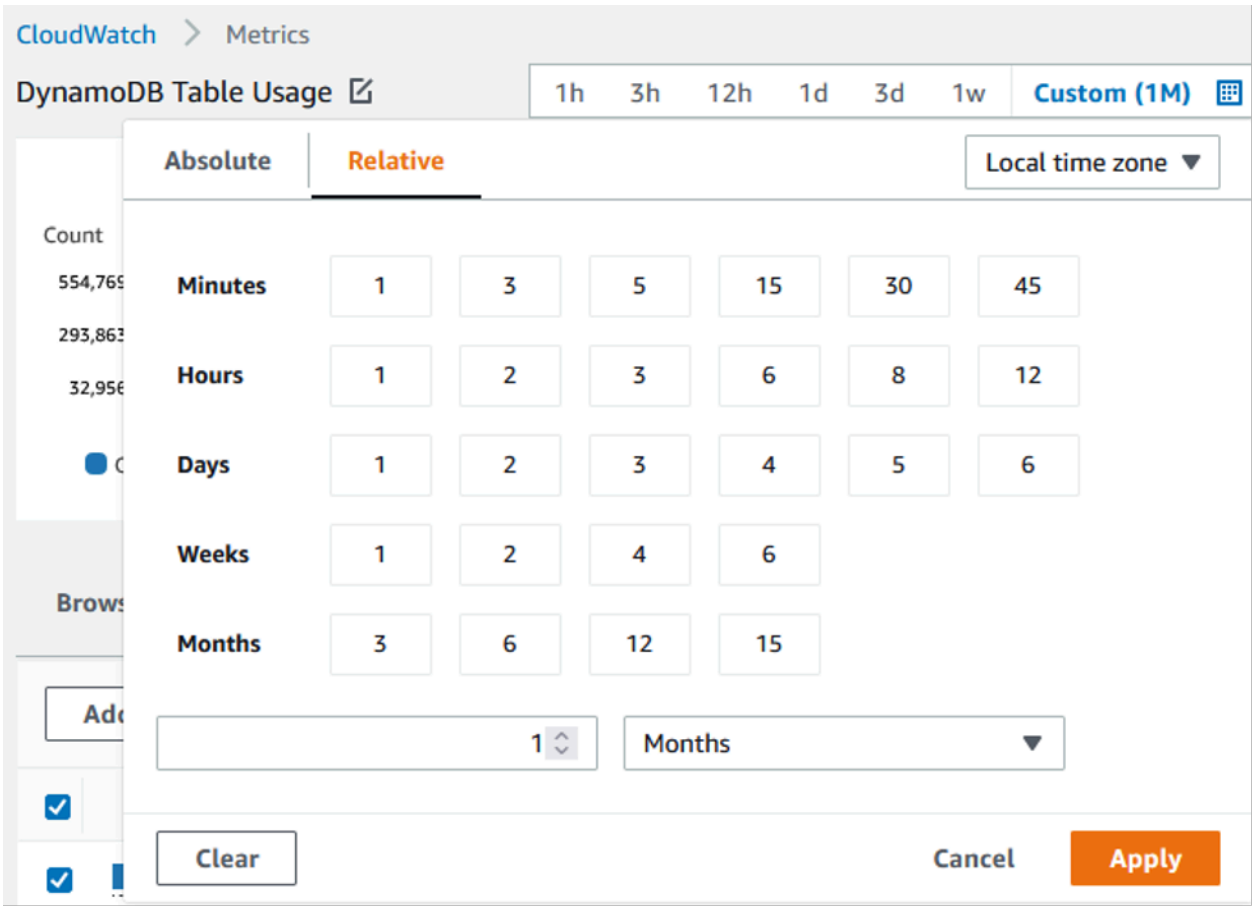
다음 단계에서는 AWS Management Console을 통해 리소스 사용률을 평가할 수 있습니다.

1. AWS에 로그인하고 https://console.aws.amazon.com/cloudwatch/의 CloudWatch 서비스 페이지로 이동합니다. 필요한 경우 콘솔의 오른쪽 상단에서 해당 AWS 리전을 선택합니다.
2. 왼쪽 탐색 메뉴에서 Metrics(지표)를 선택한 다음 All metrics(모든 지표)를 선택합니다.
3. 위 작업을 수행하면 두 개의 패널이 있는 대시보드가 열립니다. 상단 패널에서는 현재 그래프로 표시된 지표를 확인할 수 있습니다. 하단에서는 그래프로 나타낼 수 있는 지표를 선택할 수 있습니다. 하단 패널에서 DynamoDB를 선택합니다.
4. DynamoDB 지표 선택 패널에서 Table Metrics(테이블 지표) 범주를 선택하여 현재 리전의 테이블에 해당하는 지표를 표시합니다.
5. 메뉴를 아래로 스크롤하여 테이블 이름을 식별한 후 테이블에 해당하는 `ConsumedReadCapacityUnits` 및 `ConsumedWriteCapacityUnits` 지표를 선택합니다.

6. Graphed metrics (2)(그래프로 표시된 지표 (2)) 탭을 선택하고 Statistic(통계) 열을 Sum(합계)으로 조정합니다.

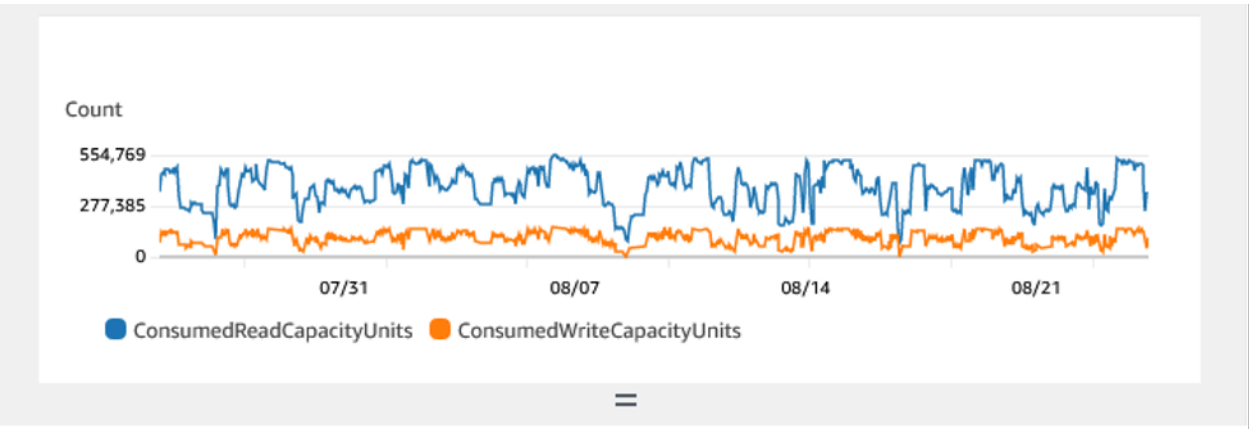



7. 테이블을 미사용 테이블로 잘못 식별하지 않으려면 더 오랜 기간 동안 지표를 평가하는 것이 좋습니다. 그래프 패널 상단에서 테이블을 평가할 적절한 기간(예: 1개월)을 선택합니다. Custom(사용자 지정)을 선택하고 드롭다운에서 1 Months(1개월)를 선택한 다음 Apply(적용)를 선택합니다.

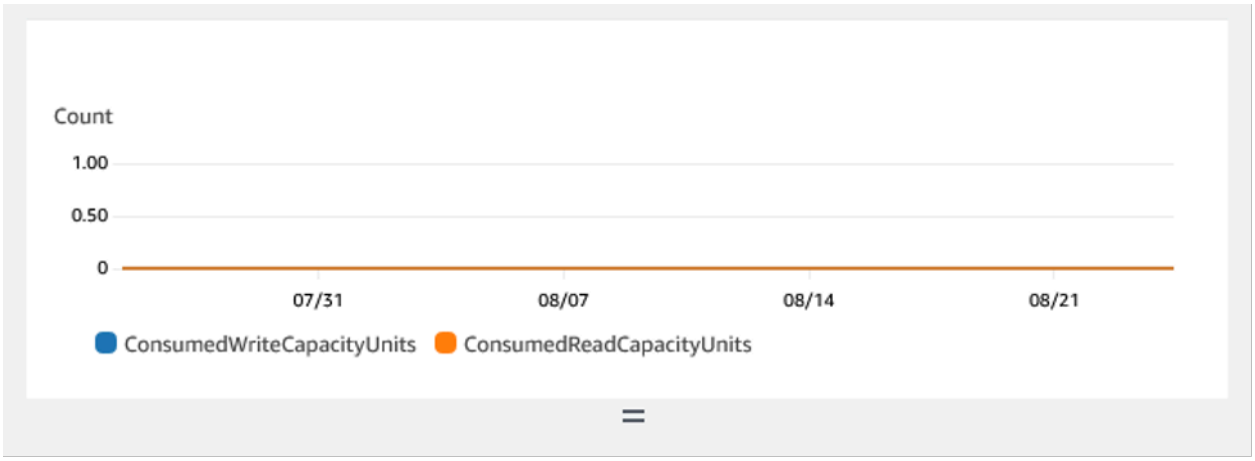



8. 테이블의 그래프로 표시된 지표를 평가하여 사용 중인지 확인하세요. 지표가 0을 초과하면 평가된 기간 동안 테이블이 사용되었음을 나타냅니다. 읽기와 쓰기 모두에 대해 0에 있는 평면 그래프는 미사용 테이블을 나타냅니다.

다음은 읽기 트래픽이 포함된 테이블이 표시된 이미지입니다.




다음은 읽기 트래픽이 포함되지 않은 테이블이 표시된 이미지입니다.




### 미사용 테이블 리소스 정리

미사용 테이블 리소스를 식별한 경우 다음과 같은 방법으로 지속적인 비용을 줄일 수 있습니다.

> **Note**
>
> 미사용 테이블을 식별했지만 나중에 액세스해야 할 경우를 대비하여 계속 사용할 수 있으려면 온디맨드 모드로 전환하는 것이 좋습니다. 그렇지 않으면 테이블을 완전히 백업하고 삭제할 수도 있습니다.

### 용량 모드

DynamoDB는 DynamoDB 테이블의 데이터 읽기, 쓰기 및 저장에 대한 요금을 청구합니다.

DynamoDB에는 테이블에서 읽기 및 쓰기 처리를 위한 특정 결제를 위한 두 가지 용량 모드가 있습니다. 읽기/쓰기 용량 모드는 읽기 및 쓰기 처리량에 대한 청구 방법과 용량 관리 방법을 제어합니다.

온디맨드 모드 테이블의 경우 애플리케이션에서 수행할 것으로 예상되는 읽기 및 쓰기 처리량을 지정할 필요가 없습니다. DynamoDB에서는 읽기 요청 단위 및 쓰기 요청 단위의 측면에서 애플리케이션이 테이블에서 수행하는 읽기 및 쓰기에 대해 요금이 부과됩니다. 테이블/인덱스에 활동이 없는 경우 처리량(throughput)에 대한 비용은 지불하지 않지만 스토리지 요금은 계속 부과됩니다.

테이블 클래스

DynamoDB는 비용을 최적화할 수 있도록 설계된 두 가지 테이블 클래스도 제공합니다. DynamoDB Standard 테이블 클래스가 기본값이며 대다수의 워크로드에 권장됩니다. DynamoDB Standard-Infrequent Access(DynamoDB Standard-IA) 테이블 클래스는 스토리지 비용이 많이 드는 테이블에 최적화되어 있습니다.

테이블이나 인덱스에 활동이 없는 경우 스토리지가 주요 비용이 될 수 있으며 테이블 클래스를 변경하면 상당한 비용 절감 효과를 얻을 수 있습니다.

테이블 삭제

미사용 테이블을 발견하여 삭제하려는 경우 먼저 데이터를 백업하거나 내보내는 것이 좋습니다.

AWS Backup을 통해 수행된 백업은 콜드 스토리지 계층화를 활용하여 비용을 더욱 절감할 수 있습니다. AWS Backup을 통해 백업을 활성화하는 방법에 대한 정보는 DynamoDB와 함께 DynamoDB에서 AWS Backup 사용 설명서를 참조하고, 수명 주기를 사용하여 백업을 콜드 스토리지로 이동하는 방법에 대한 정보는 백업 관리 계획 설명서를 참조하세요.

또는 테이블의 데이터를 S3로 내보내도록 선택할 수도 있습니다. 이렇게 하려면 Amazon S3 S3로 내보내기 설명서를 참조하세요. 데이터를 내보낸 후 S3 Glacier Instant Retrieval, S3 Glacier Flexile Retrieval 또는 S3 Glacier Deep Archive를 활용하여 비용을 더욱 절감하려면 스토리지 수명 주기 관리를 참조하세요.

테이블을 백업한 후 AWS Management Console 또는 AWS Command Line Interface를 통해 테이블을 삭제할 수 있습니다.

미사용 GSI 리소스 식별

미사용 글로벌 보조 서버를 식별하는 단계는 사용하지 않는 테이블을 식별하는 단계와 유사합니다. DynamoDB는 기본 테이블에 작성된 항목에 GSI의 파티션 키로 사용되는 속성이 포함된 경우 해당 항목을 GSI로 복제하므로 기본 테이블이 사용 중인 경우 사용하지 않은 GSI의

ConsumedWriteCapacityUnits 0보다 클 가능성이 여전히 높습니다. 따라서 GSI가 사용되지 않았는지 확인하기 위해 ConsumedReadCapacityUnits 지표만 평가하게 됩니다.

AWS AWS CLI를 통해 테이블 지표를 보려면 다음 명령을 사용할 수 있습니다.

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
Name=GlobalSecondaryIndexName,Value=<index-name>
```

테이블을 미사용 테이블로 잘못 식별하지 않으려면 더 오랜 기간 동안 지표를 평가하는 것이 좋습니다. 적절한 시작 시간 및 종료 시간 범위(예: 30 days(30일))와 적절한 기간(예: 86400)을 선택합니다.

반환된 데이터에서 0보다 큰 모든 합계는 해당 기간 동안 수신된 읽기 트래픽을 평가 중인 테이블을 나타냅니다.

다음 결과에는 평가 기간에 읽기 트래픽을 수신한 GSI가 표시됩니다.

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 36319167.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 1869136.0,
          "Unit": "Count"
        },
```

다음 결과에는 평가 기간에 최소 읽기 트래픽을 수신한 GSI가 표시됩니다.

```
        {
          "Timestamp": "2022-08-28T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-15T21:20:00Z",
          "Sum": 2.0,
          "Unit": "Count"
```

```
        },
```

다음 결과에는 평가 기간에 읽기 트래픽을 전혀 수신하지 않은 GSI가 표시됩니다.

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
```

미사용 GSI 리소스 정리

미사용 GSI를 식별한 경우 삭제하도록 선택할 수 있습니다. GSI에 있는 모든 데이터는 기본 테이블에도 있으므로 GSI를 삭제하기 전에 추가 백업이 필요하지 않습니다. 나중에 GSI가 다시 필요할 경우 테이블에 다시 추가될 수 있습니다.

자주 사용하지 않는 GSI를 확인했다면 삭제하거나 비용을 줄일 수 있도록 애플리케이션의 설계 변경을 고려해야 합니다. 예를 들어 DynamoDB 스캔은 많은 양의 시스템 리소스를 사용할 수 있으므로 자주 사용하지 않아야 하지만 지원하는 액세스 패턴이 매우 드물게 사용되는 경우 GSI보다 비용 효율적일 수 있습니다.

또한 빈번하지 않은 액세스 패턴을 지원하기 위해 GSI가 필요한 경우 더 제한된 속성 세트를 프로젝션하는 것이 좋습니다. 이렇게 하면 빈번하지 않은 액세스 패턴을 지원하기 위해 기본 테이블에 대한 후속 쿼리가 필요할 수 있지만 이를 통해 스토리지 및 쓰기 비용을 크게 줄일 수 있습니다.

미사용 글로벌 테이블 정리

Amazon DynamoDB 글로벌 테이블은 복제 솔루션을 직접 구축하여 관리하지 않고도 다중 리전의 다중 활성 데이터베이스를 배포할 수 있는 완전관리형 솔루션을 제공합니다.

글로벌 테이블은 재해 복구를 위한 보조 영역은 물론 사용자와 가까운 데이터에 대한 액세스 지연 시간을 줄이는 데 적합합니다.

데이터에 대한 액세스 지연 시간을 줄이기 위해 리소스에 글로벌 테이블 옵션을 활성화했지만 재해 복구 전략의 일부가 아닌 경우, CloudWatch 지표를 평가하여 두 복제본 모두 읽기 트래픽을 적극적으로

처리하고 있는지 확인하세요. 한 복제본이 읽기 트래픽을 처리하지 않는 경우 미사용 리소스일 수 있습니다.

글로벌 테이블이 재해 복구 전략의 일부인 경우 활성/대기 패턴에서는 읽기 트래픽을 수신하지 않는 하나의 복제본이 있을 수 있습니다.

미사용 백업 또는 시점 복구(PITR) 제거

DynamoDB는 두 가지 백업 스타일을 제공합니다. 시점 복구는 최대 35일간 연속 백업을 제공하여 실수로 쓰거나 삭제하지 못하도록 보호하며, 온디맨드 백업은 장기간 저장할 수 있는 스냅샷을 생성합니다. 복구 기간을 1~35일 사이의 값으로 설정할 수 있습니다. 두 가지 유형의 백업 모두 비용이 발생합니다.

테이블에 더 이상 필요하지 않을 수 있는 백업이 활성화되어 있는지 확인하려면 DynamoDB에 대한 백업 및 복원 및 DynamoDB의 시점 백업 설명서를 참조하세요.

## DynamoDB 테이블 사용 패턴 평가

이 섹션에서는 DynamoDB 테이블을 효율적으로 사용하고 있는지 평가하는 방법을 간략히 살펴봅니다. DynamoDB에 최적화되지 않은 특정 사용 패턴이 있으며, 이러한 패턴은 성능 및 비용 측면에서 모두 최적화될 수 있는 여지가 있습니다.

주제

- 강력히 일관된 읽기 작업 줄이기
- 읽기 작업을 위한 트랜잭션 횟수 줄이기
- 스캔 줄이기
- 속성 이름 줄이기
- 유지 시간(TTL) 활성화
- 글로벌 테이블을 리전 간 백업으로 대체

강력히 일관된 읽기 작업 줄이기

DynamoDB를 사용하면 요청별로 읽기 일관성을 구성할 수 있습니다. 기본적으로 읽기 요청은 최종적으로 일관됩니다. 최종 읽기 일관성은 최대 4KB의 데이터에 대해 0.5RCU의 요금이 부과됩니다.

분산 워크로드의 대부분은 유연하며 최종 일관성을 허용합니다. 하지만 강력히 일관된 읽기가 필요한 액세스 패턴이 있을 수 있습니다. 강력히 일관된 읽기는 최대 4KB의 데이터에 대해 1RCU의 요금이 부

과되므로 읽기 비용이 두 배로 늘어납니다. DynamoDB는 동일한 테이블에서 두 일관성 모델을 모두 사용할 수 있는 유연성을 제공합니다.

워크로드와 애플리케이션 코드를 평가하여 강력히 일관된 읽기가 필요한 경우에만 사용되는지 확인할 수 있습니다.

읽기 작업을 위한 트랜잭션 횟수 줄이기

DynamoDB를 사용하면 특정 작업을 전부 또는 전무 방식으로 그룹화할 수 있습니다. 즉, DynamoDB를 사용하여 ACID 트랜잭션을 실행할 수 있습니다. 그러나 관계형 데이터베이스의 경우처럼 모든 작업에 대해 이 접근 방식을 따를 필요는 없습니다.

최대 4KB의 트랜잭션 읽기 작업은 동일한 양의 데이터를 최종 일관성 방식으로 읽는 데 소비되는 기본 0.5RCU와 달리 2RCU를 소비합니다. 쓰기 작업의 경우 비용이 두 배로 늘어납니다. 즉, 최대 1KB의 트랜잭션 쓰기에 2WCU가 소비됩니다.

테이블의 모든 작업이 트랜잭션인지 확인하려면 테이블의 CloudWatch 지표를 트랜잭션 API로 필터링할 수 있습니다. 트랜잭션 API가 테이블의 `SuccessfulRequestLatency` 지표에서 사용할 수 있는 유일한 그래프인 경우 모든 작업이 이 테이블의 트랜잭션임을 확인할 수 있습니다. 또는 전체 용량 사용률 추세가 트랜잭션 API 추세와 일치한다면 트랜잭션 API가 주를 이루는 것처럼 보이는 애플리케이션 설계를 다시 살펴보는 것도 좋습니다.

스캔 줄이기

`Scan` 작업이 광범위하게 사용되는 이유는 일반적으로 DynamoDB 데이터에 대한 분석 쿼리를 실행해야 하기 때문입니다. 대형 테이블에서 `Scan` 작업을 자주 실행하면 비효율적이고 비용이 많이 들 수 있습니다.

더 나은 대안은 S3로 내보내기 기능을 사용하고 특정 시점을 선택하여 테이블 상태를 S3로 내보내는 것입니다. AWS가 제공하는 Athena와 같은 서비스를 통해 테이블의 용량을 전혀 소비하지 않고도 데이터에 대한 분석 쿼리를 실행할 수 있습니다.

`Scan` 작업 빈도는 `Scan`에 대한 `SuccessfulRequestLatency` 지표 아래의 `SampleCount` 통계를 사용하여 확인할 수 있습니다. `Scan` 작업이 실제로 매우 빈번하다면 액세스 패턴과 데이터 모델을 재평가해야 합니다.

속성 이름 줄이기

DynamoDB에 있는 항목의 총 크기는 속성 이름 길이와 값을 더하여 결정됩니다. 속성 이름이 길면 스토리지 비용이 증가할 뿐만 아니라 RCU/WCU 소비량도 증가할 수 있습니다. 속성 이름은 긴 것보다 짧

은 것이 좋습니다. 속성 이름을 짧게 지정하면 다음 4KB/1KB 경계 내에서 항목 크기를 제한하는 데 도움이 될 수 있으며, 이후에는 데이터에 액세스하기 위해 추가 RCU/WCU를 소비하게 됩니다.

유지 시간(TTL) 활성화

유지 시간(TTL)은 사용자가 항목에 설정한 만료 시간보다 오래된 항목을 식별하여 테이블에서 제거합니다. 시간이 지남에 따라 데이터가 늘어나고 오래된 데이터는 무용지물이 되는 경우 테이블에서 TTL을 활성화하면 데이터를 줄이고 스토리지 비용을 절감하는 데 도움이 될 수 있습니다.

TTL의 또 다른 유용한 측면은 만료된 항목이 DynamoDB 스트림에서 발생한다는 점입니다. 따라서 데이터에서 데이터를 제거하는 대신 스트림에서 해당 항목을 소비하여 더 저렴한 스토리지 계층에 보관할 수 있습니다. 또한 TTL을 통해 항목을 삭제하면 추가 비용이 들지 않으므로 용량을 소비하지 않으며 정리 애플리케이션을 설계하는 데 드는 오버헤드도 없습니다.

글로벌 테이블을 리전 간 백업으로 대체

글로벌 테이블을 사용하면 서로 다른 리전에서 여러 개의 활성 복제본 테이블을 유지 관리할 수 있습니다. 이러한 테이블은 모두 쓰기 작업을 허용하고 서로 간에 데이터를 복제할 수 있습니다. 복제본을 쉽게 설정할 수 있으며 동기화가 자동으로 관리됩니다. 복제본은 최종 쓰기 우선 전략을 사용하여 일관된 상태로 수렴합니다.

글로벌 테이블을 순전히 장애 조치 또는 재해 복구(DR) 전략의 일부로만 사용하는 경우 복구 시점 목표와 복구 시간 목표 요구 사항이 상대적으로 관대한 리전 간 백업 복사본으로 대체할 수 있습니다. 빠른 로컬 액세스와 99.999%의 고가용성이 필요하지 않은 경우 글로벌 테이블 복제본을 유지 관리하는 것이 장애 조치를 위한 최선의 방법이 아닐 수 있습니다.

대안으로 AWS Backup을 사용하여 DynamoDB 백업을 관리하는 것을 고려해 보세요. 정기적인 백업을 예약하고 리전 간에 복사하여 글로벌 테이블을 사용하는 것보다 더 비용 효과적인 접근 방식으로 DR 요구 사항을 충족할 수 있습니다.

## DynamoDB 스트림 사용량 평가

이 섹션에서는 DynamoDB Streams 사용량을 평가하는 방법을 간략히 살펴봅니다. DynamoDB에 최적화되지 않은 특정 사용 패턴이 있으며, 이러한 패턴은 성능 및 비용 측면에서 모두 최적화될 수 있는 여지가 있습니다.

다음과 같은 스트리밍 및 이벤트 기반 사용 사례를 위한 두 가지 기본 스트리밍 통합이 있습니다.

- Amazon DynamoDB Streams
- Amazon Kinesis Data Streams

이 페이지에서는 이러한 옵션에 대한 비용 최적화 전략에 초점을 맞출 것입니다. 대신 두 옵션 중 하나를 선택하는 방법을 알아보려면 변경 데이터 캡처의 스트리밍 옵션 섹션을 참조하세요.

주제

- DynamoDB Streams에 대한 비용 최적화
- Kinesis Data Streams에 대한 비용 최적화
- 두 가지 유형의 Streams 사용에 대한 비용 최적화 전략

DynamoDB Streams에 대한 비용 최적화

DynamoDB Streams의 요금 페이지에 설명되어 있는 것처럼, DynamoDB는 테이블의 처리량 용량 모드에 관계없이 테이블의 DynamoDB 스트림에 대한 읽기 요청 수에 따라 요금을 청구합니다. DynamoDB 스트림에 대한 읽기 요청은 DynamoDB 테이블에 대한 읽기 요청과 다릅니다.

스트림과 관련된 각 읽기 요청은 `GetRecords` API 호출 형태로, 응답에서 최대 1,000개 또는 1MB 상당 중 먼저 도달하는 양의 레코드를 반환할 수 있습니다. 다른 DynamoDB 스트림 API에는 요금이 부과되지 않으며 DynamoDB 스트림은 유휴 상태에 대해 요금이 청구되지 않습니다. 즉, DynamoDB 스트림에 대한 읽기 요청이 없는 경우 테이블에서 DynamoDB 스트림이 활성화되어 있어도 요금이 부과되지 않습니다.

다음은 DynamoDB Streams용 몇 가지 소비자 애플리케이션입니다.

- AWS Lambda 함수
- Amazon Kinesis Data Streams 기반 애플리케이션
- AWS SDK를 사용하여 구축된 고객 소비자 애플리케이션

DynamoDB Streams의 AWS Lambda 기반 소비자가 전송한 읽기 요청은 무료이지만, 다른 유형의 소비자에 의한 호출에는 요금이 부과됩니다. 매달 Lambda를 사용하지 않는 소비자가 전송한 첫 2,500,000건의 읽기 요청도 무료로 제공됩니다. 이는 각 AWS 리전의 AWS 계정에 있는 모든 DynamoDB 스트림에 대한 모든 읽기 요청에 적용됩니다.

DynamoDB Streams 사용량 모니터링

결제 콘솔의 DynamoDB Streams 요금은 AWS 계정 내 AWS 리전의 모든 DynamoDB 스트림에 대해 함께 그룹화됩니다. 현재 DynamoDB 스트림에 태그를 지정하는 것은 지원되지 않으므로 비용 할당 태그를 사용하여 DynamoDB Streams의 세부 비용을 식별할 수는 없습니다. `GetRecords` 호출 볼륨은 DynamoDB 스트림 수준에서 얻어 스트림당 요금을 계산할 수 있습니다. 볼륨은 DynamoDB 스트림의 CloudWatch 지표 `SuccessfulRequestLatency`와 해당 `SampleCount` 통계로 표시됩니다. 이 지표

에는 지속적인 복제를 수행하기 위해 글로벌 테이블에서 실행하는 `GetRecords` 호출과AWS Lambda 소비자가 실행하는 호출이 포함되며, 둘 다에는 요금이 부과되지 않습니다. DynamoDB Streams에서 게시한 다른 CloudWatch 지표에 대한 자세한 내용은 DynamoDB 지표 및 차원 섹션을 참조하세요.

AWS Lambda를 소비자로 사용

AWS Lambda 함수를 DynamoDB Streams의 소비자로 사용하는 것이 가능한지 평가해 보세요. 그러면 DynamoDB 스트림에서 읽기와 관련된 비용을 절감할 수 있기 때문입니다. 반면 DynamoDB Streams Kinesis 어댑터 또는 SDK 기반 소비자 애플리케이션은 DynamoDB 스트림에 대한 `GetRecords` 호출 수에 따라 요금이 부과됩니다.

Lambda 함수 호출 요금은 표준 Lambda 요금을 기준으로 부과되지만 DynamoDB Streams에 의해 발생하는 요금은 없습니다. Lambda는 초당 4회의 기본 속도로 DynamoDB 스트림의 샤드에서 레코드를 폴링합니다. 레코드를 사용할 수 있으면 Lambda가 함수를 간접 호출하고 결과를 기다립니다. 처리가 성공하면 Lambda가 레코드를 더 받을 때까지 폴링을 재개합니다.

DynamoDB Streams Kinesis 어댑터 기반 소비자 애플리케이션 조정

Lambda 기반이 아닌 소비자가 전송한 읽기 요청은 DynamoDB Streams에 대해 요금이 청구되므로 실시간에 가까운 요구 사항과 소비자 애플리케이션이 DynamoDB 스트림을 폴링해야 하는 횟수 사이의 균형을 찾는 것이 중요합니다.

DynamoDB Streams Kinesis 어댑터 기반 애플리케이션을 사용하여 DynamoDB 스트림을 폴링하는 빈도는 구성된 `idleTimeBetweenReadsInMillis` 값에 따라 결정됩니다. 이 파라미터는 동일한 샤드에 대한 이전 `GetRecords` 호출에서 레코드가 반환되지 않는 경우 소비자가 샤드를 처리하기 전에 기다려야 하는 시간(밀리초)을 결정합니다. 이 파라미터의 기본값은 1000밀리초입니다. 실시간에 가까운 처리가 필요하지 않은 경우 이 파라미터를 늘려 소비자 애플리케이션이 더 적은 `GetRecords` 호출을 실행하고 DynamoDB Streams 호출에 최적화되도록 할 수 있습니다.

Kinesis Data Streams에 대한 비용 최적화

Kinesis 데이터 스트림이 DynamoDB 테이블에 대한 변경 데이터 캡처 이벤트를 전달할 대상으로 설정된 경우, Kinesis 데이터 스트림에 별도의 크기 조정 관리가 필요할 수 있으며 이는 전체 비용에 영향을 미칩니다. DynamoDB는 DynamoDB 서비스에서 대상 Kinesis 데이터 스트림으로 시도한 1KB DynamoDB 항목 크기로 각 단위가 구성된 변경 데이터 캡처 단위(CDU)를 기준으로 요금을 청구합니다.

DynamoDB 서비스 요금 외에도 표준 Kinesis 데이터 스트림 요금이 부과됩니다. 요금 페이지에 언급되어 있는 것처럼 서비스 요금은 용량 모드(프로비저닝 및 온디맨드)에 따라 달라지며, 이는 DynamoDB 테이블 용량 모드와 달리 사용자가 정의합니다. 개괄적으로 보면 Kinesis Data Streams는 DynamoDB

서비스에 의해 스트림으로 수집된 데이터뿐만 아니라 용량 모드를 기준으로도 시간당 요금을 청구합니다. Kinesis 데이터 스트림의 사용자 구성에 따라 데이터 검색(온디맨드 모드의 경우), 데이터 보존 기간 연장(기본 24시간 이후), 향상된 팬아웃 소비자 검색과 같은 추가 요금이 부과될 수 있습니다.

Kinesis Data Streams 사용 모니터링

DynamoDB용 Kinesis Data Streams는 표준 Kinesis 데이터 스트림 CloudWatch 지표와 함께 DynamoDB의 지표를 게시합니다. 충분하지 않은 Kinesis 데이터 스트림 용량이나 Kinesis 데이터 스트림 저장 데이터를 암호화하도록 구성된 AWS KMS 서비스와 같은 종속 구성 요소로 인해 Kinesis 서비스가 DynamoDB 서비스의`Put` 시도를 제한할 수 있습니다.

DynamoDB 서비스에서 Kinesis 데이터 스트림에 대해 게시한 CloudWatch 지표에 대한 자세한 내용은 Kinesis Data Streams를 사용하여 변경 데이터 캡처 모니터링 섹션을 참조하세요. 제한으로 인한 서비스 재시도에 따른 추가 비용을 피하려면 프로비저닝 모드에서 Kinesis 데이터 스트림의 크기를 적절하게 조정하는 것이 중요합니다.

Kinesis Data Streams에 적절한 용량 모드 선택

Kinesis Data Streams는 프로비저닝 모드와 온디맨드 모드의 두 가지 용량 모드에서 지원됩니다.

- Kinesis 데이터 스트림과 관련된 워크로드에 예측 가능한 애플리케이션 트래픽, 일정하거나 점진적으로 증가하는 트래픽 또는 정확하게 예측할 수 있는 트래픽이 있는 경우 Kinesis Data Streams의 프로비저닝 모드가 적합하며 비용 효율성이 더 높습니다.
- 새로운 워크로드이거나, 워크로드에 예측할 수 없는 애플리케이션 트래픽이 있거나, 용량을 관리하지 않으려는 경우에는 Kinesis Data Streams의 온디맨드 모드가 적합하며 비용 효율성이 더 높습니다.

비용을 최적화하는 모범 사례는 Kinesis 데이터 스트림과 연결된 DynamoDB 테이블에 Kinesis 데이터 스트림의 프로비저닝 모드를 활용할 수 있는 예측 가능한 트래픽 패턴이 있는지 평가하는 것입니다. 새로운 워크로드인 경우 처음 몇 주 동안은 Kinesis Data Streams의 온디맨드 모드를 사용하고 CloudWatch 지표를 검토하여 트래픽 패턴을 파악한 다음, 워크로드의 특성에 따라 해당 스트림을 프로비저닝 모드로 전환할 수 있습니다. 프로비저닝 모드의 경우 Kinesis Data Streams의 샤드 관리 고려 사항에 따라 샤드 수를 추정할 수 있습니다.

DynamoDB용 Kinesis Data Streams를 사용하는 소비자 애플리케이션 평가

Kinesis Data Streams는 DynamoDB Streams처럼 `GetRecords` 호출 수에 대해 요금을 부과하지 않으므로 소비자 애플리케이션은 빈도가 `GetRecords` 제한 한도 미만인 경우 원하는 횟수만큼 호출을 실행할 수 있습니다. Kinesis Data Streams의 온디맨드 모드의 경우 데이터 읽기 요금은 GB당 부과됩니

다. 프로비저닝 모드 Kinesis Data Streams의 경우 데이터가 7일 미만인 경우 읽기 요금이 부과되지 않습니다. Lambda가 Kinesis Data Streams 소비자 기능을 하는 경우 Lambda는 초당 1회의 기본 속도로 Kinesis 스트림의 각 샤드에서 레코드를 폴링합니다.

두 가지 유형의 Streams 사용에 대한 비용 최적화 전략

AWS Lambda 소비자를 위한 이벤트 필터링

Lambda 이벤트 필터링을 사용하면 필터 기준에 따라 Lambda 함수 호출 배치에서 사용할 수 없도록 이벤트를 삭제할 수 있습니다. 이는 소비자 함수 로직 내에서 원치 않는 스트림 레코드를 처리하거나 폐기하는 Lambda 비용을 최적화합니다. 이벤트 필터링을 구성하고 필터링 기준을 작성하는 방법에 대한 자세한 내용은 Lambda 이벤트 필터링을 참조하세요.

AWS Lambda 소비자 조정

호출당 처리량을 늘리도록 `BatchSize`를 늘리고, 처리 중복(추가 비용 발생) 방지를 위해 `BisectBatchOnFunctionError`를 활성화하며, 재시도를 너무 많이 하지 않도록 `MaximumRetryAttempts`를 설정하는 등 Lambda 구성 파라미터를 조정하여 비용을 더욱 최적화할 수 있습니다. 기본적으로 실패한 소비자 Lambda 호출은 스트림에서 레코드가 만료될 때까지 무한정 재시도됩니다. 이는 DynamoDB Streams의 경우 약 24시간이며 Kinesis Data Streams의 경우 24시간에서 최대 1년까지 구성할 수 있습니다. 위에서 언급한 옵션을 포함하여 DynamoDB Stream 소비자를 위해 사용할 수 있는 추가 Lambda 구성 옵션은 AWS Lambda 개발자 안내서를 참조하세요.

## DynamoDB 테이블에서 적절한 규모의 프로비저닝을 위해 프로비저닝된 용량 평가

이 섹션에서는 DynamoDB 테이블에 적절한 규모의 프로비저닝이 있는지 평가하는 방법을 간략히 살펴봅니다. 워크로드가 발전함에 따라 운영 절차를 적절하게 수정해야 합니다. 특히 DynamoDB 테이블이 프로비저닝 모드로 구성되어 있고 테이블을 과다 프로비저닝하거나 과소 프로비저닝할 위험이 있는 경우에는 더욱 그렇습니다.

아래에 설명된 절차에는 프로덕션 애플리케이션을 지원하는 DynamoDB 테이블에서 캡처해야 하는 통계 정보가 필요합니다. 애플리케이션 동작을 이해하려면 애플리케이션의 데이터 계절성을 포착할 수 있을 만큼 충분히 중요한 기간을 정의해야 합니다. 예를 들어 애플리케이션이 주간 패턴을 보이는 경우 3주 기간을 사용하면 애플리케이션 처리량 요구 사항을 분석할 시간을 충분히 확보할 수 있습니다.

어디서부터 시작해야 할지 모르겠다면 아래 계산에 최소 한 달 분량의 데이터 사용량을 사용해 보세요.

DynamoDB 테이블은 용량을 평가하는 동안 읽기 용량 단위(RCU)와 쓰기 용량 단위(WCU)를 별개로 구성할 수 있습니다. 테이블에 글로벌 보조 인덱스(GSI)가 구성된 경우 소비할 처리량을 지정해야 합니다. 이 처리량은 기본 테이블의 RCU 및 WCU와도 별개입니다.

> **Note**
>
> 로컬 보조 인덱스(LSI)는 기본 테이블의 용량을 소비합니다.

주제

- DynamoDB 테이블에서 소비 지표를 검색하는 방법
- 과소 프로비저닝된 DynamoDB 테이블을 식별하는 방법
- 과다 프로비저닝된 DynamoDB 테이블을 식별하는 방법

DynamoDB 테이블에서 소비 지표를 검색하는 방법

테이블 및 GSI 용량을 평가하려면 다음 CloudWatch 지표를 모니터링하고 적절한 차원을 선택하여 테이블 또는 GSI 정보를 검색하세요.

| 읽기 용량 단위 | 쓰기 용량 단위 |
| --- | --- |
| `ConsumedReadCapacityUnits` | `ConsumedWriteCapacityUnits` |
| `ProvisionedReadCapacityUnits` | `ProvisionedWriteCapacityUnits` |
| `ReadThrottleEvents` | `WriteThrottleEvents` |

AWS CLI 또는 AWS Management Console을 통해 이 작업을 할 수 있습니다.

AWS CLI

테이블 소비 지표를 검색하기 전에 먼저 CloudWatch API를 사용하여 일부 과거 데이터 포인트를 캡처해야 합니다.

먼저 두 개의 파일(`write-calc.json` 및 `read-calc.json`)을 만듭니다. 이러한 파일은 테이블 또는 GSI에 대한 계산을 나타냅니다. 아래 표에 표시된 대로 일부 필드를 환경에 맞게 업데이트해야 합니다.

| 필드 이름 | 정의 | 예시 |
| --- | --- | --- |
| `<table-name>` | 분석하려는 테이블 이름 | SampleTable |

| 필드 이름 | 정의 | 예시 |
| --- | --- | --- |
| <period> | 사용률 목표를 평가하는 데 사용할 기간(초 단위) | 1시간인 경우 3,600 지정 |
| <start-time> | 평가 간격의 시작 부분으로, ISO8601 형식으로 지정 | 2022-02-21T23:00:00 |
| <end-time> | 평가 간격의 끝부분으로, ISO8601 형식으로 지정 | 2022-02-22T06:00:00 |

쓰기 계산 파일은 지정된 날짜 범위에 해당하는 기간 동안 프로비저닝되고 소비된 WCU 수를 검색합니다. 또한 분석에 사용할 사용률도 생성합니다. `write-calc.json` 파일의 전체 콘텐츠는 다음과 같아야 합니다.

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ProvisionedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
```

```
          "MetricName": "ConsumedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>""
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedWCU/PERIOD(consumedWCU)",
      "Label": "Consumed WCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedWCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<ent-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

읽기 계산 파일은 비슷한 파일을 사용합니다. 이 파일은 지정된 날짜 범위에 해당하는 기간 동안 프로비저닝되고 소비된 RCU 수를 검색합니다. `read-calc.json` 파일의 콘텐츠는 다음과 같아야 합니다.

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedRCU",
      "MetricStat": {
        "Metric": {
```

```
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ProvisionedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ConsumedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedRCU/PERIOD(consumedRCU)",
      "Label": "Consumed RCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedRCU)",
      "Label": "Utilization Percentage",
```

```
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<end-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

파일을 만들었으면 사용률 데이터 검색을 시작할 수 있습니다.

1. 쓰기 사용률 데이터를 검색하려면 다음 명령을 실행합니다.

```
aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
```

2. 읽기 사용률 데이터를 검색하려면 다음 명령을 실행합니다.

```
aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
```

두 쿼리의 결과는 분석에 사용될 일련의 JSON 형식 데이터 포인트입니다. 결과는 지정한 데이터 포인트 수, 기간 및 고유한 워크로드 데이터에 따라 달라집니다. 결과는 다음과 같을 것입니다.

```
{
    "MetricDataResults": [
        {
            "Id": "utilizationPercentage",
            "Label": "Utilization Percentage",
            "Timestamps": [
                "2022-02-22T05:00:00+00:00",
                "2022-02-22T04:00:00+00:00",
                "2022-02-22T03:00:00+00:00",
                "2022-02-22T02:00:00+00:00",
                "2022-02-22T01:00:00+00:00",
                "2022-02-22T00:00:00+00:00",
                "2022-02-21T23:00:00+00:00"
            ],
            "Values": [
                91.55364583333333,
                55.066631944444445,
                2.6114930555555556,
                24.9496875,
```

```
                40.94725694444445,
                25.61819444444444,
                0.0
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

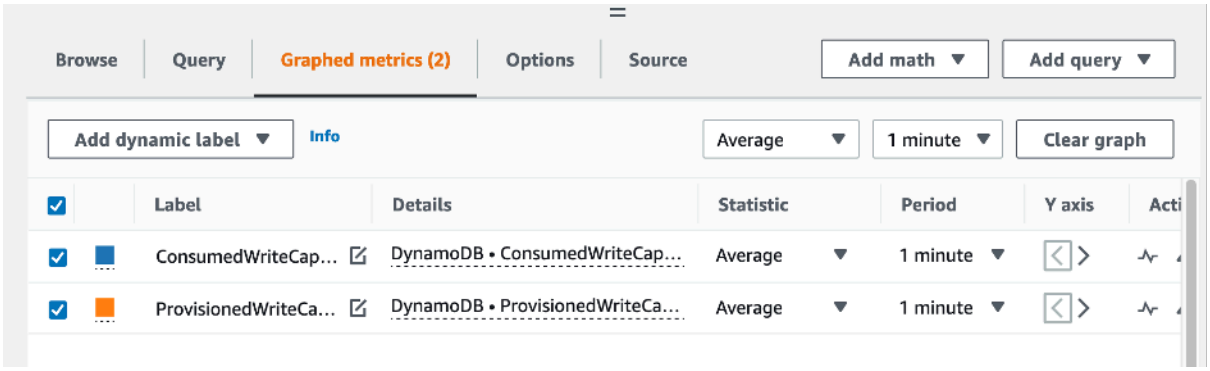
> **Note**
>
> 짧은 기간과 긴 시간 범위를 지정하는 경우 스크립트에서 기본값인 24로 설정되어 있는 `MaxDatapoints`를 수정해야 할 수 있습니다. 이는 시간당 하나의 데이터 포인트, 하루에 24개의 데이터 포인트를 나타냅니다.

AWS Management Console

1. AWS Management Console에 로그인하고 CloudWatch 서비스 페이지로 이동합니다. 필요한 경우 적절한 AWS 리전을 선택합니다.
2. 왼쪽 탐색 메뉴에서 지표 섹션을 찾은 다음 모든 지표를 선택합니다.
3. 그러면 두 개의 패널이 있는 대시보드가 열립니다. 상단 패널에는 그래픽이 표시되고 하단 패널에는 그래프로 표시하려는 지표가 나타납니다. DynamoDB를 선택합니다.
4. 테이블 지표를 선택합니다. 그러면 현재 리전에 있는 테이블이 표시됩니다.
5. 검색 창을 사용하여 테이블 이름을 검색하고 쓰기 작업 지표(`ConsumedWriteCapacityUnits` 및 `ProvisionedWriteCapacityUnits`)를 선택합니다.

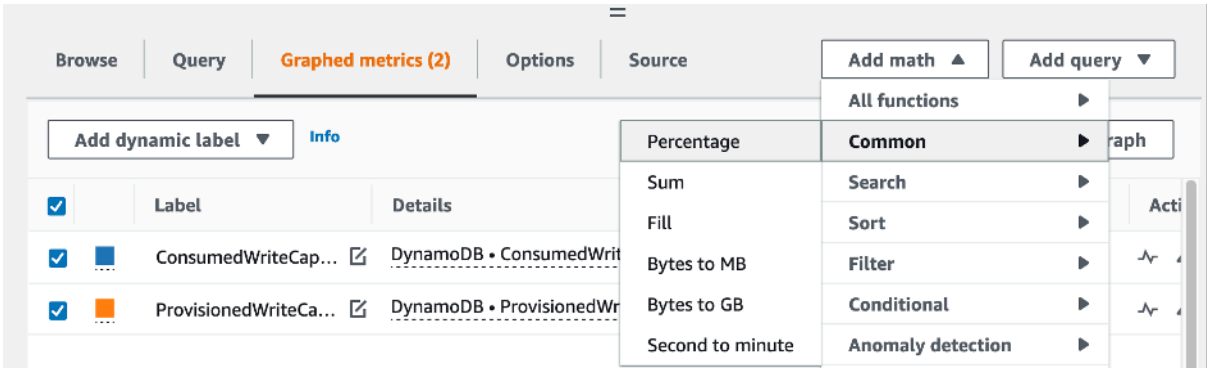
   > **Note**
   >
   > 이 예제에서는 쓰기 작업 지표에 대해 설명하지만 다음 단계를 사용하여 읽기 작업 지표를 그래프로 표시할 수도 있습니다.

6. 그래프로 표시된 지표(2) 탭을 선택하여 수식을 수정합니다. 기본적으로 CloudWatch는 그래프에 통계 함수 Average를 선택합니다.

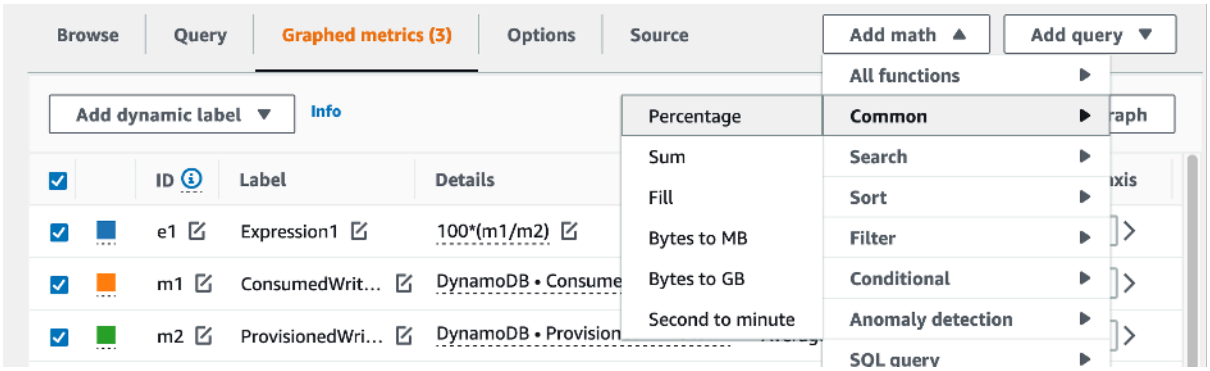



7. 그래프로 표시된 지표 두 개를 모두 선택한 상태에서(왼쪽의 확인란) Add math(수식 추가), Common(공통) 메뉴를 차례로 선택한 다음 Percentage 함수를 선택합니다. 절차를 두 번 반복합니다.

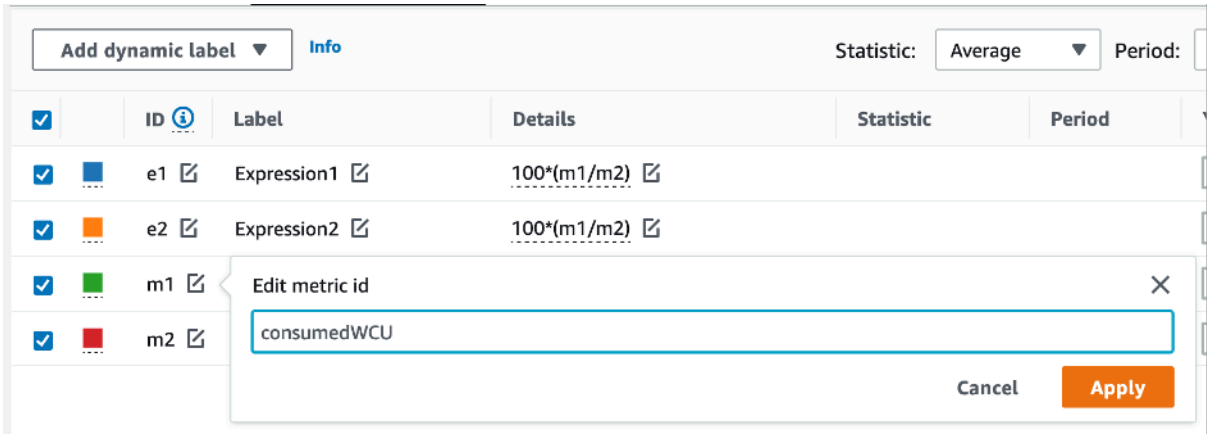
   처음으로 Percentage 함수를 선택할 때:



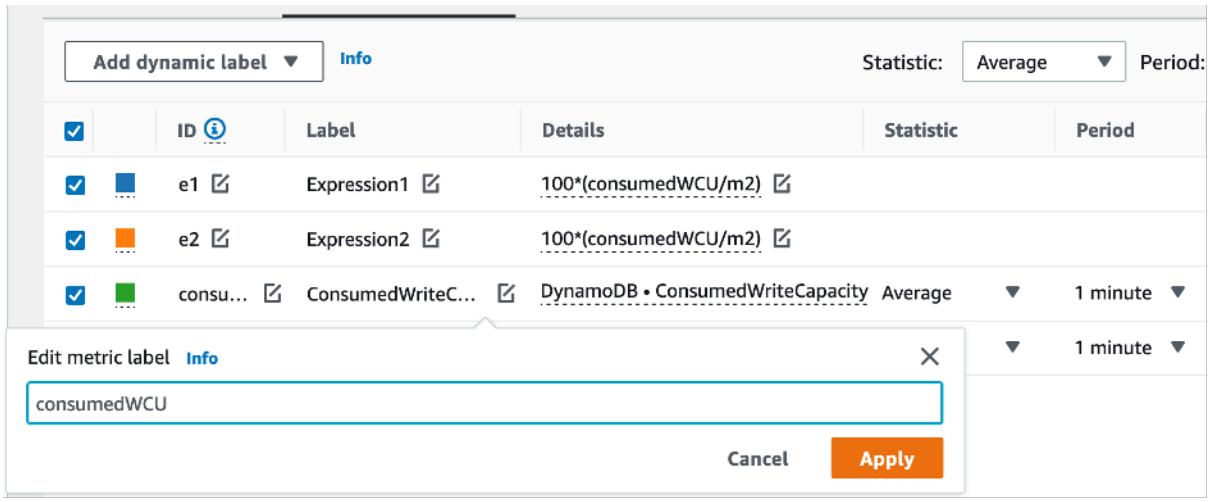

   두 번째로 Percentage 함수를 선택할 때:



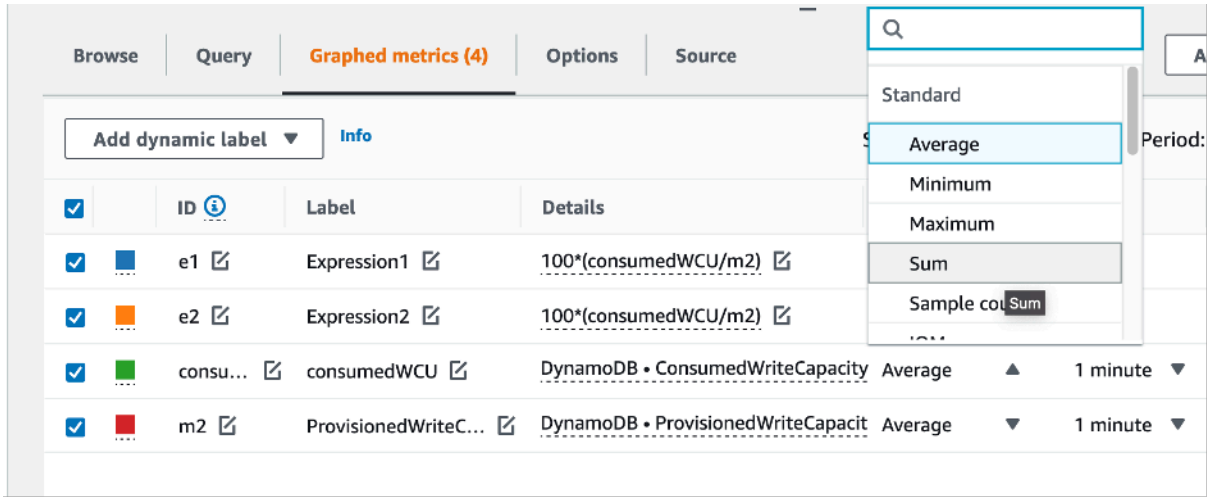

8. 이제 하단 메뉴에 네 개의 지표가 있어야 합니다. `ConsumedWriteCapacityUnits` 계산 작업을 해봅시다. 일관성을 유지하려면 AWS CLI 섹션에서 사용한 이름과 동일한 이름을 사용해야 합니다. m1 ID를 클릭하고 이 값을 ConsumedWCU로 변경합니다.

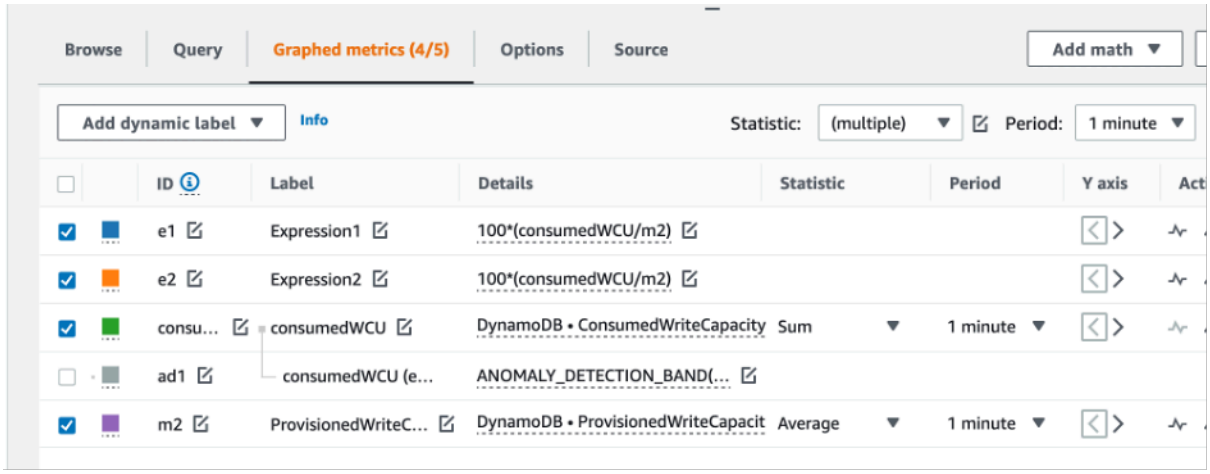



ConsumedWriteCapacityUnit 레이블의 이름을 **consumedWCU**로 변경합니다.

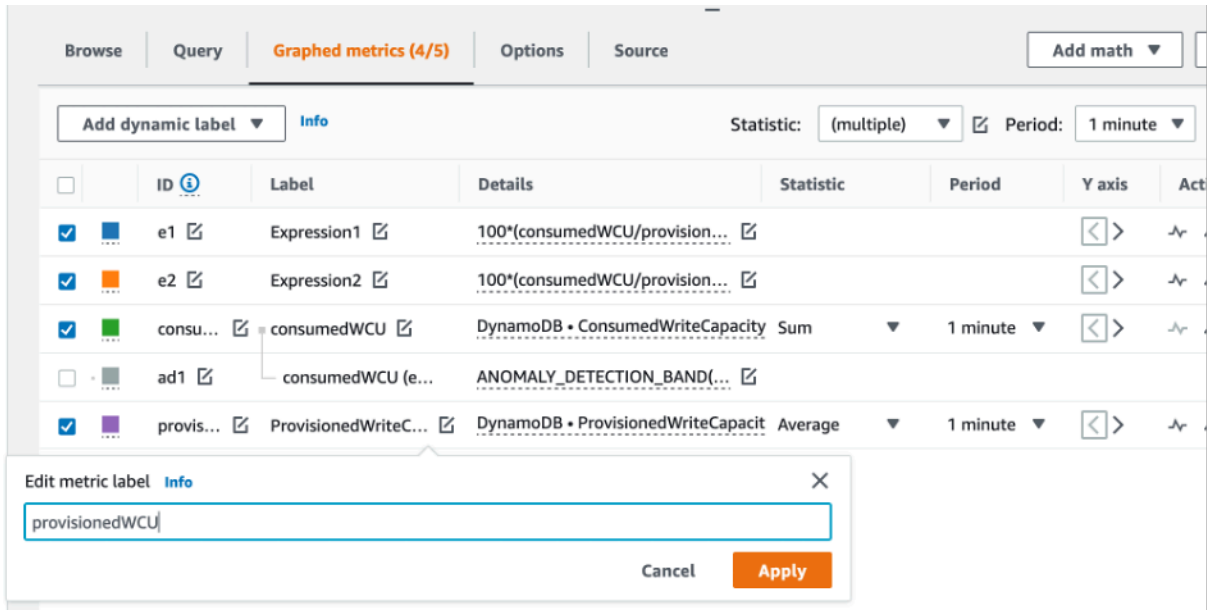



9. 통계를 Average에서 Sum으로 변경합니다. 이 작업을 수행하면 ANOMALY_DETECTION_BAND라는 또 다른 지표가 자동으로 생성됩니다. 이 절차의 범위에서는 새로 생성된 ad1 지표에서 확인란을 선택 취소하여 이 절차를 무시하겠습니다.



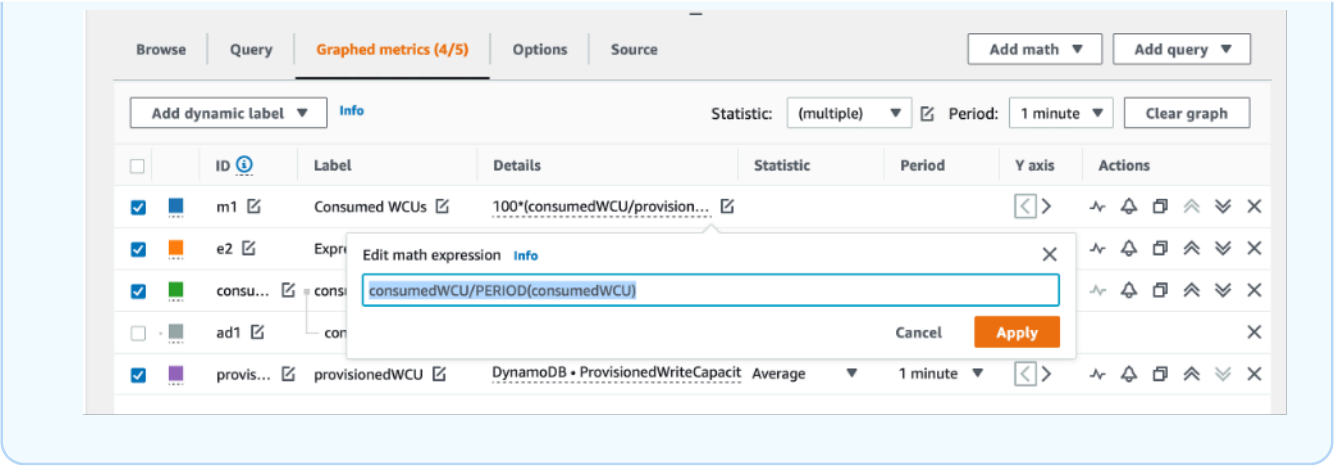



10. 8단계를 반복하여 m2 ID의 이름을 provisionedWCU로 변경합니다. 통계는 Average로 설정된 상태로 둡니다.

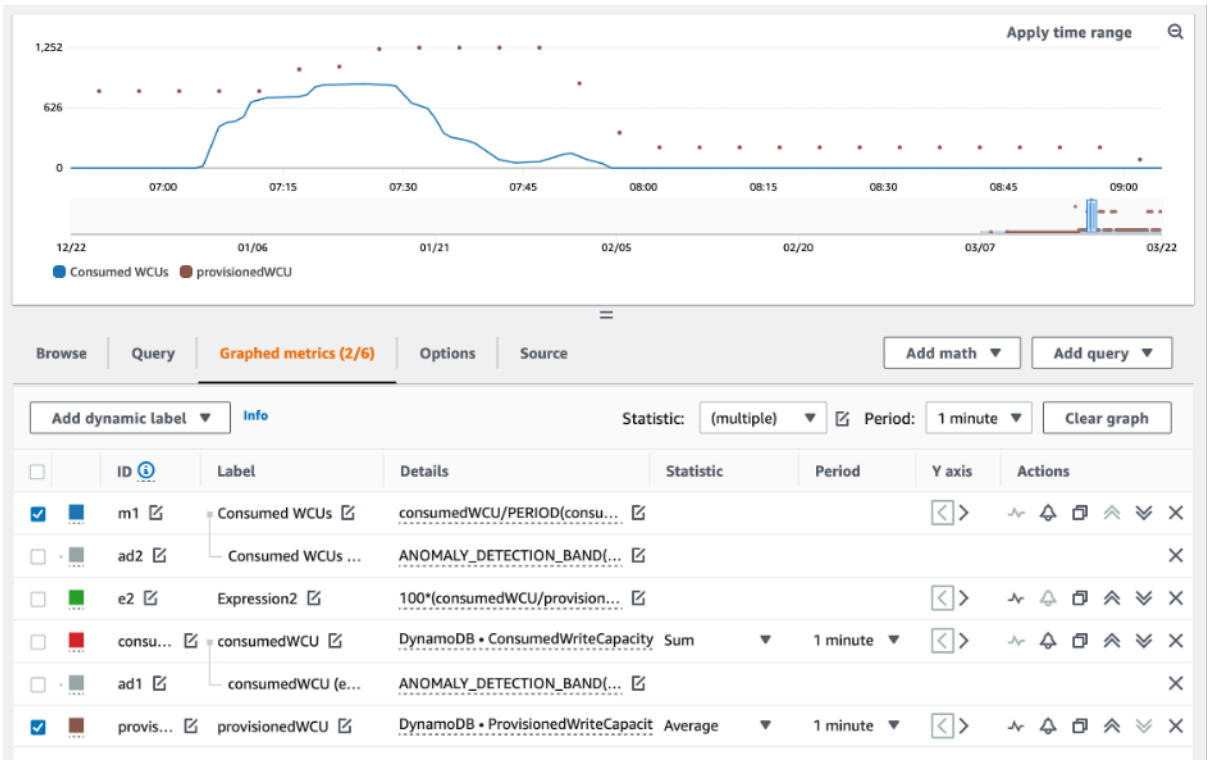



11. Expression1 레이블을 선택하고 값을 m1로 업데이트한 다음 레이블을 Consumed WCUs로 업데이트합니다.

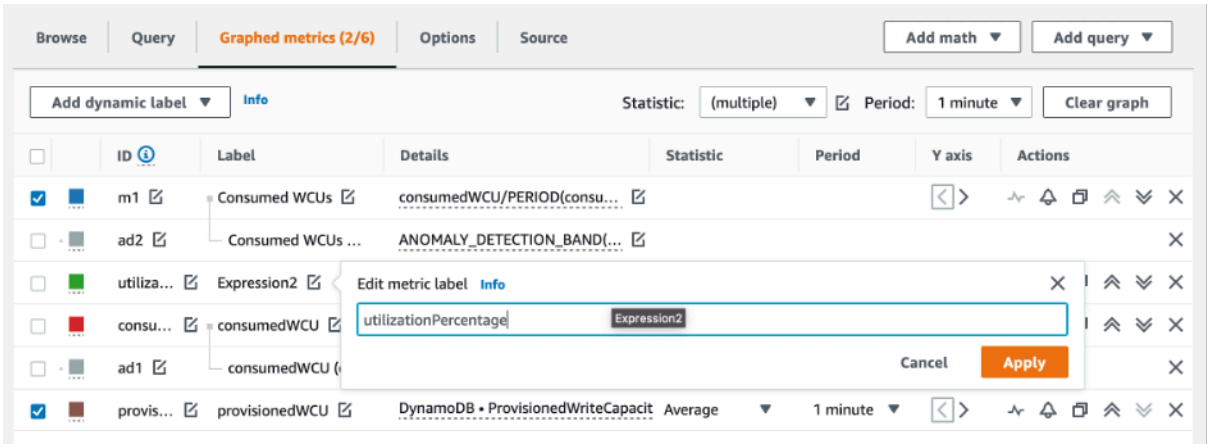
> **Note**
>
> 데이터를 제대로 시각화하려면 m1(왼쪽의 확인란) 및 provisionedWCU만 선택했는지 확인하세요. Details(세부 정보)를 클릭하고 수식을 consumedWCU/PERIOD(consumedWCU)로 변경하여 수식을 업데이트합니다. 이 단계는 또 다른 ANOMALY_DETECTION_BAND 지표를 생성할 수도 있지만 이 절차의 범위에서는 무시해도 됩니다.

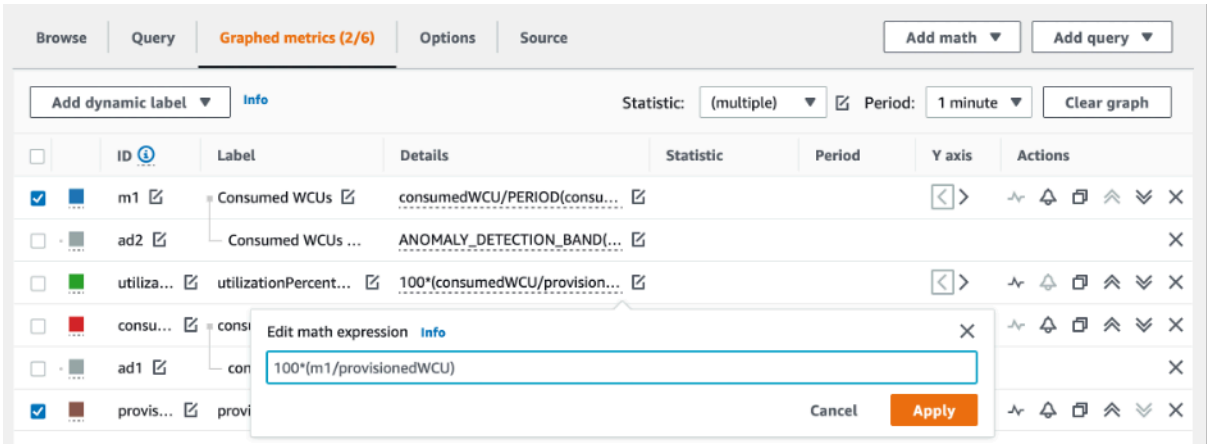



12. 이제 두 개의 그래픽이 생겼을 것입니다. 하나는 테이블의 프로비저닝된 WCU를 나타내고 다른 하나는 소비된 WCU를 나타냅니다. 그래픽의 모양은 아래와 다를 수 있지만 참조로 사용할 수 있습니다.




13. Expression2 그래픽(e2)을 선택하여 백분율 수식을 업데이트합니다. 레이블 및 ID의 이름을 utilizationPercentage로 변경합니다. 수식의 이름을 100*(m1/provisionedWCU)와 일치하도록 변경합니다.

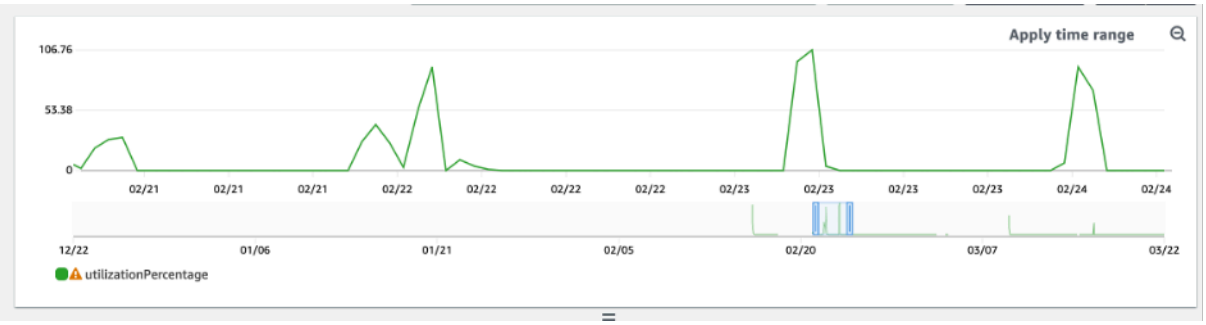



14. 사용률 패턴을 시각화하려면 utilizationPercentage를 제외한 모든 지표에서 확인란을 선택 취소하세요. 기본 간격은 1분으로 설정되어 있지만 필요에 따라 자유롭게 수정할 수 있습니다.




다음은 더 긴 시간 및 더 큰 1시간 기간의 모습입니다. 사용률이 100%보다 높은 간격이 있는 것을 볼 수 있지만, 이 특정 워크로드는 간격이 더 길고 사용률이 0입니다.




이 시점에서는 이 예제의 그림과 다른 결과를 얻을 수 있습니다. 이 모든 것은 워크로드의 데이터에 따라 달라집니다. 사용률이 100%를 초과하는 간격은 제한 이벤트가 발생하기 쉽습니다. DynamoDB는 버스트 용량을 제공하지만 버스트 용량이 완료되는 즉시 100%를 초과하는 용량은 제한됩니다.

### 과소 프로비저닝된 DynamoDB 테이블을 식별하는 방법

대부분의 워크로드에서 테이블은 프로비저닝된 용량의 80%를 초과하여 지속적으로 소비하는 경우 과소 프로비저닝된 것으로 간주됩니다.

버스트 용량은 고객이 원래 프로비저닝된 것보다 더 많은(테이블에 정의된 초당 프로비저닝된 처리량보다 많은) RCU/WCU를 일시적으로 소비할 수 있도록 하는 DynamoDB 기능입니다. 버스트 용량은 특수 이벤트 또는 사용량 급증으로 인한 갑작스러운 트래픽 증가를 흡수하기 위해 만들어졌습니다. 이 버스트 용량은 지속되지 않습니다. 사용되지 않은 RCU와 WCU가 소진되었을 때 프로비저닝된 용량보다 더 많은 용량을 소비하려고 하면 제한이 발생합니다. 애플리케이션 트래픽이 80% 사용률에 가까워지면 제한 위험이 훨씬 높아집니다.

80% 사용률 규칙은 데이터의 계절성 및 트래픽 증가에 따라 달라집니다. 다음 시나리오를 고려해 보세요.

- 지난 12개월 동안 트래픽이 약 90%의 사용률로 안정적이었다면 테이블의 용량이 적절한 것입니다.
- 애플리케이션 트래픽이 3개월 이내에 매월 8%씩 증가하면 100%에 도달하게 됩니다.
- 애플리케이션 트래픽이 4개월이 조금 넘는 기간 이내에 5%씩 증가해도 100%에 도달하게 됩니다.

위 쿼리의 결과는 사용률을 보여줍니다. 이를 참고하여 필요에 따라 테이블 용량을 늘리도록 선택하는 데 도움이 될 수 있는 다른 지표(예: 월별 또는 주별 증가율)를 추가로 평가해 보세요. 운영 팀과 협력하여 워크로드와 테이블에 적합한 비율을 정하세요.

일별 또는 주별로 데이터를 분석할 때 데이터가 왜곡되는 특수한 시나리오가 있습니다. 예를 들어 근무 시간 동안 사용량이 급증하다가 근무 시간 외에는 거의 0으로 떨어지는 계절성 애플리케이션의 경우,

프로비저닝된 용량을 늘리거나 줄일 하루 중 시간(및 요일)을 지정하는 Auto Scaling을 예약하면 효과를 볼 수 있습니다. 바쁜 시간을 처리하기 위해 더 높은 용량을 목표로 하는 대신 계절성이 덜 두드러지는 경우 DynamoDB 테이블 Auto Scaling 구성을 활용할 수도 있습니다.

> **Note**
>
> 기본 테이블에 대해 DynamoDB Auto Scaling 구성을 생성할 때는 테이블과 연결된 GSI에 대해 다른 구성을 포함해야 한다는 점을 기억하세요.

과다 프로비저닝된 DynamoDB 테이블을 식별하는 방법

위 스크립트에서 얻은 쿼리 결과는 일부 초기 분석을 수행하는 데 필요한 데이터 포인트를 제공합니다. 데이터 세트의 여러 간격에 사용률이 20% 미만인 값이 표시되면 테이블이 과다 프로비저닝된 것일 수 있습니다. WCU 및 RCU 수를 줄여야 하는지 여부를 더 자세히 정의하려면 해당 간격의 다른 측정값을 다시 살펴봐야 합니다.

테이블에 낮은 사용 간격이 여러 개 있는 경우 Auto Scaling을 예약하거나 사용률을 기반으로 테이블에 대한 기본 Auto Scaling 정책을 구성하여 Auto Scaling 정책을 사용하면 큰 이점을 얻을 수 있습니다.

워크로드에 사용률이 낮고 제한은 높은 비율(간격의 Max(ThrottleEvents)/Min(ThrottleEvents))이 있는 경우 며칠(또는 몇 시간) 동안 트래픽이 많이 증가하지만 일반적으로는 트래픽이 지속적으로 낮은 변동이 심한 워크로드일 때 이런 일이 발생할 수 있습니다. 이러한 시나리오에서는 예약된 Auto Scaling을 사용하는 것이 유용할 수 있습니다.

AWS Well-Architected 프레임워크는 클라우드 아키텍트가 다양한 애플리케이션 및 워크로드를 위한 안전하고 성능 및 복원력이 뛰어나며 효율적인 인프라를 구축할 수 있도록 지원합니다. 운영 우수성, 보안, 신뢰성, 성능 효율성, 비용 최적화 및 지속 가능성의 6가지 요소를 중심으로 구축된 AWS Well-Architected는 고객과 파트너가 아키텍처를 평가하고 확장 가능한 설계를 구현할 수 있는 일관된 접근 방식을 제공합니다.

AWS Well-Architected 렌즈는 AWS Well-Architected가 제공하는 지침을 구체적인 산업 및 기술 영역으로 확장합니다. Amazon DynamoDB Well-Architected 렌즈는 DynamoDB 워크로드에 중점을 두고 있습니다. DynamoDB 워크로드를 평가하고 검토하기 위한 모범 사례, 설계 원칙 및 질문을 제공합니다. Amazon DynamoDB Well-Architected 렌즈 검토를 완료하면 각 AWS Well-Architected 요소와 관련된 권장 설계 원칙에 대한 교육 및 지침을 얻을 수 있습니다. 이 지침은 다양한 산업, 부문, 규모 및 지역의 고객과 협력한 경험을 바탕으로 작성되었습니다.

Well-Architected 렌즈 검토의 직접적인 결과로 DynamoDB 워크로드를 최적화하고 개선하기 위한 실행 가능한 권장 사항의 요약을 받게 됩니다.

# Amazon DynamoDB Well-Architected 렌즈 검토 실시

DynamoDB Well-Architected 렌즈 검토는 일반적으로 AWS 솔루션 아키텍트가 고객과 함께 수행하지만 고객이 셀프서비스로 수행할 수도 있습니다. Amazon DynamoDB Well-Architected 렌즈의 일환으로 여섯 가지 Well-Architected 요소를 모두 검토해 보는 것이 좋지만, 우선 하나 이상의 요소에 초점을 맞출 수도 있습니다.

Amazon DynamoDB Well-Architected 렌즈 검토를 수행하기 위한 추가 정보 및 지침은 이 비디오 및 DynamoDB Well-Architected 렌즈 GitHub 페이지에서 확인할 수 있습니다.

## Amazon DynamoDB Well-Architected 렌즈의 요소

Amazon DynamoDB Well-Architected 렌즈는 다음과 같은 6가지 요소를 기반으로 구축되었습니다.

성능 효율성 요소

성능 효율성 요소에는 컴퓨팅 리소스를 시스템 요구 사항에 맞게 효율적으로 사용하고, 수요 변화 및 기술 진화에 발맞춰 그러한 효율성을 유지하는 능력이 포함됩니다.

이 요소의 기본 DynamoDB 설계 원칙은 데이터 모델링, 파티션 키 및 정렬 키 선택, 애플리케이션 액세스 패턴에 따른 보조 인덱스 정의를 중심으로 이루어집니다. 추가 고려 사항으로는 워크로드에 맞는 최적의 처리량 모드 선택, AWS SDK 조정, 해당하는 경우 최적의 캐싱 전략 사용 등이 있습니다. 이러한 설계 원칙에 대해 자세히 알아보려면 DynamoDB Well-Architected 렌즈의 성능 효율성 요소에 대한 이 심층 분석 비디오를 시청하세요.

비용 최적화 요소

비용 최적화 요소는 불필요한 비용을 피하는 데 중점을 둡니다.

주요 주제에는 비용 지출 대상 파악 및 제어, 가장 적절하고 올바른 리소스 유형 선택, 시간 경과에 따른 지출 분석, 애플리케이션별 액세스 패턴에 맞게 비용을 최적화하도록 데이터 모델 설계, 과도한 지출 없이 비즈니스 요구 사항을 충족하도록 확장하는 것이 포함됩니다.

DynamoDB의 주요 비용 최적화 설계 원칙은 테이블에 가장 적합한 용량 모드와 테이블 클래스를 선택하고 온디맨드 용량 모드를 사용하거나 AutoScaling와 함께 프로비저닝된 용량 모드를 사용하여 용량이 과도하게 프로비저닝되는 것을 방지하는 데 중점을 둡니다. 추가 고려 사항으로는 효율적인 데이터 모델링 및 쿼리를 통해 소비 용량을 줄이고, 소비된 용량의 일부를 할인된 가격으로 예약하고, 항목 크기를 최소화하고, 사용하지 않는 리소스를 식별 및 제거하고, TTL을 사용하여 노후화된 데이터를 무료로 자동 삭제하는 것이 포함됩니다. 이러한 설계 원칙에 대해 자세히 알아보려면 DynamoDB Well-Architected 렌즈의 비용 최적화 요소에 대한 이 심층 분석 비디오를 시청하세요.

DynamoDB의 비용 최적화 모범 사례에 대한 추가 정보는 비용 최적화를 참조하세요.

### 운영 우수성 요소

운영 우수성 요소는 비즈니스 가치를 제공하고 프로세스 및 절차를 지속적으로 개선하기 위한 시스템 운영 및 모니터링에 중점을 둡니다. 주요 주제에는 변경 자동화, 이벤트 대응, 일상 운영 관리를 위한 표준 정의 등이 포함됩니다.

DynamoDB의 주요 운영 우수성 설계 원칙에는 Amazon CloudWatch 및 AWS Config를 통해 DynamoDB 지표를 모니터링하고, 사전 정의된 임계값이 위반되거나 규정을 준수하지 않는 규칙이 감지되면 자동으로 경고하고 해결하는 것이 포함됩니다. 추가 고려 사항으로는 인프라를 통한 DynamoDB 리소스를 코드로 정의하고, DynamoDB 리소스의 더 나은 구성, 식별 및 비용 계산을 위해 태그를 활용하는 것이 있습니다. 이러한 설계 원칙에 대해 자세히 알아보려면 DynamoDB Well-Architected 렌즈의 운영 우수성 요소에 대한 이 심층 분석 비디오를 시청하세요.

### 신뢰성 요소

신뢰성 요소는 워크로드가 필요한 시점에 의도된 기능을 정확하고 일관되게 수행하도록 하는 데 중점을 둡니다. 복원력이 뛰어난 워크로드는 장애 발생 시 신속하게 복구하여 비즈니스 및 고객 요구를 충족합니다. 주요 주제에는 분산 시스템 설계, 복구 계획 및 변경 처리 방법이 포함됩니다.

DynamoDB의 필수 신뢰성 설계 원칙은 RPO 및 RTO 요구 사항을 기반으로 백업 전략 및 보존을 선택하거나, 다중 리전 워크로드 또는 RTO가 낮은 크로스 리전 재해 복구 시나리오를 위해 DynamoDB 글로벌 테이블을 사용하고, AWS SDK의 이러한 기능을 구성 및 사용하여 애플리케이션에 지수 백오프가 포함된 재시도 로직을 구현하고, Amazon CloudWatch를 통해 DynamoDB 지표를 모니터링하고, 사전 정의된 임계값이 초과된 경우 자동으로 알림 및 문제 해결을 수행하는 것을 중점으로 합니다. 이러한 설계 원칙에 대해 자세히 알아보려면 DynamoDB Well-Architected 렌즈의 신뢰성 요소에 대한 이 심층 분석 비디오를 시청하세요.

### 보안 요소

보안 요소는 정보 및 시스템 보호에 중점을 둡니다. 주요 주제에는 데이터의 기밀성 및 무결성, 권한 관리를 통해 누가 무엇을 할 수 있는지 식별 및 관리, 시스템 보호, 보안 이벤트 탐지를 위한 제어 설정 등이 포함됩니다.

DynamoDB의 주요 보안 설계 원칙은 HTTPS로 전송 중 데이터를 암호화하고, 저장 데이터 암호화를 위한 키 유형을 선택하고, DynamoDB 리소스에 대한 세부 액세스를 인증, 승인 및 제공하기 위한 IAM 역할 및 정책을 정의하는 것입니다. 추가 고려 사항으로는 AWS CloudTrail을 통한 DynamoDB 컨트롤 플레인 및 데이터 영역 작업의 감사가 포함됩니다. 이러한 설계 원칙에 대해 자세히 알아보려면 DynamoDB Well-Architected 렌즈의 보안 요소에 대한 이 심층 분석 비디오를 시청하세요.

DynamoDB 보안에 대한 추가 정보는 보안을 참조하세요.

지속 가능성 요소

지속 가능성 요소는 클라우드 워크로드 실행이 환경에 미치는 영향을 최소화하는 데 중점을 둡니다. 주요 주제에는 지속 가능성에 대한 공동 책임 모델, 영향 이해, 활용률 극대화로 필요한 리소스를 최소화하고 다운스트림 영향을 줄이는 것이 포함됩니다.

DynamoDB의 주요 지속 가능성 설계 원칙에는 미사용 DynamoDB 리소스 식별 및 제거, AutoScaling이 포함된 온디맨드 용량 모드 또는 프로비저닝된 용량 모드 사용을 통한 과다 프로비저닝 방지, 소비되는 용량을 줄이기 위한 효율적인 쿼리, 데이터를 압축하고 TTL을 통해 노후화된 데이터를 삭제하여 스토리지 점유 공간을 줄이는 것이 포함됩니다. 이러한 설계 원칙에 대해 자세히 알아보려면 DynamoDB Well-Architected 렌즈의 지속 가능성 요소에 대한 이 심층 분석 비디오를 시청하세요.

# DynamoDB에서 효과적으로 파티션 키를 설계해 사용하는 모범 사례입니다.

Amazon DynamoDB 테이블에서 각 항목을 고유하게 식별하는 기본 키는 간단하거나(파티션 키만) 복합적일(정렬 키가 통합된 파티션 키) 수 있습니다.

테이블과 보조 인덱스의 모든 파티션 키에서 균일하게 활동을 하도록 애플리케이션을 설계해야 합니다. 애플리케이션에 필요한 액세스 패턴을 결정하고, 각 테이블 및 보조 인덱스에 필요한 읽기 및 쓰기 유닛을 결정할 수 있습니다.

> **Note**
>
> 조정 용량은 온디맨드 모드 및 프로비저닝된 용량에 적용됩니다.

기본적으로 DynamoDB 테이블의 모든 파티션은 초당 3,000 읽기 단위 및 초당 1,000 쓰기 단위의 최대 용량을 제공하도록 설계되었습니다. 읽기 단위 1은 초당 강력히 일관된 읽기 1 또는 초당 최종적으로 일관된 읽기 2(최대 4KB 크기 항목의 경우)를 나타냅니다. 쓰기 단위 1은 최대 1KB 크기의 항목에 대해 초당 1회 쓰기 작업을 나타냅니다.

테이블의 파티션 처리량 한도를 평가할 때는 항목 크기를 고려해야 합니다. 예를 들어 테이블의 항목 크기가 20KB인 경우 일관된 단일 읽기 작업에는 읽기 단위 5가 사용됩니다. 즉, 파티션 한도에 도달하기 전에 해당 단일 항목에 대해 초당 600회의 일관된 읽기 작업을 동시에 실행할 수 있습니다. 테이블의 모든 파티션에 걸친 총 처리량은 프로비저닝된 모드의 프로비저닝된 처리량 또는 온디맨드 모드의 테이블 수준 처리량 한도에 의해 제한될 수 있습니다. 자세한 내용은 Service Quotas를 참조하세요.

주제


- DynamoDB에 워크로드가 배포되도록 파티션 키 설계
- 쓰기 샤딩을 사용해 DynamoDB 테이블에 워크로드를 고르게 배포
- DynamoDB에 데이터 업로드 중 효율적으로 쓰기 활동 배포


## DynamoDB에 워크로드가 배포되도록 파티션 키 설계

테이블 기본 키의 파티션 키 부분은 테이블의 데이터가 저장되는 논리적 파티션을 결정합니다. 이는 기본 물리적 파티션에 영향을 줍니다. I/O 요청을 효과적으로 분산시키지 않는 파티션 키 설계는 '핫' 파티션을 발생시킬 수 있으며, 이는 제한을 발생시키고 프로비저닝된 I/O 용량을 비효율적으로 사용하게 되는 문제를 초래합니다.

테이블의 프로비저닝된 처리량의 최적 사용량은 개별 항목의 워크로드 패턴과 파티션 키 설계가 결정합니다. 이는 모든 파티션 키 값에 액세스하여 효율적인 처리량 수준을 달성해야 한다는 의미는 아닙니다. 또한 액세스된 파티션 키 값의 백분율이 높아야 한다는 의미도 아닙니다. 워크로드가 액세스 하는 고유 파티션 키 값이 많을 수록 요청이 여러 파티션 공간으로 더 많이 분산된다는 의미입니다. 일반적으로 총 파티션 키 값에 액세스한 파티션 키 값의 비율이 증가할수록 처리량을 보다 효율적으로 활용할 수 있습니다.

다음은 몇몇 범용 파티션 키 스키마의 프로비저닝된 처리량 효율성을 비교한 내용입니다.

| 파티션 키 값 | 균일성 |
| --- | --- |
| 사용자 ID, 애플리케이션의 사용자가 많은 경우. | 좋음 |
| 상태 코드, 가능한 상태 코드가 몇 개 없는 경우. | 나쁨 |
| 항목 생성 날짜, 가장 가까운 시간(예: 날, 시, 분)으로 반올림. | 나쁨 |
| 디바이스 ID, 각 디바이스가 비교적 비슷한 간격으로 데이터에 액세스하는 경우. | 좋음 |
| 디바이스 ID, 추적되는 디바이스는 많지만 다른 디바이스보다 한 디바이스가 훨씬 더 인기 있는 경우. | 나쁨 |

단일 테이블에 파티션 키 값의 수가 매우 적은 경우, 쓰기 작업을 더 많은 고유 파티션 값에 배포하는 것이 좋습니다. 다시 말해 전반적인 성능 저하를 야기하는 하나의 '핫'한(자주 요청되는) 파티션 키 값이 발생하지 않도록 기본 키 요소를 구성합니다.

복합 기본 키가 하나만 있는 테이블을 예로 들어 보겠습니다. 파티션 키는 항목의 생성 날짜를 나타내며 가장 가까운 날로 반올림됩니다. 정렬 키는 항목 식별자입니다. 지정된 날, 예를 들면 `2014-07-09`에 모든 새 항목이 동일한 파티션-키 값(및 상응하는 물리적 파티션)에 작성됩니다.

테이블이 단일 파티션에 꼭 맞으며(시간 경과에 따른 데이터 증가 고려) 애플리케이션의 읽기 및 쓰기 처리량 요구 사항이 단일 파티션의 읽기 및 쓰기 능력을 초과하지 않으면, 애플리케이션에 파티셔닝으로 인한 예상치 않은 제한(병목) 현상이 발생하지 않습니다.

DynamoDB용 NoSQL Workbench를 사용하여 파티션 키 설계를 시각화하려면 NoSQL Workbench로 데이터 모델 빌드 섹션을 참조하세요.

## 쓰기 샤딩을 사용해 DynamoDB 테이블에 워크로드를 고르게 배포

Amazon DynamoDB의 파티션 키 공간에 더 효과적으로 쓰기 작업을 배포하는 방법 중 하나는 공간 확장입니다. 여러 방법을 사용할 수 있습니다. 파티션 키 값에 난수를 추가하여 여러 파티션으로 항목을 배포할 수 있습니다. 또는 쿼리하는 항목을 기반으로 계산된 숫자를 사용할 수 있습니다.

### 임의의 접미사를 사용하는 샤딩

여러 파티션 키 공간에 로드를 더 골고루 배포할 수 있는 전략 중 하나는 파티션 키 값 끝에 난수(임의의 수)를 추가하는 것입니다. 그러면 더 큰 공간으로 쓰기를 무작위화 할 수 있습니다.

예를 들어, 오늘 날짜를 나타내는 파티션 키의 경우, `1`부터 `200` 사이의 난수를 선택하고, 날짜의 접미사로 연결시킬 수 있습니다. 이는 `2014-07-09.1`, `2014-07-09.2`에서 `2014-07-09.200`까지의 파티션 키 값을 생성합니다. 파티션 키를 임의 지정했기 때문에, 각 날짜의 테이블에 대한 쓰기가 모든 파티션 키 값에 고르게 분산됩니다. 이는 병렬 처리를 개선하고 전반적인 처리량을 높입니다.

하지만 지정된 날짜의 모든 항목을 읽으려면, 모든 접미사의 항목을 쿼리해 결과를 병합해야 합니다. 예를 들어 먼저 파티션 키 값 `2014-07-09.1`에 대한 `Query` 요청을 생성합니다. 그런 다음 `2014-07-09.2`에서 `2014-07-09.200`까지에 대해 또 다른 `Query` 요청을 생성합니다. 마지막으로 애플리케이션은 모든 `Query` 요청 결과를 병합해야 합니다.

### 계산한 접미사를 사용하는 샤딩

임의 지정 전략으로 쓰기 처리량을 크게 향상시킬 수 있습니다. 그러나 특정 항목을 쓸 때 어떤 접미사 값이 사용되었는지 모르기 때문에 해당 항목을 읽기 어렵습니다. 개별 항목을 더 쉽게 읽을 수 있도록

만들려면 다른 전략을 사용합니다. 난수(임의의 수)를 사용해 여러 파티션으로 항목을 배포하는 대신, 쿼리하고 싶은 내용을 토대로 계산할 수 있는 수를 사용합니다.

테이블의 파티션 키에 오늘 날짜를 사용한 앞의 예를 가정하겠습니다. 이제 각 항목에 액세스가 가능한 `OrderId` 속성이 있고, 날짜에 더해 주문 ID로 항목을 찾아야 하는 경우가 많다고 가정하겠습니다. 애플리케이션이 테이블에 항목을 쓰기 전에 주문 ID를 기준으로 해시 접미사를 계산하고 파티션 키 날짜에 추가할 수 있습니다. 계산 결과 임의 지정 전략에서 생산되는 결과와 유사하게 골고루 배포된 1-200 사이의 숫자가 생성될 것입니다.

주문 ID의 문자에 대한 UTF-8 코드 포인트 값의 제품이나 모듈로 200 + 1과 같이 간단한 계산으로 충분합니다. 파티션 키 값은 계산 결과와 연결된 날짜가 됩니다.

이러한 전략을 통해 쓰기가 파티션-키 값 및 물리적 파티션에 고르게 분산됩니다. 특정 항목과 날짜에서 `GetItem` 작업을 손쉽게 수행할 수 있습니다. 특정 `OrderId` 값에 대한 파티션-키 값을 계산할 수 있기 때문입니다.

지정된 날의 모든 항목을 읽으려면 각 `2014-07-09.N`(여기서 `N`은 1~200) 키를 `Query`해야 하며, 애플리케이션은 모든 결과를 병합해야 합니다. 장점은 모든 워크로드를 취하는 하나의 '핫' 파티션 키 값이 발생하지 않도록 만든다는 것입니다.

> **Note**
>
> 볼륨이 많은 시계열 데이터를 더 효율적으로 처리할 수 있는 전략에 대한 자세한 내용은 시계열 데이터 섹션을 참조하세요.

## DynamoDB에 데이터 업로드 중 효율적으로 쓰기 활동 배포

일반적으로 다른 데이터 소스에서 데이터를 로드할 때, Amazon DynamoDB는 여러 서버의 테이블 데이터를 파티션합니다. 테이블에 데이터를 업로드할 때 할당된 모든 서버에 데이터를 동시에 업로드하면 성능이 향상됩니다.

예를 들어, `UserID`를 파티션 키로, `MessageID`를 정렬 키로 사용하는 복합 기본 키를 사용하는 DynamoDB 테이블로 사용자 메시지를 업로드하는 경우를 가정합니다.

데이터를 업로드하는 방법 중 하나는 각 사용자 별로 한 명씩 모든 메시지 항목을 업로드하는 것입니다.

| UserID | MessageID |
| --- | --- |
| U1 | 1 |
| U1 | 2 |
| U1 | ... |
| U1 | ... 최대 100 |
| U2 | 1 |
| U2 | 2 |
| U2 | ... |
| U2 | ... 최대 200 |

이 경우 문제는 DynamoDB에 대한 쓰기 요청을 파티션 키 값에 분산하지 못한다는 것입니다. 한 번에 파티션 키 값 하나를 취하고 모든 항목을 업로드한 후 다음 파티션 값에서 동일한 작업을 반복합니다.

표시되지는 않지만 DynamoDB는 테이블의 데이터를 여러 서버에 분할합니다. 테이블에 대해 프로비저닝된 모든 처리량 용량을 완전히 활용하려면 워크로드를 파티션 키 값에 분산해야 합니다. 이 경우 불균일한 업로드 작업 양을 파티션 키 값이 모두 동일한 항목으로 전달하면 DynamoDB가 테이블에 대해 프로비저닝한 모든 리소스를 완전히 활용하지 못할 수 있습니다.

정렬 키를 사용하여 업로드 작업을 배포, 각 파티션 키 값으로부터 하나 씩 항목을 로드할 수 있습니다.

| UserID | MessageID |
| --- | --- |
| U1 | 1 |
| U2 | 1 |
| U3 | 1 |
| ... | ... |
| U1 | 2 |

| UserID | MessageID |
| --- | --- |
| U2 | 2 |
| U3 | 2 |
| ... | ... |

이 시퀀스에서 모든 업로드는 서로 다른 파티션 키 값을 사용하므로, 더 많은 DynamoDB 서버를 동시에 사용하도록 하여 처리량 성능을 향상합니다.

# 정렬 키를 사용하여 DynamoDB의 데이터를 정리하는 모범 사례

Amazon DynamoDB 테이블에서 테이블의 각 항목을 고유하게 식별하는 프라이머리 키는 파티션 키와 정렬 키로 구성할 수 있습니다.

잘 설계된 정렬 키에는 중요한 장점이 두 가지 있습니다.

- 효율적으로 쿼리를 할 수 있는 단일 장소로 관련 정보를 수집합니다. 신중하게 정렬 키를 설계하면 `begins_with`, `between`, `>`, `<` 등 연산자를 사용하는 범위 쿼리를 사용하여 관련 항목의 필요한 그룹을 검색할 수 있습니다.
- 복합 정렬 키는 계층 구조의 어느 수준에서 쿼리를 할 수 있도록 데이터의 계층적(일대다) 관계를 정의할 수 있도록 도와줍니다.

  예를 들어, 지리적 위치를 나열한 테이블에서 다음과 같이 정렬 키를 구성할 수 있습니다.

  ```
  [country]#[region]#[state]#[county]#[city]#[neighborhood]
  ```

  이렇게 하면 `country`부터 `neighborhood`까지, 그리고 그 사이의 집계 수준에서 위치 목록을 효율적으로 범위 쿼리할 수 있습니다.

## 버전 제어에 정렬 키 사용

감사나 규정 준수 목적에서 항목 수준의 수정 이력을 유지하고, 쉽게 가장 최근 버전을 검색할 수 있어야 하는 애플리케이션이 많습니다. 정렬 키 접두사를 사용하여 이를 쉽게 달성할 수 있는 효과적인 설계 패턴이 있습니다.

- 각 새 항목에 대해 항복 복사본 두 개를 생성합니다. 첫 번째 복사본에서 정렬 키 시작 지점의 버전 번호 접두사는 0(v0_), 두 번째 복사본의 버전 번호 접두사는 1(v1_)이여야 합니다.
- 항목을 업데이트할 때마다 업데이트 버전의 정렬 키에 다음으로 높은 버전의 접두사를 사용하고, 업데이트한 내용을 버전 접두사가 0인 항목으로 복사합니다. 그러면 0이라는 접두사를 사용해 항목의 최신 버전이 위치한 장소를 쉽게 찾을 수 있습니다.

예를 들어, 부품 제조업체가 아래 설명한 것과 같은 스키마를 사용한다고 가정하겠습니다.

| Primary Key | | Data-Item Attributes... | | | | |
|---|---|---|---|---|---|---|
| Partition Key<br>*Equipment_ID* | Sort Key<br>*(varies)* | Attribute 1 | Attribute 2 | Attribute 3 | Attribute 4 | ... |
| Equipment_1 | Details | Name: Biphasic Cardiometer<br>*(equipment name)* | Factory_ID: S14_Tukwilla<br>*(factory where manufactured)* | Line_ID R_7<br>*(assembly-line ID)* | | |
| | v0_Audit | Auditor: Padma<br>*(name of the auditor)* | Latest: 3<br>*(most recent audit version)* | Time: 2018-04-15T11:00<br>*(audit date and time)* | Result Passed<br>*(audit result)* | ...etc. |
| | v1_Audit | Auditor: Rick<br>*(name of the auditor)* | Time: 2018-03-14T11:00<br>*(audit date and time)* | Result Open<br>*(audit result)* | Report: 0943922EKG14<br>*(detailed problem report in S3)* | ...etc. |
| | v2_Audit | Auditor: George<br>*(name of the auditor)* | Time: 2018-03-18T11:00<br>*(audit date and time)* | Result Open<br>*(audit result)* | Report: 0943923EKG15<br>*(detailed problem report in S3)* | ...etc. |
| | v3_Audit | Auditor: Padma<br>*(name of the auditor)* | Time: 2018-04-15T11:00<br>*(audit date and time)* | Result Passed<br>*(audit result)* | Report: x792<br>*(pass confirmation report)* | ...etc. |

`Equipment_1` 항목은 여러 감사자의 감사 시퀀스를 거칩니다. 새로운 감사 결과가 테이블의 새 항목으로 캡처됩니다. 시작 버전 번호는 1이며, 이후 연속된 각 개정 버전에 대해 번호를 증분시킵니다.

각 새 개정 버전을 추가할 때, 애플리케이션 계층은 0 버전 항목(정렬 키가 `v0_Audit`)의 내용을 새 개정 버전의 내용으로 대체합니다.

애플리케이션이 가장 최근 감사 상태를 검색해야 할 때마다 접두사가 `v0_`인 정렬 키를 쿼리할 수 있습니다.

애플리케이션이 전체 개정 이력을 검색해야 한다면, 항목의 파티션 키 아래 모든 항목을 쿼리해 `v0_` 항목을 필터링 할 수 있습니다.

이런 설계는 특정 장치나 장비의 여러 부품에 대한 감사에도 효과가 있습니다. 정렬 키에서 정렬 키 접두사 다음에 개별 부품의 ID를 포함시키는 방법을 사용합니다.

## DynamoDB의 보조 인덱스 사용에 대한 모범 사례

사용자의 애플리케이션이 요구하는 쿼리 패턴을 지원하기 위해 반드시 보조 인덱스가 필요한 경우가 있습니다. 한편으로는 보조 인덱스를 과도하게 사용하거나 비효율적으로 사용하면 비용이 추가되고, 불필요하게 성능이 저하될 수 있습니다.

목차


- DynamoDB의 보조 인덱스 사용에 대한 일반 지침
  - 효율적으로 인덱스 사용
  - 신중하게 프로젝션 선택
  - 빈번한 쿼리를 최적화하여 페치 방지
  - 로컬 보조 인덱스를 생성할 때 항목 모음의 크기 제한 유의
- 희소 인덱스 활용
  - DynamoDB의 희소 인덱스 예
- DynamoDB의 구체화된 집계 쿼리에 글로벌 보조 인덱스 사용
- DynamoDB의 글로벌 보조 인덱스 오버로딩
- DynamoDB의 일부 테이블 쿼리에 글로벌 보조 인덱스 쓰기 샤딩 사용
  - 패턴 설계
  - 샤딩 전략
  - 샤딩된 GSI 쿼리
  - 병렬 쿼리 실행 고려 사항
  - 코드 예제
- 글로벌 보조 인덱스를 사용하여 DynamoDB에 최종적으로 일관된 복제본 생성


## DynamoDB의 보조 인덱스 사용에 대한 일반 지침

Amazon DynamoDB는 두 종류의 보조 인덱스를 지원합니다.

- 글로벌 보조 인덱스(GSI) - 기본 테이블의 파티션 키 및 정렬 키와 다른 파티션 키와 정렬 키가 있는 인덱스입니다. 모든 파티션에서 인덱스의 쿼리가 기본 테이블의 모든 데이터에 적용될 수 있으므로 글로벌 보조 인덱스는 글로벌하게 간주됩니다. 글로벌 보조 인덱스에는 크기 제한이 없고, 테이블과 다른 읽기 및 쓰기 작업에 대한 프로비저닝된 처리량 설정 값을 가지고 있습니다.
- 로컬 보조 인덱스(LSI) - 기본 테이블과 동일한 파티션 키가 있지만, 정렬 키가 다른 인덱스입니다. 로컬 보조 인덱스는 로컬 보조 인덱스의 모든 파티션이 동일한 파티션 키 값을 갖는 기본 테이블 파티션으로 한정된다는 의미에서 "로컬"이라고 합니다. 따라서 특정 파티션 키 값에 대해 인덱스된 항목의 총 크기가 10GB를 초과할 수 없습니다. 또한 로컬 보조 인덱스는 인덱싱한 테이블과 쓰기 및 읽기 작업에 대해 프로비저닝된 처리량 설정을 공유합니다.

DynamoDB의 각 테이블은 최대 20개의 글로벌 보조 인덱스(기본 할당량)와 5개의 로컬 보조 인덱스를 가질 수 있습니다.

글로벌 보조 인덱스는 로컬 보조 인덱스보다 더 유용한 경우가 많습니다. 사용할 인덱스 유형을 결정하는 것도 애플리케이션의 요구 사항에 따라 달라집니다. 글로벌 보조 인덱스와 로컬 보조 인덱스의 비교 및 선택 방법에 대한 자세한 내용은 the section called “인덱스 작업” 섹션을 참조하세요.

다음은 DynamoDB에 인덱스를 생성할 때 유의해야 할 몇 가지 일반 원칙과 설계 패턴입니다.

주제


- 효율적으로 인덱스 사용
- 신중하게 프로젝션 선택
- 빈번한 쿼리를 최적화하여 페치 방지
- 로컬 보조 인덱스를 생성할 때 항목 모음의 크기 제한 유의


## 효율적으로 인덱스 사용

인덱스의 수를 최소한으로 유지합니다. 자주 쿼리하지 않는 속성에서는 보조 인덱스를 생성하지 않습니다. 거의 사용하지 않는 인덱스는 애플리케이션 성능 향상에 도움을 주지 못하며, 스토리지 및 I/O 비용을 높입니다.

## 신중하게 프로젝션 선택

보조 인덱스는 스토리지 및 프로비저닝된 처리량을 이용하므로 인덱스를 가능한 한 작은 크기로 유지해야 합니다. 또한 인덱스 크기가 작을수록 전체 테이블에서 쿼리하는 데 비해 성능면에서 훨씬 큰 이점이 있습니다. 일반적으로 쿼리에서 속성의 작은 하위 집합만 반환하고 해당 속성의 총 크기가 전체 항목보다 훨씬 작은 경우, 정기적으로 요청할 속성만 프로젝션하세요.

테이블에서 쓰기 작업이 읽기에 비해 많을 것으로 예상되는 경우, 다음 모범 사례를 적용합니다.

- 적은 수의 속성을 프로젝션하여 인덱스에 쓸 항목 크기를 최소화하도록 고려하세요. 그러나 프로젝션된 속성의 크기가 단일 쓰기 용량 유닛(1KB)보다 큰 경우에만 적용됩니다. 예를 들어 인덱스 항목 크기가 200바이트인 경우, DynamoDB가 항목 크기를 1KB로 반올림합니다. 즉, 인덱스 항목이 작으면 추가 비용 없이 더 많은 속성을 프로젝션할 수 있습니다.
- 쿼리에 거의 필요하지 않은 프로젝센된 속성을 피합니다. 인덱스에 프로젝션된 속성을 업데이트할 때마다 인덱스 업데이트에 추가 비용이 발생합니다. `Query`에서 프로젝션하지 않은 속성을 검색하는 경우 프로비저닝된 처리량 비용이 증가하지만, 쿼리 비용이 자주 인덱스를 업데이트하는 비용보다 훨씬 더 낮습니다.

- 쿼리에서 전체 테이블 항목을 반환하지만 다른 정렬 키로 테이블을 정렬하는 경우에만 ALL을 지정하세요. 모든 속성을 프로젝션하면 테이블 페치가 필요 없지만, 저장과 쓰기 작업 비용이 두 배 증가합니다.

다음 섹션에서 설명하겠지만, 페치를 최소한으로 유지하는 동시에 인덱스를 가능한 작게 유지해야 합니다.

### 빈번한 쿼리를 최적화하여 페치 방지

가장 적은 지연 시간으로 가장 빠른 쿼리를 가져오려면 해당 쿼리가 반환할 것으로 예상되는 모든 속성을 프로젝션하세요. 특히 프로젝션되지 않은 속성에 대해 로컬 보조 인덱스를 쿼리하면, DynamoDB가 자동으로 테이블에서 이런 속성을 가져오는데, 이 경우 테이블에서 전체 항목을 읽어야 합니다. 이렇게 하면 지연 시간과 추가적인 I/O 작업 발생을 피할 수 없습니다.

종종 '간헐적' 쿼리가 '필수' 쿼리로 변한다는 점을 유의합시오. 간헐적으로만 쿼리할 것으로 예상하기 때문에 프로젝션을 하지 않는 속성이 있다면, 상황이 바뀔지 여부, 속성을 프로젝션하지 않은 것을 후회하게 될지 여부를 고려해야 합니다.

테이블 가져오기에 대한 자세한 내용은 로컬 보조 인덱스에 대해 프로비저닝된 처리량 고려 사항 섹션을 참조하세요.

### 로컬 보조 인덱스를 생성할 때 항목 모음의 크기 제한 유의

항목 모음은 파티션 키가 동일한 테이블과 로컬 보조 인덱스의 모든 항목입니다. 항목 모음은 10GB를 초과할 수 없으므로 특정 파티션 키 값에 대한 공간이 부족할 수 있습니다.

테이블 항목을 추가하거나 업데이트하는 경우 DynamoDB는 영향을 받는 모든 로컬 보조 인덱스도 업데이트합니다. 인덱싱된 속성이 테이블에 정의되면 로컬 보조 인덱스 또한 늘어납니다.

로컬 보조 인덱스를 생성할 때, 여기에 기록할 데이터의 양과 동일한 파티션 키 값을 갖게 될 데이터 항목의 수를 고려하세요. 특정 파티션 키 값에 대한 테이블과 인덱스 항목의 합이 10GB를 초과할 것으로 예상되면 인덱스를 생성하지 않아야 하는지를 고려하세요.

로컬 보조 인덱스를 반드시 생성해야 한다면 항목 모음 크기 제한을 예상하여 초과되기 전에 조치를 취해야 합니다. 가장 좋은 방법은 항목을 작성할 때 `ReturnItemCollectionMetrics` 파라미터를 활용하여 10GB 크기 한도에 근접하는 항목 모음 크기를 모니터링하고 이에 대해 경고하는 것입니다. 최대 항목 모음 크기를 초과하면 쓰기 시도가 실패합니다. 애플리케이션에 영향을 미치기 전에 항목 모음 크기를 모니터링하고 경고하여 항목 모음 크기 문제를 완화할 수 있습니다.

> **Note**
>
> 로컬 보조 인덱스를 생성한 후에는 삭제할 수 없습니다.

제한값 내에서 작업 및 교정 작업에 대한 전략은 항목 컬렉션 크기 제한의 내용을 참조하세요.

## 희소 인덱스 활용

테이블의 모든 항목에 대해 DynamoDB는 인덱스 정렬 키 값이 항목에 있는 경우에만 해당 인덱스 항목을 기록합니다. 정렬 키가 모든 테이블 항목에 표시되지 않거나 인덱스 파티션 키가 항목에 없는 경우 인덱스가 희소하다고 말합니다.

희소한 인덱스는 테이블의 작은 하위 항목에 대한 쿼리에 유용합니다. 예를 들어, 다음의 키 속성을 가지고 있는 고객 주문을 모두 저장한 테이블이 있다고 가정하겠습니다.

- 파티션 키: `CustomerId`
- Sort key: `OrderId`

열려 있는 주문을 추적할 경우, 아직 발송하지 않은 주문 항목에 `isOpen`이라는 속성을 삽입할 수 있습니다. 그리고 주문을 발송한 후에 이 속성을 삭제할 수 있습니다. 그리고 `CustomerId`(파티션 키) 및 `isOpen`(정렬 키)에 인덱스를 생성하면, `isOpen` 이 정의된 주문만 여기에 표시됩니다. 수천 가지의 주문이 있지만 열려 있는 주문이 적은 경우, 전체 테이블을 스캔하는 대신 열려 있는 주문의 인덱스에 쿼리를 하는 것이 훨씬 빠르고 경제적입니다.

`isOpen` 같은 속성 유형을 사용하는 대신, 인덱스에서 유용하게 주문을 분류하는 값을 가진 속성을 사용할 수도 있습니다. 예를 들어, 각 주문 날짜에 대한 `OrderOpenDate` 속성을 사용한 후, 주문을 처리한 후에 이를 삭제할 수도 있습니다. 이는 희소 인덱스를 쿼리할 때, 주문한 날짜를 기준으로 항목을 분류해 반환하는 방식입니다.

### DynamoDB의 희소 인덱스 예

글로벌 보조 인덱스는 기본값이 '희소'입니다. 글로벌 보조 인덱스를 생성할 때는 파티션 키를 지정해야 합니다(선택적으로 정렬 키 지정). 이 속성을 가진 기본 테이블의 항목만 인덱스에 표시됩니다.

글로벌 보조 인덱스를 희소 인덱스로 설계해서 프로비저닝을 하면, 기본 테이블보다 쓰기 처리량을 줄이고, 성능을 높일 수 있습니다.

예를 들어, 게임 애플리케이션이 모든 사용자의 점수를 추적하지만, 점수가 높은 소수만 쿼리할 필요가 있다고 가정하겠습니다. 다음 설계가 이런 시나리오를 효율적으로 처리합니다.

Table

| Primary Key | | Data Attributes... | | |
|---|---|---|---|---|
| *Partition Key* | *Sort Key* | | | |
| Player_ID | Game_ID | Attribute 1 | Attribute 2 | Attribute 3 |
| Rick | Game_1 | Score: 36,750 *(game score)* | Date: 2017-11-14 *(date of game)* | |
| | Game_2 | Score: 69,450 *(game score)* | Date: 2017-12-31 *(date of game)* | |
| | Game_3 | Score: 135,900 *(game score)* | Date: 2018-01-19 *(date of game)* | Award: Champ *(type of award)* |
| Padma | Game_4 | Score: 25,350 *(game score)* | Date: 2018-01-27 *(date of game)* | |
| | Game_5 | Score: 69,450 *(game score)* | Date: 2028-01-19 *(date of game)* | |
| | Game_6 | Score: 147,300 *(game score)* | Date: 2018-02-02 *(date of game)* | Award: Champ *(type of award)* |
| | Game_7 | Score: 169,100 *(game score)* | Date: 2018-03-10 *(date of game)* | Award: Champ *(type of award)* |

Rick이 3가지 게임을 플레이했는 데, 이 중 하나에서 Champ 상태를 달성했다고 가정하겠습니다. Padama는 4가지 게임을 플레이했는 데, 이 중 2개에서 Champ 상태를 달성했습니다. Award 속성은 사용자가 성과(Award)를 달성한 항목에만 표시됩니다. 이과 관련된 글로벌 보조 인덱스는 다음과 같습니다.

GSI

| Primary Key | Projected Attributes... | | | |
|---|---|---|---|---|
| *Partition Key* | | | | |
| Award | Player_ID | Game_ID | Score | Date |
| Champ | Rick | Game_3 | 135,900 | 2018-01-19 |
| | Padma | Game_6 | 147,300 | 2018-02-02 |
| | Padma | Game_7 | 169,100 | 2018-03-10 |

글로벌 보조 인덱스에는 기본 테이블의 항목 중 소수 하위 집합으로 자주 쿼리되는 높은 점수만 포함되어 있습니다.

## DynamoDB의 구체화된 집계 쿼리에 글로벌 보조 인덱스 사용

빠르게 변화하는 데이터에 대해 근실시간 집계와 주요 지표를 유지하는 것이 빠르게 결정을 내려야 하는 비즈니스에 점점 더 중요해지고 있습니다. 예를 들어, 음악 라이브러리가 가장 많이 다운로드된 노래를 근실시간으로 보여줘야 한다고 가정하겠습니다.

다음과 같은 음악 라이브러리 테이블 레이아웃을 고려하세요.

Music Library Table

| Primary Key | | Data-Item Attributes... | | | |
|---|---|---|---|---|---|
| Partition Key | Sort Key | Attribute 1 | Attribute 2 | Attribute 3 | |
| Song-129 *(song ID)* | Details | Title: Wild Love *(song title)* | Artist: Argyboots *(artist or band name)* | Downloads: 15,314,822 *(lifetime total downloads)* | ...etc. |
| | Month-2018-01 | GSI Primary Key<br>Month: 2018-01 *(download month)* | GSI Secondary Key<br>MonthTotal: 1,746,992 *(month total downloads)* | | |
| | Dld-9349823681 | Time: 2018-01-01T00:00:07 *(download timestamp)* | | | |
| | Dld-9349823682 | Time: 2018-01-01T00:00:07 *(download timestamp)* | | | |
| | Dld-9349823683 | Time: 2018-01-01T00:00:07 *(download timestamp)* | | | |

이 예의 테이블은 songID라는 파티션 키로 노래를 저장합니다. 이 테이블에 Amazon DynamoDB Streams를 활성화하고 스트림에 Lambda 함수를 연결하여 각 노래를 다운로드할 때 Partition-Key=SongID 및 Sort-Key=DownloadID로 테이블에 항목을 추가할 수 있습니다. 이런 업데이트를 수행하면 DynamoDB Streams에 Lambda 함수가 트리거됩니다. Lambda 함수는 songID 별로 다운로드를 집계해 그룹화 하고, 최상위 항목, Partition-Key=songID 및 Sort-Key=Month를 업데이트 할 수 있습니다. 새 집계된 값을 작성한 직후에 lambda 실행이 실패하면 두 번 이상 다시 시도하고 값을 집계하여 대략적인 값을 제공할 수 있습니다.

한 자리 수 밀리초의 지연 시간으로 근실시간으로 업데이트를 읽으려면, Month=2018-01, ScanIndexForward=False, Limit=1이라는 쿼리 조건으로 글로벌 보조 인덱스를 사용합니다.

여기에서 사용한 또 다른 주요 최적화는 글로벌 보조 인덱스가 희소 인덱스이고, 실시간으로 데이터를 검색하는 쿼리가 필요한 항목에서만 사용할 수 있다는 것입니다. 글로벌 보조 인덱스는 인기가 있는 최상위 10곡이나 해당 월에 다운로드된 노래에 대한 정보가 필요한 또 다른 워크플로우로 기능할 수 있습니다.

## DynamoDB의 글로벌 보조 인덱스 오버로딩

Amazon DynamoDB에는 테이블당 20개의 글로벌 보조 인덱스라는 기본 할당량이 있지만, 20개 이상의 데이터 필드에서 인덱싱할 수 있습니다. 스키마가 균일한 RDBMS(관계형 데이터베이스 관리 시스템)의 테이블과 다르게, DynamoDB의 테이블은 한 번에 여러 종류의 데이터 항목을 보관할 수 있습니다. 또한 여러 항목의 동일한 속성에 완전히 다른 종류의 정보를 포함할 수 있습니다.

다양한 종류의 데이터를 저장하는 DynamoDB 테이블 레이아웃에 대한 다음 예를 고려하세요.

| Primary Key | | Data-Item Attributes... | | |
|---|---|---|---|---|
| Partition Key | Sort Key | Attribute 1 | Attribute 2 | ... |
| HR-974 *(employee ID)* | Employee_Name | Data: Murphy, John *(employee name)* | Start: 2008-11-08 *(start date)* | ...etc. |
| | YYYY-Q1 | Data: $5,477 *(order totals in USD)* | Name: Murphy, John *(employee name)* | |
| | HR_confidential | Data: 2008-11-08 *(hire date)* | Name: Murphy, John *(employee name)* | ...etc. |
| | Warehouse_01 | Data: Murphy, John *(employee name)* | | |
| | v0_Job_title | Data: Operator-1 *(job title)* | Start: 2008-11-08 *(start date)* | ...etc. |
| | v1_Job_title | Data: Operator-2 *(job title)* | Start: 2016-11-04 *(start date)* | ...etc. |
| | v2_Job_title | Data: Supervisor-1 *(job title)* | Start: 2017-11-01 *(start date)* | ...etc. |

모든 항목에 공통된 `Data` 속성은 상위 항목에 따라 내용이 다릅니다. 테이블 정렬 키를 파티션 키로 `Data` 속성을 정렬 키로 사용하는 테이블에 대해 글로벌 보조 인덱스를 생성하는 경우, 단일 글로벌 보조 인덱스를 사용해 여러 다양한 쿼리를 수행할 수 있습니다. 이러한 쿼리에는 다음이 포함될 수 있습니다.

- 글로벌 보조 인덱스에서 `Employee_Name`을 파티션 키 값, 직원 이름(예:`Murphy, John`)을 정렬 키 값으로 사용하여 이름으로 직원을 찾습니다.
- 글로벌 보조 인덱스를 사용하고, 창고 ID(예: `Warehouse_01`)를 검색해 특정 창고에서 일하는 모든 지원을 찾습니다.
- `HR_confidential`의 글로벌 보조 인덱스를 파티션-키 값으로 쿼리하고 정렬 키 값에 날짜 범위를 사용하여 최근 채용한 직원 명부를 얻습니다.

# DynamoDB의 일부 테이블 쿼리에 글로벌 보조 인덱스 쓰기 샤딩 사용

특정 기간 내의 최근 데이터를 쿼리해야 하는 경우 DynamoDB가 대부분의 읽기 작업에 파티션 키를 제공해야 하는 요구 사항으로 인해 문제가 발생할 수 있습니다. 이 시나리오를 해결하기 위해 쓰기 샤딩과 글로벌 보조 인덱스(GSI)의 조합을 사용하여 효과적인 쿼리 패턴을 구현할 수 있습니다.

이 접근 방식을 사용하면 리소스 집약적이고 비용이 많이 들 수 있는 전체 테이블 스캔을 수행하지 않고도 시간에 민감한 데이터를 효율적으로 검색하고 분석할 수 있습니다. 테이블 구조와 인덱싱을 전략적으로 설계하면 최적의 성능을 유지하면서 시간 기반 데이터 검색을 지원하는 유연한 솔루션을 생성할 수 있습니다.

주제


- 패턴 설계
- 샤딩 전략
- 샤딩된 GSI 쿼리
- 병렬 쿼리 실행 고려 사항
- 코드 예제


## 패턴 설계

DynamoDB를 사용할 때 최근 데이터 기간에 대한 유연하고 효율적인 쿼리가 가능하도록 쓰기 샤딩과 글로벌 보조 인덱스를 결합한 정교한 패턴을 구현하면 시간 기반 데이터 검색 문제를 해결할 수 있습니다.

테이블 구조

- 파티션 키(PK): "Username"

GSI 구조

- GSI 파티션 키(PK_GSI): "ShardNumber#"
- GSI 정렬 키(SK_GSI): ISO 8601 타임스탬프(예: "2030-04-01T12:00:00Z")

[TABLE] - TimeBounded Metadata Actions

| PK | SK_GSI | Address | PK_GSI |
|---|---|---|---|
| Trenton69 | 2023-12-05T06:40:31.312Z | 117 Hudson Divide | ShardNumber#0 |
| Scot_Langworth-Prosacco | 2024-10-09T14:44:45.819Z | 84021 Herzog Skyway | ShardNumber#6 |
| Juliet.Morissette | 2025-03-28T07:20:56.538Z | 4871 N Broadway | ShardNumber#4 |
| Kay.Von71 | 2024-11-23T16:19:26.763Z | 14554 Odell Throughway | ShardNumber#2 |
| Kiarra43 | 2023-11-15T16:21:57.078Z | 8471 London Road | ShardNumber#1 |
| Marcella91 | 2024-12-17T09:02:31.623Z | 10271 Nick Crescent | ShardNumber#6 |
| Bernard.Lemke36 | 2025-09-09T00:18:34.482Z | 5438 Douglas Groves | ShardNumber#2 |
| Daisha83 | 2024-01-24T06:38:08.482Z | 8786 Armstrong Radial | ShardNumber#3 |
| Marcos_Schmitt58 | 2024-04-06T17:38:58.551Z | 510 S Center Street | ShardNumber#3 |
| Jeremie_VonRueden-Mills | 2025-04-06T04:24:23.701Z | 157 Koch Drives | ShardNumber#1 |
| Verna28 | 2025-10-10T18:42:21.059Z | 70040 Baylee Trafficway | ShardNumber#1 |
| Trycia.Doyle | 2024-01-10T17:00:08.360Z | 9511 Tillman Extensions | ShardNumber#2 |

## 샤딩 전략

샤드 10개를 사용하기로 결정했다면 샤드 번호의 범위는 0~9입니다. 활동을 로깅할 때 샤드 번호를 계산하고(예: 사용자 ID에서 해시 함수를 사용한 다음 샤드 수의 모듈러스를 구함) GSI 파티션 키 앞에 추가합니다. 이 방법은 항목을 여러 샤드에 분산하여 핫 파티션의 위험을 완화합니다.

## 샤딩된 GSI 쿼리

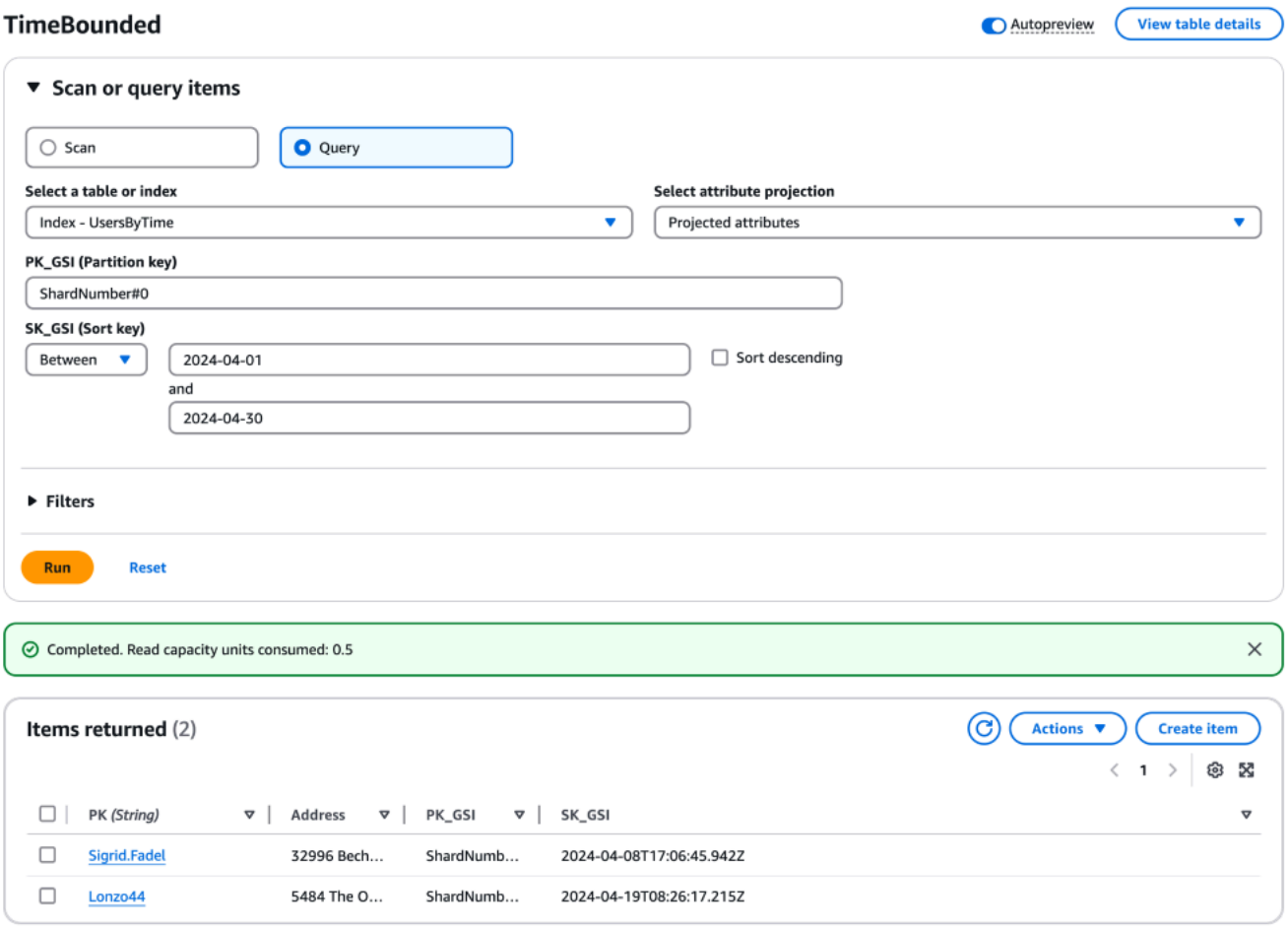
데이터가 여러 파티션 키에 샤딩된 DynamoDB 테이블에서 특정 시간 범위 내의 항목을 모든 샤드에 걸쳐 쿼리하려면 단일 파티션을 쿼리하는 것과 다른 접근 방식이 필요합니다. DynamoDB 쿼리는 한 번에 하나의 파티션 키로 제한되므로 단일 쿼리 작업으로 여러 샤드에 걸쳐 직접 쿼리할 수 없습니다. 그러나 애플리케이션 수준 로직을 통해 각각 특정 샤드를 대상으로 하는 여러 쿼리를 수행한 다음 결과를 집계하여 원하는 결과를 얻을 수 있습니다. 아래 절차에 이렇게 하는 방법이 나와 있습니다.

샤드를 쿼리하고 집계하려면

1. 샤딩 전략에 사용되는 샤드 번호의 범위를 식별합니다. 예를 들어 샤드가 10개인 경우 샤드 번호의 범위는 0~9입니다.

2. 각 샤드에 대해 쿼리를 구성하고 실행하여 원하는 시간 범위 내의 항목을 가져옵니다. 이러한 쿼리를 병렬로 실행하여 효율성을 개선할 수 있습니다. 파티션 키를 샤드 번호와 함께 사용하고 정렬 키를 이러한 쿼리의 시간 범위와 함께 사용합니다. 다음은 단일 샤드에 대한 쿼리의 예입니다.

```
aws dynamodb query \
    --table-name "YourTableName" \
    --index-name "YourIndexName" \
    --key-condition-expression "PK_GSI = :pk_val AND SK_GSI BETWEEN :start_date
 AND :end_date" \
    --expression-attribute-values '{
        ":pk_val": {"S": "ShardNumber#0"},
        ":start_date": {"S": "2024-04-01"},
        ":end_date": {"S": "2024-04-30"}
    }'
```




각 샤드에 대해 파티션 키를 적절히 조정하여(예: "ShardNumber#1", "ShardNumber#2", ..., "ShardNumber#9") 이 쿼리를 반복합니다.

3. 모든 쿼리가 완료된 후 각 쿼리의 결과를 집계합니다. 애플리케이션 코드에서 이 집계를 수행하여 결과를 지정된 시간 범위 내의 모든 샤드 항목을 나타내는 단일 데이터세트로 결합합니다.

## 병렬 쿼리 실행 고려 사항

각 쿼리는 테이블 또는 인덱스의 읽기 용량을 사용합니다. 프로비저닝된 처리량을 사용하는 경우 병렬 쿼리의 버스트를 처리하기에 충분한 용량을 테이블에 프로비저닝해야 합니다. 온디맨드 용량을 사용하는 경우 잠재적 비용 영향에 유의하세요.

## 코드 예제

Python을 사용하여 DynamoDB의 샤드 간에 병렬 쿼리를 실행하려면 Amazon Web Services SDK for Python인 boto3 라이브러리를 사용할 수 있습니다. 이 예제에서는 boto3를 설치하고 적절한 AWS 자격 증명으로 구성했다고 가정합니다.

다음 Python 코드는 지정된 시간 범위에서 여러 샤드에 걸쳐 병렬 쿼리를 수행하는 방법을 보여줍니다. concurrent futures를 사용하여 병렬로 쿼리를 실행하므로 순차 실행에 비해 전체 실행 시간이 단축됩니다.

```
import boto3
from concurrent.futures import ThreadPoolExecutor, as_completed

# Initialize a DynamoDB client
dynamodb = boto3.client('dynamodb')

# Define your table name and the total number of shards
table_name = 'YourTableName'
total_shards = 10  # Example: 10 shards numbered 0 to 9
time_start = "2030-03-15T09:00:00Z"
time_end = "2030-03-15T10:00:00Z"

def query_shard(shard_number):
    """
    Query items in a specific shard for the given time range.
    """
    response = dynamodb.query(
        TableName=table_name,
        IndexName='YourGSIName',  # Replace with your GSI name
        KeyConditionExpression="PK_GSI = :pk_val AND SK_GSI BETWEEN :date_start
 AND :date_end",
        ExpressionAttributeValues={
            ":pk_val": {"S": f"ShardNumber#{shard_number}"},
```

```
            ":date_start": {"S": time_start},
            ":date_end": {"S": time_end},
        }
    )
    return response['Items']

# Use ThreadPoolExecutor to query across shards in parallel
with ThreadPoolExecutor(max_workers=total_shards) as executor:
    # Submit a future for each shard query
    futures = {executor.submit(query_shard, shard_number): shard_number for
 shard_number in range(total_shards)}

    # Collect and aggregate results from all shards
    all_items = []
    for future in as_completed(futures):
        shard_number = futures[future]
        try:
            shard_items = future.result()
            all_items.extend(shard_items)
            print(f"Shard {shard_number} returned {len(shard_items)} items")
        except Exception as exc:
            print(f"Shard {shard_number} generated an exception: {exc}")

# Process the aggregated results (e.g., sorting, filtering) as needed
# For example, simply printing the count of all retrieved items
print(f"Total items retrieved from all shards: {len(all_items)}")
```

이 코드를 실행하기 전에 `YourTableName` 및 `YourGSIName`을 DynamoDB 설정의 실제 테이블 및 GSI 이름으로 교체해야 합니다. 또한 특정 요구 사항에 따라 `total_shards`, `time_start` 및 `time_end` 변수를 조정합니다.

이 스크립트는 각 샤드에서 지정된 시간 범위 내의 항목을 쿼리하고 결과를 집계합니다.

## 글로벌 보조 인덱스를 사용하여 DynamoDB에 최종적으로 일관된 복제본 생성

글로벌 보조 인덱스를 사용하여 최종적으로 일관된 테이블 복제본을 생성할 수 있습니다. 복제본을 생성하면 다음 작업을 수행할 수 있습니다.

- 독자마다 각각 다르게 프로비저닝된 읽기 용량을 설정합니다. 예를 들어, 우선 순위가 높은 쿼리를 처리해야 해서 최고의 읽기 성능이 필요한 애플리케이션과 우선 순위가 낮아서 읽기 작업의 병목 현상이 용인되는 애플리케이션 2개가 있다고 가정하겠습니다.

두 애플리케이션이 동일한 테이블을 읽는다면, 우선 순위가 낮은 애플리케이션의 과도한 읽기 로드가 해당 테이블에서 사용할 수 있는 읽기 용량을 모두 소비할 수도 있습니다. 이는 우선 순위가 높은 애플리케이션의 읽기 작업을 제한하게 됩니다.

대신 글로벌 보조 인덱스를 통해 읽기 용량을 테이블과 다르게 설정한 복제본을 생성할 수 있습니다. 그런 후, 낮은 우선 순위의 앱이 테이블 대신 복제본에 쿼리를 하도록 만들 수 있습니다.

- 테이블 읽기를 완전히 없앱니다. 예를 들어, 웹사이트에서 대량의 클릭스트림 활동을 캡처하는 애플리케이션이 있는데, 읽기에 방해가 초래되는 위험을 원하지 않는다고 가정하겠습니다. 이 테이블을 격리해 다른 애플리케이션의 읽기를 방지하면서(IAM 정책 조건을 사용하여 세분화된 액세스 제어 구현참조), 다른 애플리케이션은 글로벌 보조 인덱스로 생성한 복제본을 읽도록 만들 수 있습니다.

상위 테이블과 키 스키마가 동일하면서, 키를 제외한 속성은 일부(또는 모두)를 인덱스로 가져오는 글로벌 보조 인덱스를 설정해 복제본을 생성할 수 있습니다. 그러면 애플리케이션에서는 읽기 작업 일부 또는 모두를 상위 테이블이 아닌 이 글로벌 보조 인덱스로 보낼 수 있습니다. 그런 후, 상위 테이블의 프로비저닝된 읽기 용량을 변경하지 않은 상태에서 글로벌 보조 인덱스의 프로비저닝된 읽기 용량을 조정할 수 있습니다.

단 항상 상위 테이블에 데이터를 작성한 후 작성된 데이터가 인덱스에 나타날 때까지 짧게 전달 지연이 발생합니다. 다시 말해 애플리케이션은 글로벌 보조 인덱스 복제본이 유일하게 상위 테이블에 최종적으로 일관된 복제본이라는 점을 고려해야 합니다.

여러 글로벌 보조 인덱스 복제본을 생성해 여러 읽기 패턴을 지원할 수 있습니다. 단 복제본을 생성할 때, 각 읽기 패턴에 실제 필요한 속성만 반영해야 합니다. 그래야만 애플리케이션이 상위 테이블의 항목을 읽지 않고, 필요한 데이터만 가져오면서, 프로비저닝된 읽기 용량을 더 적게 소비합니다. 이런 최적화 덕분에 시간이 지날수록 많은 비용을 절감할 수 있습니다.

## DynamoDB에 큰 항목과 속성 저장 모범 사례

Amazon DynamoDB는 테이블에 저장하는 각 항목의 크기를 400KB로 제한합니다(Amazon DynamoDB의 할당량 참조). 애플리케이션이 DynamoDB 크기 제한보다 많은 데이터를 항목에 저장해야 하는 경우, 하나 이상의 라지 속성을 압축하거나, 항목을 여러 개(정렬 키로 효율적으로 인덱싱된)로 나눌 수 있습니다. 또한 항목을 Amazon Simple Storage Service(Amazon S3)에 객체로 저장하고 Amazon S3 객체 식별자를 DynamoDB 항목에 저장할 수도 있습니다.

가장 좋은 방법은 항목을 작성할 때 `ReturnConsumedCapacity` 파라미터를 활용하여 최대 400KB의 항목 크기 한도에 근접하는 항목 크기를 모니터링하고 이에 대해 경고하는 것입니다. 최대 항목 크기를

초과하면 쓰기 시도가 실패합니다. DynamoDB는 ValidationException 오류를 반환합니다. 애플리케이션에 영향을 미치기 전에 항목 크기를 모니터링하고 경고하여 항목 크기 문제를 완화할 수 있습니다.

## 큰 속성 값 압축

라지 속성 값을 압축하면 DynamoDB의 항목 제한에 맞출 수 있고, 스토리지 비용을 절감할 수 있습니다. GZIP 또는 LZO와 같은 압축 알고리즘은 이진 출력을 생성하며, 이를 항목 내의 `Binary` 속성 유형에 저장할 수 있습니다.

포럼 사용자가 작성한 메시지를 저장하는 테이블을 예로 들어 보겠습니다. 이러한 메시지에는 압축하기에 적합한 긴 텍스트 문자열이 포함되는 경우가 많습니다. 압축하면 항목 크기를 줄일 수 있지만, 압축된 속성 값은 필터링에 유용하지 않다는 단점이 있습니다.

DynamoDB에서 긴 메시지를 압축하는 방법을 보여주는 샘플 코드는 다음을 참조하세요.

- 예: AWS SDK for Java 문서 API를 사용하여 이진 형식 속성 처리
- 예: AWS SDK for .NET 하위 수준 API를 사용하여 이진 형식 속성 처리

## 수직 파티셔닝

큰 항목을 처리하는 또 다른 방법은 항목을 더 작은 데이터 청크로 나누고 모든 관련 항목을 파티션 키 값을 기준으로 연결하는 것입니다. 그런 다음 정렬 키 문자열을 사용하여 정렬 키 문자열과 함께 저장된 관련 정보를 식별할 수 있습니다. 이 방법을 사용하고 여러 항목을 동일한 파티션 키 값으로 그룹화하면 항목 모음이 생성됩니다.

이 방법에 대한 자세한 정보는 다음을 참조하세요.

- Use vertical partitioning to scale data efficiently in Amazon DynamoDB
- Implement vertical partitioning in Amazon DynamoDB using AWS Glue

## Amazon S3에 큰 속성 값 저장

앞에서 언급한 것처럼 Amazon S3을 사용하여 DynamoDB 항목에 맞지 않는 라지 속성 값을 저장할 수도 있습니다. 이를 Amazon S3에 객체로 저장한 후, 객체 식별자를 DynamoDB 항목에 저장할 수 있습니다.

또한 Amazon S3의 객체 메타데이터 지원을 사용하여 DynamoDB의 상위 항목에 링크를 다시 제공할 수 있습니다. 항목의 기본 키 값을 Amazon S3 객체의 Amazon S3 메타데이터로 저장합니다. 이렇게 하면 Amazon S3 객체를 유지 관리할 때 도움을 받을 수 있습니다.

예를 들어 `ProductCatalog` 테이블을 고려해 보세요. 이 테이블의 항목은 항목 가격, 설명, 책의 저자 및 여러 제품의 규격에 대한 정보를 저장합니다. 항목에 맞지 않을 정도로 큰 제품 이미지를 저장하고 싶다면, DynamoDB 대신 Amazon S3에 이미지를 저장할 수 있습니다.

이 전략을 구현할 때는 다음을 염두에 두세요.

- DynamoDB는 Amazon S3 및 DynamoDB를 교차하는 트랜잭션을 지원하지 않습니다. 따라서 애플리케이션이 분리된 Amazon S3 객체를 정리하는 등 실패를 처리할 수 있어야 합니다.
- Amazon S3는 객체 식별자의 길이를 제한합니다. 따라서 길이가 지나치게 긴 객체 식별자를 생성하지 않거나 기타 Amazon S3 제약을 위반하지 않는 방식으로 데이터를 구성해야 합니다.

Amazon S3를 사용하는 방법에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서를 참조하세요.

# DynamoDB의 시계열 데이터 처리 모범 사례

Amazon DynamoDB의 일반 설계 원칙은 사용할 테이블의 수를 최소한으로 유지할 것을 권장합니다. 대부분의 애플리케이션에서 단 하나의 테이블만 필요합니다. 하지만 시계열 데이터의 경우 기간별로 애플리케이션당 하나의 테이블을 사용하여 처리하는 것이 가장 좋을 수도 있습니다.

## 시계열 데이터의 설계 패턴

많은 볼륨의 이벤트를 추적하고 싶은 경우에 일반적인 시계열 시나리오를 고려하세요. 쓰기 액세스 패턴은 기록할 모든 이벤트가 오늘 날짜인 패턴입니다. 읽기 액세스 패턴은 오늘 이벤트를 가장 많이, 어제 이벤트는 이보다 적게, 더 오래된 이벤트는 거의 읽지 않는 패턴입니다. 이 문제를 처리하는 한 가지 방법은 현재 데이터와 시간을 기본 키에 빌드하는 것입니다.

이런 종류의 시나리오를 가장 효과적으로 처리할 수 있는 설계 패턴입니다.

- 필요한 읽기 및 쓰기 용량과 필요한 인덱스로 프로비저닝되는 테이블을 기간당 하나씩 생성합니다.
- 각 기간이 끝나기 전에 다음 기간에 대한 테이블을 사전에 빌드합니다. 현재 기간이 끝나면, 이벤트 트래픽을 새 테이블로 보냅니다. 기록 기간이 설명된 테이블에 이름을 지정할 수 있습니다.
- 테이블에 기록이 중지된 즉시 프로비저닝된 쓰기 용량은 더 낮은 값(예: 1WCU)으로 줄이고, 적절한 읽기 용량을 프로비저닝합니다. 시간이 지나면서 이전 테이블에 프로비저닝된 읽기 용량을 줄입니다. 내용이 거의 또는 아예 필요 없다면 테이블을 아카이브하거나 삭제할 수 있습니다.

가장 높은 트래픽 볼륨이 발생하는 현재 기간에 필요한 리소스를 할당하고 적극적으로 사용되지 않는 이전 테이블에 대한 프로비저닝을 축소하여 비용을 절감하는 것이 좋습니다. 비즈니스 요구 사항에 따

라 논리적 파티션 키에 트래픽을 균등하게 배포하기 위해 쓰기 샤딩을 고려해야 할 수 있습니다. 자세한 내용은 쓰기 샤딩을 사용해 DynamoDB 테이블에 워크로드를 고르게 배포 섹션을 참조하세요.

## 시계열 테이블 예

다음은 현재 테이블은 더 높은 읽기/쓰기 용량으로 프로비저닝되고 이전 테이블은 자주 액세스되지 않으므로 축소되는 시계열 데이터 예입니다.

Current table — Provisioned at: WCU=750 and RCU=300

| Primary Key | | Attributes | | |
|---|---|---|---|---|
| Partition Key | Sort Key | Radiant Intensity | Wavelength | ... |
| 2018-03-15 | 00:00:00.002 | 17.372 W/Sr | 713 nm | ... |
| 2018-03-15 | 00:00:00.004 | 17.385 W/Sr | 712 nm | ... |
| 2018-03-15 | 00:00:00.005 | 17.478 W/Sr | 708 nm | ... |
| 2018-03-15 | 00:00:00.007 | 19.172 W/Sr | 674 nm | ... |
| . . . | . . . | . . . | . . . | ... |

Previous table — Provisioned at: WCU=1 and RCU=100

| Primary Key | | Attributes | | |
|---|---|---|---|---|
| Partition Key | Sort Key | Radiant Intensity | Wavelength | ... |
| 2018-03-14 | 00:00:00.001 | 16.473 W/Sr | 512 | ... |
| 2018-03-14 | 00:00:00.003 | 16.489 W/Sr | 519 | ... |
| 2018-03-14 | 00:00:00.004 | 16.814 W/Sr | 522 | ... |
| 2018-03-14 | 00:00:00.006 | 16.719 W/Sr | 506 | ... |
| . . . | . . . | . . . | . . . | ... |

Older table — Provisioned at: WCU=1 and RCU=1

| Primary Key | | Attributes | | |
|---|---|---|---|---|
| Partition Key | Sort Key | Radiant Intensity | Wavelength | ... |
| 2018-03-10 | 00:00:00.001 | 13.669 W/Sr | 456 | ... |
| 2018-03-10 | 00:00:00.002 | 13.522 W/Sr | 459 | ... |
| 2018-03-10 | 00:00:00.004 | 13.596 W/Sr | 457 | ... |
| 2018-03-10 | 00:00:00.005 | 15.721 W/Sr | 425 | ... |
| . . . | . . . | . . . | . . . | ... |

# DynamoDB 테이블의 다대다 관계 관리 모범 사례

인접 목록은 Amazon DynamoDB의 다대다 관계 모델링에 유용한 설계 패턴입니다. 대체로 DynamoDB에 그래프 데이터(노드 및 엣지)를 표시하는 방법을 제공합니다.

## 인접 목록 설계 패턴

특정 애플리케이션의 여러 개체 간 관계가 다대다 관계일 때, 이런 관계를 인접 목록으로 모델링할 수 있습니다. 이 패턴에서는 파티션 키를 사용하여 모든 최상위 개체(그래프 모델의 노드와 비슷)를 표현합니다. 다른 개체(그래프의 엣지)와의 관계는 대상 개체 ID(대상 노드)에 정렬 키 값을 설정해 파티션 내에서 하나의 항목으로 표시합니다.

이런 패턴은 데이터 중복을 최소화하고, 쿼리 패턴을 단순화시켜 대상 개체(대상 노드의 엣지)와 연결된 모든 개체(노드)를 찾을 수 있다는 점이 장점입니다.

실제 이런 패턴을 유용하게 사용하는 사례는 인보이스 여러 요금 청구 항목이 있는 인보이스 시스템입니다. 하나의 요금 항목이 여러 인보이스에 속할 수 있습니다. 이 예제의 파티션 키는 `InvoiceID` 또는 `BillID`입니다. `BillID` 파티션에는 청구서와 관련된 모든 속성이 포함되어 있습니다. `InvoiceID` 파티션에는 인보이스별 속성이 저장된 항목과 인보이스에 롤업된 각 `BillID`에 대한 항목이 있습니다.

스키마는 다음과 같습니다.

| Table | Primary Key | | Data Attributes... | |
|---|---|---|---|---|
| | *Partition Key* | *Sort Key (and GSI PK)* | | |
| | `Invoice-92551` | **Inv_ID:** Invoice-92551 *(invoice ID)* | **Dated:** 2018-02-07 *(date created)* | More attributes of this invoice... |
| | | **Bill_ID:** Bill-4224663 *(bill ID)* | **Dated:** 2017-12-03 *(date created)* | Attributes of this bill *in this invoice*.. |
| | | **Bill_ID:** Bill-4224687 *(bill ID)* | **Dated:** 2018-01-09 *(date created)* | Attributes of this bill *in this invoice*.. |
| | `Invoice-92552` | **Inv_ID:** Invoice-92552 *(invoice ID)* | **Dated:** 2018-03-04 *(date created)* | More attributes of this invoice... |
| | | **Bill_ID:** Bill-4224687 *(bill ID)* | **Dated:** 2018-01-09 *(date created)* | Attributes of this bill *in this invoice*.. |
| | `Bill-4224663` | **Bill_ID:** Bill-4224663 *(bill ID)* | **Dated:** 2017-12-03 *(date created)* | More attributes of this bill... |
| | `Bill-4224687` | **Bill_ID:** Bill-4224687 *(bill ID)* | **Dated:** 2018-01-09 *(date created)* | More attributes of this bill... |

앞서 스키마를 사용하여 특정 인보이스의 요금 항목을 테이블 기본 키를 사용해 쿼리할 수 있다는 것을 확인할 수 있습니다. 특정 요금 항목의 일부가 포함된 모든 인보이스를 찾으려면 테이블 정렬 키에 글로벌 보조 인덱스를 생성합니다.

글로벌 보조 인덱스의 프로젝션은 다음과 같습니다.

| GSI | Primary Key<br>*Partition Key* | Projected Attributes... | |
|---|---|---|---|
| | Bill-4224663 | **Bill_ID:** Bill-4224663 *(table primary key)* | Attributes of this bill... |
| | | **Inv_ID:** Invoice-92551 *(table primary key)* | Attributes of this bill *in this invoice*.. |
| | Bill-4224687 | **Bill_ID:** Bill-4224687 *(table primary key)* | Attributes of this bill... |
| | | **Inv_ID:** Invoice-92551 *(table primary key)* | Attributes of this bill *in this invoice*.. |
| | | **Inv_ID:** Invoice-92552 *(table primary key)* | Attributes of this bill *in this invoice*.. |
| | Invoice-92551 | **Inv_ID:** Invoice-92551 *(table primary key)* | Attributes of this invoice... |
| | Invoice-92552 | **Inv_ID:** Invoice-92552 *(table primary key)* | Attributes of this invoice... |

## 구체화된 그래프 패턴

피어 간 순위, 개체 간 공통 관계, 이웃 개체 상태, 기타 그래프 스타일 워크스타일에 대한 이해를 토대로 빌드되는 애플리케이션이 많습니다. 이런 유형의 애플리케이션에 대해서는 다음 스키마 설계 패턴을 고려하세요.

| TABLE | Primary Key | | Attributes | | |
|---|---|---|---|---|---|
| | PK (NodeId) | SK (TypeTarget, GSI 2 SK) | Data | GSI PK | Graph Projections |
| | 1 | DATE\|2\|BIRTH | 1980-12-19 | Hash(Person.Data) | |
| | | PERSON\|1 | Data (GSI1 SK) | GSI PK | |
| | | | John Doe | Hash(Person.Data) | |
| | | PERSON\|5\|FRIEND | Data | GSI PK | |
| | | | Jane Smith | Hash(Person.Data) | |
| | | PLACE\|4\|BIRTH | Data | GSI PK | |
| | | | USA\|Texas\|Austin | Hash(Person.Data) | |
| | | SKILL\|6 | Data | GSI PK | |
| | | | Java Developer\|Senior | Hash(Person.Data) | |
| | 2 | DATE\|2 | Data | GSI PK | |
| | | | 1980-12-19 | 0 | |
| | 3 | PLACE\|3 | Data | GSI PK | |
| | | | UK\|England\|London | 0 | |
| | 4 | PLACE\|4 | Data | GSI PK | |
| | | | USA\|Texas\|Austin | 0 | ... |
| | 5 | DATE\|2\|BIRTH | Data | GSI PK | |
| | | | 1980-12-19 | Hash(Person.Data) | |
| | | PERSON\|5 | Data | GSI PK | |
| | | | Jane Smith | Hash(Person.Data) | |
| | | PERSON\|1\|FRIEND | Data | GSI PK | |
| | | | John Doe | Hash(Person.Data) | |
| | | PLACE\|3\|BIRTH | Data | GSI PK | |
| | | | UK\|England\|London | Hash(Person.Data) | |
| | | SKILL\|7 | Data | GSI PK | |
| | | | Guitar\|Advanced | Hash(Person.Data) | |
| | 6 | SKILL\|6 | Data | GSI PK | |
| | | | Java Developer | 0 | |
| | 7 | SKILL\|7 | Data | GSI PK | |
| | | | Guitar | 0 | |

| | Primary Key | | Attributes | | |
|---|---|---|---|---|---|
| | GSI PK | GSI 1 SK (Data) | | | |
| GSI 1 | O-N | 1980-12-19 | NodeID | TypeTarget | Graph Projections |
| | | | 2 | DATE\|2 | ... |
| | | | NodeID | TypeTarget | |
| | | | 1 | DATE\|2\|BIRTH | |
| | | | NodeID | | |
| | | | 5 | | |
| | | Guitar | NodeID | TypeTarget | |
| | | | 7 | SKILL\|7 | |
| | | Guitar\|Advanced | NodeID | | |
| | | | 5 | | |
| | | Jane Smith | NodeID | TypeTarget | |
| | | | 5 | Person\|5 | |
| | | | NodeID | TypeTarget | |
| | | | 1 | Person\|5\|FRIEND | |
| | | Java Developer | NodeID | TypeTarget | |
| | | | 6 | SKILL\|6 | |
| | | Java Developer\|Senior | NodeID | | |
| | | | 1 | | |
| | | John Doe | NodeID | TypeTarget | |
| | | | 1 | Person\|1 | |
| | | | NodeID | TypeTarget | |
| | | | 5 | Person\|1\|FRIEND | |
| | | UK\|England\|London | NodeID | TypeTarget | |
| | | | 3 | PLACE\|3 | |
| | | | NodeID | TypeTarget | |
| | | | 1 | PLACE\|3\|BIRTH | |
| | | USA\|Texas\|Austin | NodeID | TypeTarget | |
| | | | 4 | PLACE\|4 | |
| | | | NodeID | TypeTarget | |
| | | | 5 | PLACE\|4\|BIRTH | |

| | Primary Key | | Attributes | | |
|---|---|---|---|---|---|
| | GSI PK | GSI 2 SK (TypeTarget) | | | |
| GSI 2 | O-N | DATE\|2 | NodeID | Data | Graph Projections |
| | | | 2 | 1980-12-19 | ... |
| | | DATE\|2\|BIRTH | NodeID | | |
| | | | 1 | | |
| | | | NodeID | | |
| | | | 5 | | |
| | | PERSON\|1 | NodeID | Data | |
| | | | 1 | John Doe | |
| | | PERSON\|1\|FRIEND | NodeID | | |
| | | | 5 | | |
| | | PERSON\|5 | NodeID | Data | |
| | | | 5 | Jane Smith | |
| | | PERSON\|5\|FRIEND | NodeID | | |
| | | | 1 | | |
| | | PLACE\|3 | NodeID | Data | |
| | | | 3 | UK\|England\|London | |
| | | PLACE\|3\|BIRTH | NodeID | | |
| | | | 5 | | |
| | | PLACE\|4 | NodeID | Data | |
| | | | 4 | USA\|texas\|Austin | |
| | | PLACE\|4\|BIRTH | NodeID | | |
| | | | 1 | | |
| | | SKILL\|6 | NodeID | Data | |
| | | | 6 | Java Developer | |
| | | | NodeID | Data | |
| | | | 1 | Java Developer\|Senior | |
| | | SKILL\|7 | NodeID | Data | |
| | | | 7 | Guitar | |
| | | | NodeID | Data | |
| | | | 5 | Guitar\|Advanced | |

앞의 스키마는 그래프의 엣지와 노드를 정의하는 항목이 포함된 데이터 파티션 세트로 정의한 그래프 데이터의 구조를 보여주고 있습니다. 엣지 항목에는 `Target` 및 `Type` 속성이 포함되어 있습니다. 이러한 속성들을 "TypeTarget"이라는 복합 키의 일부로 사용하여 기본 테이블이나 보조 글로벌 보조 인덱스의 파티션에 있는 항목을 식별할 수 있습니다.

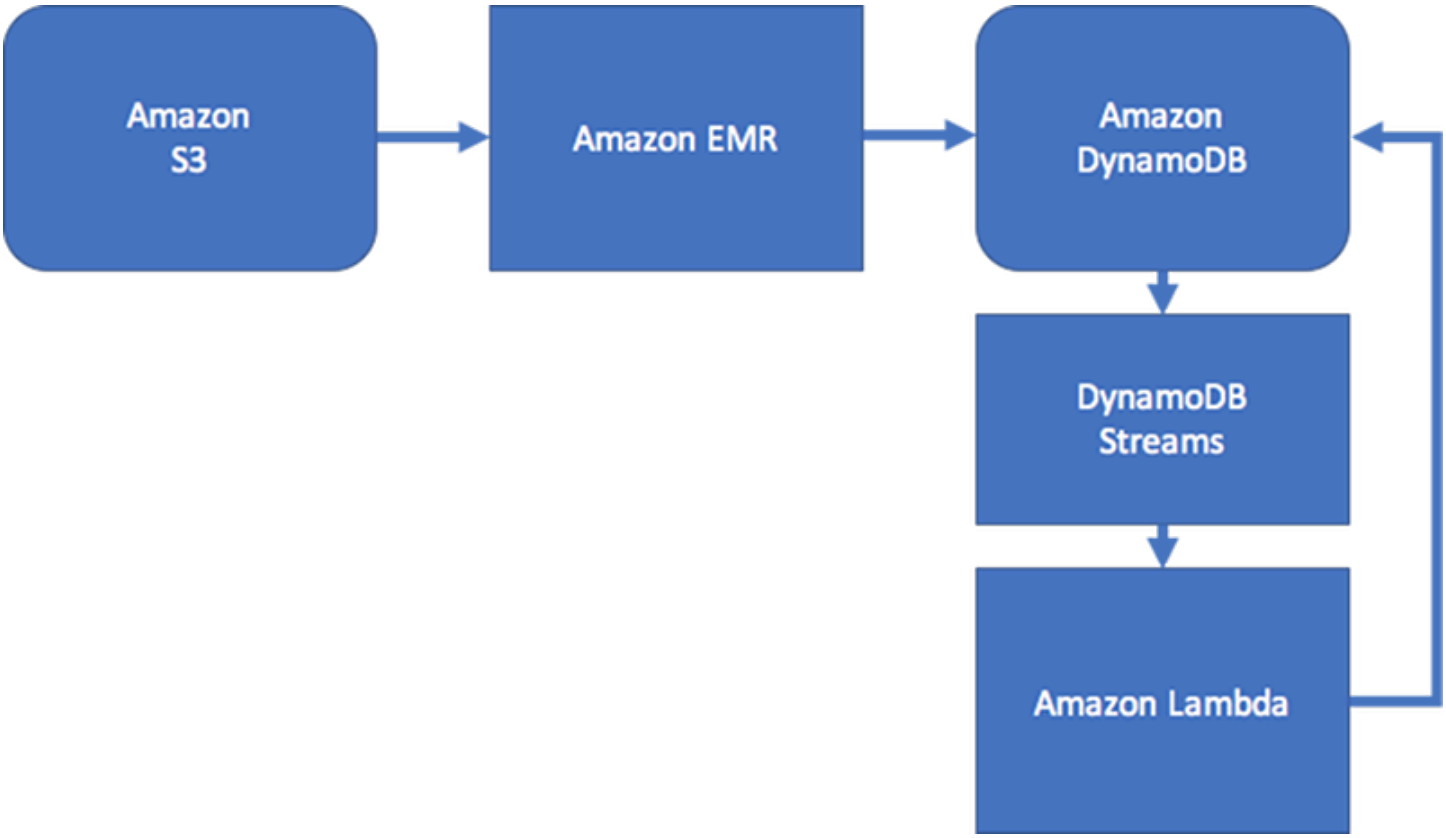
첫 번째 글로벌 보조 인덱스는 `Data` 속성에 빌드됩니다. 이 속성은 앞서 설명한 글로벌 보조 인덱스 오버로딩을 사용하여 `Dates`, `Names`, `Places`, `Skills` 등 몇몇 속성 유형을 인덱스합니다. 효과적으로 4개의 속성들을 인덱싱하는 글로벌 보조 인덱스입니다.

테이블에 항목들을 삽입하면, 지능형 샤딩 전략을 사용하여 필요한 만큼 많은 글로벌 보조 인덱스의 논리적 파티션에 크게 집계된(Birthdate, Skill) 항목 세트를 배포, '핫' 읽기/쓰기 문제를 방지할 수 있습니다.

이렇게 디자인 패턴을 결합해서 고효율 실시간 그래프 워크플로우를 위한 데이터 스토어를 구현합니다. 이런 워크플로우는 고성능의 이웃 개체 상태와 소셜 네트워킹 애플리케이션의 추천 엔진에 대한 엣지 집계 쿼리, 노드 순위, 하위 트리 집계, 기타 범용 그래프 사용 사례를 제공할 수 있습니다.

실시간의 데이터 일관성이 중요하지 않은 사용 사례인 경우, 예약한 Amazon EMR 프로세스를 사용해 워크플로우에 관련 그래프를 요약 집계해서 엣지를 채울 수 있습니다. 애플리케이션이 엣지가 그래프에 추가된 때를 즉시 알 필요가 없다면, 예약된 프로세스를 사용해 결과를 집계할 수 있습니다.

일정 수준의 일관성을 유지하기 위해 설계에 Amazon DynamoDB Streams 및 AWS Lambda를 포함하여 엣지 업데이트를 처리할 수 있습니다. 또한 Amazon EMR 작업을 사용하여 정기적인 간격으로 결과를 확인할 수 있습니다. 다음 다이어그램에 이런 방식이 설명되어 있습니다. 실시간 쿼리 비용이 높고 개별 사용자의 업데이트를 즉시 파악할 필요성이 낮은 소셜 네트워킹 애플리케이션에 많이 사용되고 있습니다.




IT 서비스 관리(ITSM) 및 보안 애플리케이션은 일반적으로 복잡한 엣지들이 통합되어 구성된 개체 상태 변경에 실시간으로 응답할 필요가 있습니다. 이런 애플리케이션에는 두 번째와 세 번째 수준의 관계나 복잡한 엣지 통과에 대해 실시간으로 다중 노드 집계를 지원할 수 있는 시스템이 필요합니다. 사용 사례에 이런 유형의 실시간 그래프 쿼리 워크플로우가 필요하다면, 워크플로우 관리에 Amazon Neptune 사용을 고려하는 것이 좋습니다.

> **Note**
>
> 연결성이 높은 데이터 세트를 쿼리하거나 여러 노드를 통과해야 하는 쿼리(멀티 홉 쿼리라고도 함)를 몇 밀리초의 지연 시간으로 실행해야 하는 경우 Amazon Neptune을 사용하는 것이 좋습

니다. Amazon Neptune은 수십억 개의 관계를 저장하고 몇 밀리초의 지연 시간으로 그래프를 쿼리하도록 최적화된 특수 목적 고성능 그래프 데이터베이스 엔진입니다.

# DynamoDB의 데이터 쿼리 및 스캔 모범 사례

이 섹션에서는 Amazon DynamoDB의 `Query` 및 `Scan` 작업에 대한 몇 가지 모범 사례를 살펴봅니다.

## 스캔에 대한 성능 고려 사항

일반적으로 `Scan` 작업은 DynamoDB의 다른 작업보다 비효율적입니다. `Scan` 작업은 항상 전체 테이블이나 보조 인덱스를 스캔합니다. 그런 후 값을 필터링하여 원하는 결과를 얻기 때문에 결과 세트에서 데이터를 제거해야 하는 단계가 추가됩니다.

가능한 경우, 대용량 테이블 또는 인덱스에서 필터를 사용해 다수의 결과를 제거해야 하는 경우에는 `Scan` 작업을 최대한 피하는 것이 좋습니다. 여기에 테이블이나 인덱스 용량이 커질수록 `Scan` 작업 속도가 느려집니다. `Scan` 작업은 요청한 값을 찾기 위해 전체 항목을 검사하기 때문에 대용량 테이블이나 인덱스일 경우에는 단 한 번의 작업으로 프로비저닝된 처리량을 모두 사용할 수 있습니다. 이런 경우 응답 시간을 빠르게 하려면 애플리케이션이 `Scan`이 아닌 `Query`를 사용하도록 테이블과 인덱스를 설계해야 합니다. (테이블의 경우에는 `GetItem` 및 `BatchGetItem` API 사용도 고려할 수 있습니다.)

그 밖에도 `Scan` 작업이 요청량에 미치는 영향을 최소화할 수 있도록 애플리케이션을 설계할 수 있습니다. 여기에는 `Scan` 작업 대신 글로벌 보조 인덱스를 사용하는 것이 더 효율적일 수 있는 경우의 모델링이 포함될 수 있습니다. 이 프로세스에 대한 자세한 내용은 다음 비디오에 나와 있습니다.

저속 액세스 패턴 모델링

## 갑작스러운 읽기 활동 급증 방지

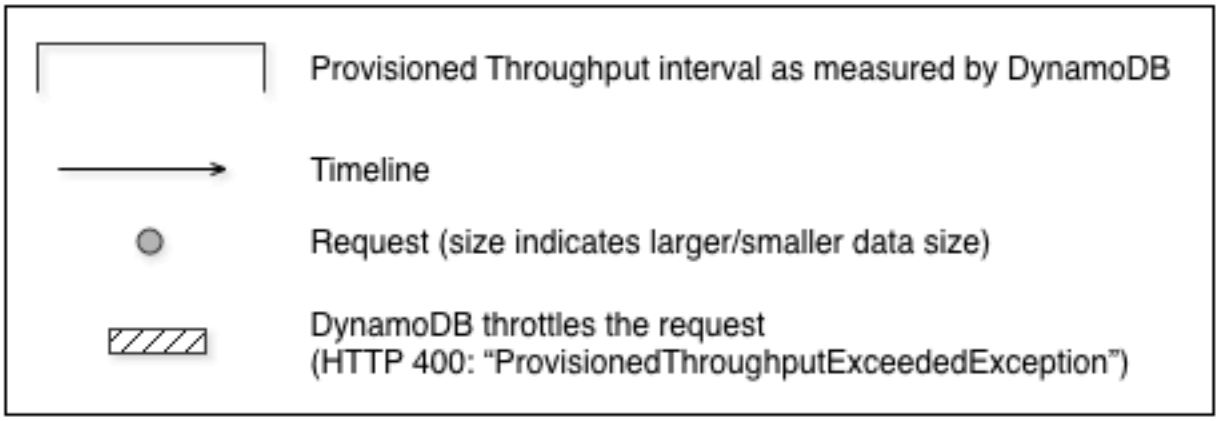
테이블을 생성할 때는 읽기 및 쓰기 용량 단위의 요건을 설정합니다. 읽기의 경우, 4KB 데이터의 초당 강력하게 일관된 읽기 요청 수로 용량 단위를 표현합니다. 최종적으로 일관된 읽기일 때는 읽기 용량 단위가 초당 4KB 읽기 요청 2개입니다. `Scan` 작업은 기본적으로 최종적으로 일관된 일기를 실행하며 최대 1MB(한 페이지)까지 데이터를 반환할 수 있습니다. 따라서 `Scan` 요청 한 번으로 (1MB 페이지 크기/4KB 항목 크기)/2(최종적으로 일관된 읽기 수)의 결과인 128개의 읽기 작업을 소비할 수 있습니다. 대신 강력하게 일관된 읽기를 요청하면 `Scan` 작업은 두 배 더 많은 프로비저닝된 처리량인 256개의 읽기 작업을 소비합니다.

이 현장은 테이블에 구성된 읽기 용량과 비교하여 갑작스러운 사용량 급증을 나타냅니다. 스캔 작업에서 이러한 용량 단위의 사용은 용량 단위의 부족으로 이어져 동일한 테이블에서 잠재적으로 더욱

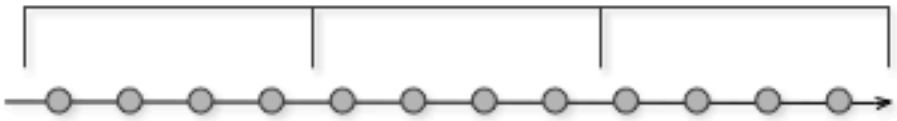
중요한 요청을 실행하지 못하는 원인이 될 수 있습니다. 그 결과 더욱 중요한 요청을 실행하더라도 ProvisionedThroughputExceeded 예외가 발생할 가능성이 높습니다.

Scan이 사용하는 용량 단위의 갑작스러운 증가 외에도 문제는 또 있습니다. 스캔 요청 시 파티션에서 서로 인접한 항목을 읽어오다 보니 동일한 파티션의 용량 단위를 모두 소비할 가능성이 높습니다. 다시 말해서 여러 요청이 동일 파티션으로 몰리면서 용량 단위를 모두 소비하고 결국 해당 파티션에 대한 다른 요청은 병목 현상을 겪고 맙니다. 이 경우 데이터 읽기 요청을 여러 파티션으로 분산하였다면 작업으로 인한 특정 파티션의 병목 현상은 발생하지 않을 것입니다.

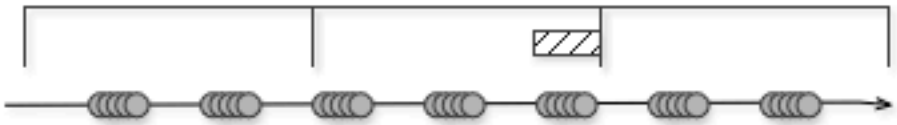
다음은 Query 및 Scan 작업에 따른 용량 단위 사용량의 급증이 미치는 영향과, 동일한 테이블에 대한 다른 요청에 미치는 영향을 나타낸 다이어그램입니다.



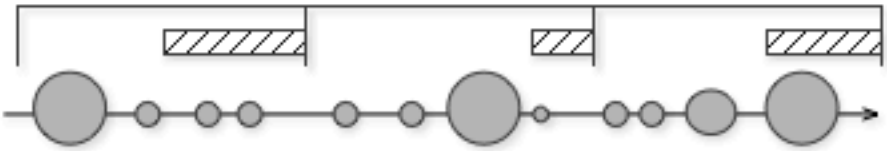

1. Good: Even distribution of requests and size



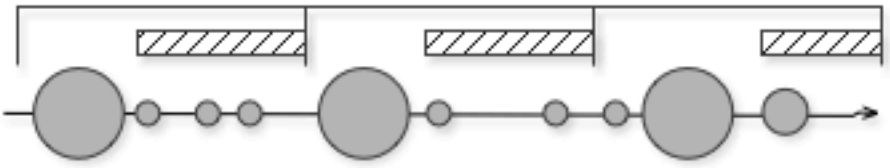
2. Not as Good: Frequent requests in bursts



3. Bad: A few random large requests



4. Bad: Large scan operations



여기에서 설명했듯, 사용량 급증은 몇몇 방식으로 테이블의 프로비저닝된 처리량에 영향을 초래할 수 있습니다.

1. 좋음: 요청과 크기를 고르게 배포

2. 중간: 버스트의 잦은 요청
3. 나쁨: 소수의 임의 대량 요청
4. 나쁨: 대형 스캔 작업

대용량의 Scan 작업 대신에 다음과 같은 기술을 사용하여 스캔이 테이블의 할당 처리량에 미치는 영향을 최소화할 수 있습니다.

- 페이지 크기 축소

  스캔 작업은 전체 페이지(기본 1MB)를 읽어오기 때문에 페이지 크기를 축소하면 스캔 작업의 영향을 줄일 수 있습니다. Scan 작업의 Limit 파라미터는 페이지 크기를 요청에 맞게 설정하는 데 사용됩니다. 각 Query 또는 Scan 요청마다 페이지 크기를 축소하면 읽기 작업 수가 줄어들면서 각 요청 간 "멈춤"이 발생합니다. 예를 들어, 각 항목이 4KB이고, 페이지 크기를 항목 40개로 설정했다고 가정합니다. Query 요청은 20개의 최종적 일관된 읽기 작업이나 40개의 강력히 일관된 읽기 작업만 사용합니다. 페이지 크기를 축소한 Query 또는 Scan 작업은 그 수를 늘리더라도 병목 현상 없이 다른 중요한 요청까지 처리할 수 있습니다.
- 스캔 작업 격리

  DynamoDB는 확장이 용이하도록 설계되었습니다. 그 결과 애플리케이션이 개별 목적으로 테이블을 생성할 뿐만 아니라 내용을 여러 테이블로 복제까지 할 수 있습니다. 예를 들어 "미션 크리티컬" 트래픽이 발생하지 않는 테이블을 스캔하려고 합니다. 일부 애플리케이션은 중요 트래픽용과 기록용의 두 테이블 간에 트래픽을 매시간 순환하여 이 부하를 처리합니다. 또 어떤 애플리케이션은 "미션 크리티컬" 테이블과 "섀도" 테이블, 두 개에서 모든 쓰기 작업을 실행하여 부하를 처리하기도 합니다.

애플리케이션은 할당 처리량을 초과했다는 응답 코드가 수신되는 요청은 모두 재시도하거나 작업으로 테이블의 프로비저닝된 처리량을 높일 수 있도록 구성합니다. 또는 UpdateTable 작업을 사용하여 테이블의 프로비저닝된 처리량을 증가시킵니다. 워크로드의 일시적인 급증으로 인해 처리량이 간혹 할당된 수준을 초과할 때는 지수 백오프로 요청을 재시도합니다. 지수 백오프의 구현에 대한 자세한 내용은 오류 재시도 횟수 및 지수 백오프 섹션을 참조하세요.

## 병렬 스캔 활용

다양한 애플리케이션이 순차식 스캔이 아닌 병렬식 Scan 작업의 이점을 이용할 수 있습니다. 예를 들어 대용량의 이력 데이터 테이블을 처리하는 애플리케이션의 경우 순차식보다는 병렬식 스캔이 훨씬 빠릅니다. 백그라운드 "스위퍼" 프로세스에서 다수의 작업자 스레드가 우선순위를 낮춰 테이블을 스캔하기 때문에 프로덕션 트래픽에 아무런 영향도 끼치지 않습니다. 이 두 가지 예에서 모두 병렬식 Scan은 다른 애플리케이션에 필요한 할당 처리량 리소스까지 소비하지는 않습니다.

이처럼 병렬식 스캔이 유용하기는 하지만 할당 처리량에 대한 수요가 지나치게 높아집니다. 병렬 스캔의 경우, 애플리케이션이 여러 작업자로 하여금 동시에 Scan 작업을 실행시킬 수 있습니다. 이렇게 하면 테이블의 프로비저닝된 읽기 용량이 빠르게 소비됩니다. 이 경우 동일한 테이블에 액세스해야 하는 다른 애플리케이션은 병목 현상이 발생하게 됩니다.

병렬식 스캔은 다음과 같은 조건만 충족된다면 올바른 선택이라고 할 수 있습니다.

- 테이블 크기가 20GB 이상입니다.
- 테이블에 프로비저닝된 읽기 처리량이 사용하고도 남습니다.
- 순차식 Scan 작업이 너무 느립니다.

### TotalSegments 선택

최적의 TotalSegments 설정 값은 특정 데이터, 테이블의 할당 처리량 설정 값, 그리고 성능 요건에 따라 결정됩니다. 올바른 설정을 위해서는 시험이 필요할 것입니다. 먼저 2GB 데이터당 한 세그먼트 같이 단순 비율부터 시작하는 것이 좋습니다. 예를 들어 테이블 크기가 30GB라면 TotalSegments를 15(30GB / 2GB)로 설정할 수 있습니다. 그러면 애플리케이션은 15명의 작업자를 사용하지만 각 작업자마다 다른 세그먼트를 스캔합니다.

그 밖에 클라이언트 리소스를 기준으로 TotalSegments 값을 선택하는 방법도 있습니다. TotalSegments를 1부터 1,000,000 중 한 숫자로 설정합니다. 그러면 DynamoDB에서 해당 개수의 세그먼트를 스캔할 수 있습니다. 예를 들어 클라이언트가 동시에 실행할 수 있는 스레드 수를 제한하는 경우 애플리케이션의 Scan 성능이 최적화될 때까지 TotalSegments를 일정하게 늘리면 됩니다.

프로비저닝된 처리량을 최적화하려면 병렬식 스캔을 모니터링해야 하지만 동시에 다른 애플리케이션에 필요한 리소스까지 소비해서는 안 됩니다. 프로비저닝된 처리량을 모두 소비하지 않는데도 여전히 Scan 요청에 병목 현상을 겪고 있다면 TotalSegments 값을 높여 설정하세요. 그리고 Scan 요청 시 할당 처리량을 의도한 것보다 많이 소비한다면 TotalSegments 값을 내려 설정하세요.

## DynamoDB 테이블 설계 모범 사례

Amazon DynamoDB의 일반 설계 원칙은 사용할 테이블의 수를 최소한으로 유지할 것을 권장합니다. 대부분의 경우 단일 테이블 사용을 고려하는 것이 좋습니다. 하지만 단일 또는 소수의 테이블 사용이 어려운 경우에는 다음 지침이 유용할 수 있습니다.

- 계정당 한도는 계정당 테이블 10,000개 이상으로 늘릴 수 없습니다. 애플리케이션에 더 많은 테이블이 필요한 경우, 테이블을 여러 계정에 배포할 계획을 세우세요. 자세한 내용은 Amazon DynamoDB의 서비스, 계정 및 테이블 할당량을 참조하세요.

- 테이블 관리에 영향을 미칠 수 있는 동시 컨트롤 플레인 작업에 대한 컨트롤 플레인 제한을 고려하세요.
- AWS 솔루션 아키텍트와 협력하여 멀티 테넌트 설계의 설계 패턴을 검증하세요.

# DynamoDB 글로벌 테이블 설계 모범 사례

글로벌 테이블은 Amazon DynamoDB의 국제적인 입지를 기반으로 구축되어 완전관리형 다중 리전 다중 활성 데이터베이스를 제공하며, 이 데이터베이스는 대규모로 확장되는 글로벌 애플리케이션의 신속한 로컬 읽기 및 쓰기 성능을 제공합니다. 글로벌 테이블을 사용하면 선택된 AWS 리전에 데이터가 자동으로 복제됩니다. 글로벌 테이블은 기존 DynamoDB API를 사용하므로 애플리케이션을 변경할 필요가 없습니다. 글로벌 테이블 사용에 따른 선결제 비용이나 약정은 없으며 사용한 리소스에 대해서만 비용을 지불합니다.

주제

- DynamoDB 글로벌 테이블 설계를 위한 규범적 지침
- DynamoDB 글로벌 테이블 설계에 대한 주요 사실
- DynamoDB 글로벌 테이블 사용 사례
- DynamoDB 글로벌 테이블을 사용한 쓰기 모드
- DynamoDB 글로벌 테이블을 사용한 요청 라우팅
- DynamoDB 글로벌 테이블을 사용한 리전 대피
- DynamoDB 글로벌 테이블의 처리량 용량 계획
- DynamoDB 글로벌 테이블 준비 체크리스트 및 자주 묻는 질문

## DynamoDB 글로벌 테이블 설계를 위한 규범적 지침

글로벌 테이블을 효율적으로 사용하려면 선호하는 쓰기 모드, 라우팅 모델, 대피 프로세스와 같은 요소를 신중하게 고려해야 합니다. 글로벌 상태를 유지하기 위해서는 모든 리전에서 애플리케이션을 계측하고 라우팅을 조정하거나 대피를 수행할 준비가 되어 있어야 합니다. 그 보상으로 읽기 및 쓰기 지연 시간이 짧고 99.999% SLA(서비스 수준 계약)를 갖춘 전 세계에 분산된 데이터 세트를 보유할 수 있습니다.

# DynamoDB 글로벌 테이블 설계에 대한 주요 사실

- 글로벌 테이블에는 현재 버전인 글로벌 테이블 버전 2019.11.21(현재)('V2'라고도 함)과 글로벌 테이블 버전 2017.11.29(레거시)('V1'이라고도 함)의 두 가지 버전이 있습니다. 이 가이드에서는 현재 버전인 V2에만 초점을 맞춥니다.
- 글로벌 테이블을 사용하지 않는 DynamoDB는 리전 서비스입니다. 이 서비스는 가용성이 높으며, 전체 가용 영역(AZ) 장애를 포함한 리전의 인프라 장애에 대한 내재적 복원력이 뛰어납니다. 단일 리전 DynamoDB 테이블에는 99.99%의 가용성 https://aws.amazon.com/dynamodb/sla/서비스 수준 계약(SLA)이 적용됩니다.
- 글로벌 테이블을 사용하면 DynamoDB에서 테이블이 둘 이상의 리전 간에 데이터를 복제할 수 있습니다. 다중 리전 DynamoDB 테이블에는 99.999%의 가용성 SLA가 있습니다. 적절한 계획을 세우면 글로벌 테이블은 복원력이 뛰어나고 리전 장애에 강한 아키텍처를 만드는 데 도움이 될 수 있습니다.
- 글로벌 테이블은 액티브-액티브 복제 모델을 사용합니다. DynamoDB의 관점에서 볼 때 각 리전의 테이블은 읽기 및 쓰기 요청을 수락할 수 있는 동등한 지위를 갖습니다. 로컬 복제본 테이블은 쓰기 요청을 받은 후 백그라운드에서 다른 참여 리전에 쓰기를 복제합니다.
- 항목은 개별적으로 복제됩니다. 단일 트랜잭션 내에서 업데이트된 항목은 함께 복제되지 않을 수 있습니다.
- 소스 리전의 각 테이블 파티션은 다른 모든 파티션과 병렬로 쓰기를 복제합니다. 원격 리전 내의 쓰기 순서는 소스 리전 내에서 발생한 쓰기 순서와 일치하지 않을 수 있습니다. 테이블 파티션에 대한 자세한 내용은 블로그 게시물 Scaling DynamoDB: How partitions, hot keys, and split for heat impact performance를 참조하세요.
- 글로벌 테이블에 새로 쓰여진 항목은 일반적으로 1초 안에 모든 복제본 테이블에 전파됩니다. 가까운 리전이 대체로 더 빨리 전파됩니다.
- Amazon CloudWatch는 각 리전 페어의 `ReplicationLatency` 지표를 제공합니다. 이 지표는 도착하는 항목을 보고 도착 시간을 최초 쓰기 시간과 비교하여 평균을 내 계산됩니다. 타이밍은 소스 리전의 CloudWatch 내에 저장됩니다. 평균 및 최대 타이밍을 보면 평균 복제 지연과 최악의 경우 복제 지연을 파악하는 데 도움이 될 수 있습니다. 이 지연 시간에는 SLA가 없습니다.
- 동일한 항목이 서로 다른 두 리전에서 거의 같은 시간에(이 `ReplicationLatency` 기간 내에서) 업데이트되고 첫 번째 쓰기가 복제되기 전에 두 번째 쓰기가 발생하는 경우, 쓰기 충돌이 발생할 수 있습니다. 글로벌 테이블은 쓰기 타임스탬프를 기반으로 최종 쓰기 우선 메커니즘을 사용하여 이러한 충돌을 해결합니다. 첫 번째 쓰기는 두 번째 쓰기에 우선순위가 밀립니다. 이러한 충돌은 CloudWatch 또는 AWS CloudTrail에 기록되지 않습니다.
- 각 항목에는 비공개 시스템 속성으로 보관되는 마지막 쓰기 타임스탬프가 있습니다. 최종 쓰기 우선 접근 방식은 수신 항목의 타임스탬프가 기존 항목의 타임스탬프보다 커야 하는 조건부 쓰기를 사용하여 구현됩니다.

- 글로벌 테이블은 모든 항목을 모든 참여 리전에 복제합니다. 다른 복제 범위를 지정하려는 경우, 다른 테이블을 생성하고 각 테이블에 서로 다른 참여 리전을 지정할 수 있습니다.
- 복제본 리전이 오프라인 상태이거나 `ReplicationLatency`가 증가하더라도 로컬 리전에 대한 쓰기가 허용됩니다. 로컬 테이블은 각 항목이 성공할 때까지 원격 테이블에 항목 복제를 계속 시도합니다.
- 드문 경우이긴 하지만 리전이 완전히 오프라인 상태가 되었다가 나중에 온라인 상태가 되면 보류 중인 아웃바운드 및 인바운드 복제가 모두 재시도됩니다. 테이블을 다시 동기화하기 위한 특별한 작업은 필요하지 않습니다. 최종 쓰기 우선 메커니즘은 최종적인 데이터 일관성을 보장합니다.
- 언제든지 DynamoDB 테이블에 새 리전을 추가할 수 있습니다. DynamoDB는 초기 동기화와 진행 중인 복제를 처리합니다. 원래 리전이더라도 리전이 제거되면 해당 리전의 테이블만 삭제됩니다.
- DynamoDB에는 글로벌 엔드포인트가 없습니다. 모든 요청은 리전 엔드포인트로 전송되며, 이 엔드포인트는 해당 리전에 로컬인 글로벌 테이블 인스턴스에 액세스합니다.
- DynamoDB에 대한 호출이 리전을 넘나들어서는 안 됩니다. 모범 사례는 어느 한 리전의 컴퓨팅 계층이 해당 리전의 로컬 DynamoDB 엔드포인트에만 직접 액세스하는 것입니다. 문제가 DynamoDB 계층에 있든 주변 스택에 있든 관계없이 리전 내에서 문제가 감지되면 최종 사용자 트래픽을 다른 리전에서 호스팅되는 다른 컴퓨팅 계층으로 라우팅해야 합니다. 글로벌 테이블 복제 덕에 로컬에서 작업할 수 있는 동일한 데이터의 로컬 사본이 다른 리전에 이미 있게 됩니다. 경우에 따라 한 리전의 컴퓨팅 계층이 처리를 위해 다른 리전의 컴퓨팅 계층으로 요청을 전달할 수 있지만 이 계층이 원격 DynamoDB 엔드포인트에 직접 액세스해서는 안 됩니다. 이 특정 사용 사례에 대한 자세한 내용은 컴퓨팅 계층 요청 라우팅을 참조하세요.

## DynamoDB 글로벌 테이블 사용 사례

글로벌 테이블은 다음과 같은 일반적인 이점을 제공합니다.

- 읽기 지연 시간 단축. 데이터 사본을 최종 사용자 가까이 두어 읽기 중 네트워크 지연 시간을 줄일 수 있습니다. 캐시는 `ReplicationLatency` 값만큼 최신 상태로 유지됩니다.
- 쓰기 지연 시간 단축. 가까운 리전에 쓰기를 수행하여 네트워크 지연 시간과 쓰기에 걸리는 시간을 줄일 수 있습니다. 충돌이 발생하지 않도록 쓰기 트래픽을 신중하게 라우팅해야 합니다. 라우팅 기법은 DynamoDB 글로벌 테이블을 사용한 요청 라우팅에서 자세히 설명합니다.
- 복원력 및 재해 복구 향상. 리전에 성능 저하 또는 완전 중단이 발생하는 경우, 초 단위로 측정되는 Recovery Point Objective(RPO)와 Recovery Time Objective(RTO)를 사용하여 리전을 대피(해당 리전으로 전송되는 요청의 일부 또는 전부를 이동)시킬 수 있습니다. 또한 글로벌 테이블을 사용하면 DynamoDB SLA가 99.99%에서 99.999%로 증가합니다.

- 원활한 리전 마이그레이션. 새 리전을 추가한 다음 이전 리전을 삭제하여 데이터 계층의 가동 중단 없이 한 리전에서 다른 리전으로 배포를 마이그레이션할 수 있습니다. 예를 들어 DynamoDB 글로벌 테이블을 주문 관리 시스템에 사용하면 안정적으로 지연 시간이 짧은 대규모 처리가 가능하며 동시에 AZ 및 리전 장애에 대한 복원력을 유지할 수 있습니다.

## DynamoDB 글로벌 테이블을 사용한 쓰기 모드

글로벌 테이블은 테이블 수준에서 항상 액티브-액티브입니다. 그러나 쓰기 요청을 라우팅하는 방법을 제어하여 이를 액티브-패시브로 처리하는 것이 좋을 경우가 있습니다. 예를 들어 잠재적인 쓰기 충돌을 방지하기 위해 쓰기 요청을 단일 리전으로 라우팅하기로 결정할 수 있습니다.

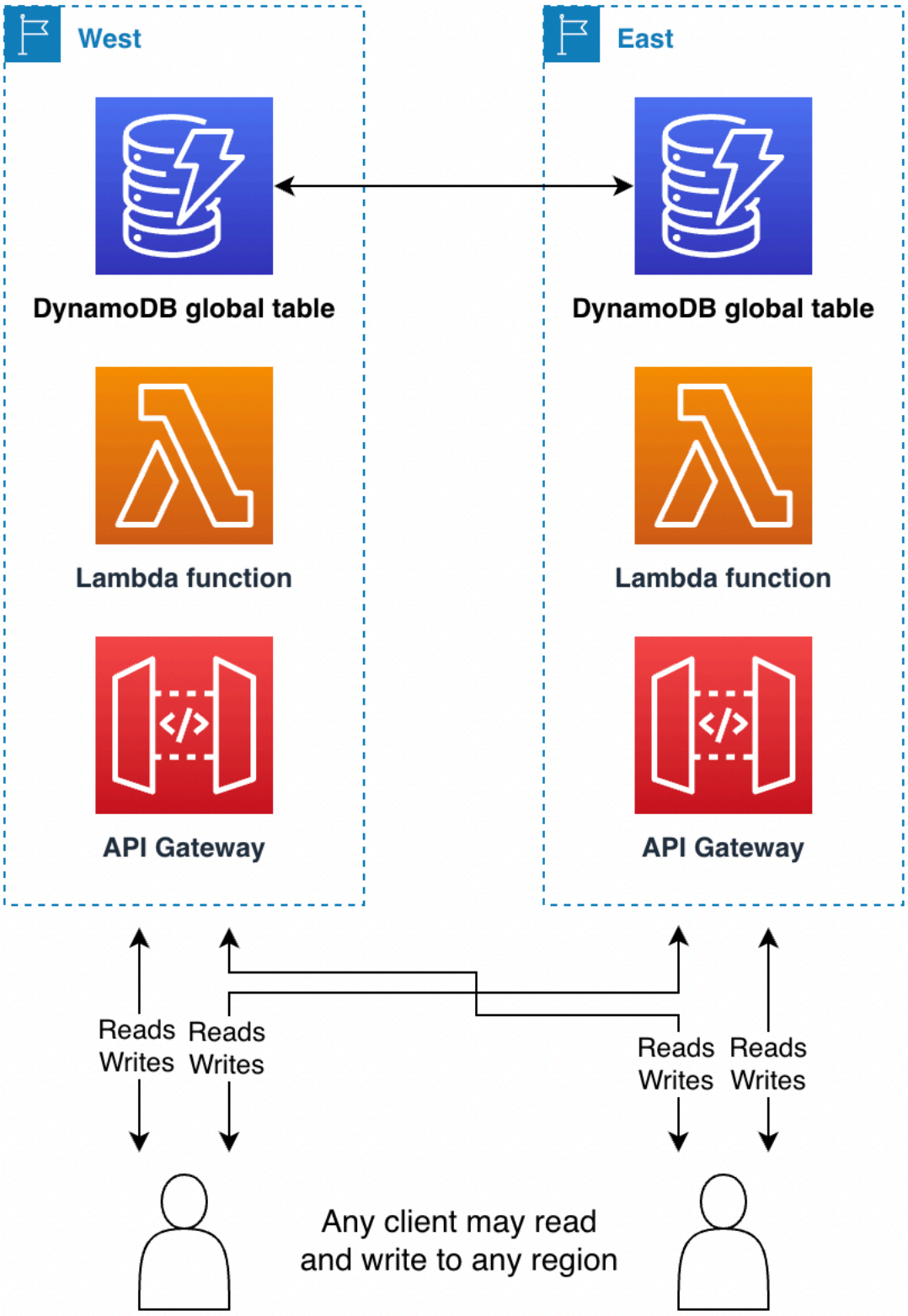
관리 쓰기 패턴에는 다음과 같은 세 가지 주요 범주가 있습니다.

- 임의의 리전에 쓰기 모드(기본 리전 없음)
- 한 리전에 쓰기 모드(단일 기본 리전)
- 사용자 리전에 쓰기 모드(혼합 기본 리전)

어떤 쓰기 패턴이 사용 사례에 적합한지 고려해야 합니다. 이 선택은 요청 라우팅, 리전 대피, 재해 복구 처리 방법에 영향을 미칩니다. 애플리케이션의 쓰기 모드에 따라 전반적인 모범 사례가 달라질 수 있습니다.

### 임의의 리전에 쓰기 모드(기본 리전 없음)

임의의 리전에 쓰기 모드는 완전한 액티브-액티브이며, 쓰기가 발생할 수 있는 위치에 제한을 두지 않습니다. 어느 리전이나 언제든지 쓰기를 수락할 수 있습니다. 가장 간단한 모드입니다. 이 모드는 일부 애플리케이션 유형에만 사용할 수 있습니다. 모든 라이터가 멱등적이고 따라서 안전하게 반복할 수 있어 여러 리전에서 동시 또는 반복 쓰기 작업이 충돌하지 않는 경우에 적합합니다. 사용자가 연락처 데이터를 업데이트하는 경우를 예로 들 수 있습니다. 이 모드는 모든 쓰기가 결정적 프라이머리 키 아래의 고유한 삽입인 멱등적 추가 전용 데이터 세트 같은 특수한 경우에도 효과적입니다. 마지막으로 이 모드는 쓰기 충돌 위험이 허용 가능한 수준인 경우에 적합합니다.


West
East
DynamoDB global table
DynamoDB global table
Lambda function
Lambda function
API Gateway
API Gateway
Reads
Writes
Reads
Writes
Reads
Writes
Reads
Writes
Any client may read
and write to any region

임의의 리전에 쓰기 모드는 가장 간단하게 구현할 수 있는 아키텍처입니다. 어느 리전이나 언제든지 쓰기 대상이 될 수 있으므로 라우팅이 더 쉽습니다. 원하는 횟수만큼 최근 쓰기를 보조 리전에 재생할 수 있으므로 장애 조치가 더 쉽습니다. 가능하면 이 쓰기 모드에 맞춰 설계해야 합니다.

예를 들어 비디오 스트리밍 서비스는 북마크, 리뷰, 시청 상태 플래그 등을 추적하기 위해 글로벌 테이블을 사용하는 경우가 많습니다. 이러한 배포에서는 모든 쓰기가 멱등적이고 항목의 다음 올바른 값이 현재 값에 좌우되지 않는 한 임의의 리전에 쓰기 모드를 사용할 수 있습니다. 새로운 최신 타임 코드 설정, 새 리뷰 할당, 새 시계 상태 설정과 같이 사용자의 새 상태를 직접 할당하는 사용자 업데이트가 여기 해당합니다. 사용자의 쓰기 요청이 다른 리전으로 라우팅되는 경우 마지막 쓰기 작업이 지속되고 글로벌 상태는 마지막 할당에 따라 결정됩니다. 이 모드에서의 읽기 작업은 최신 `ReplicationLatency` 값만큼 지연된 후 최종적 일관성을 갖게 됩니다.

또 다른 예로 한 금융 서비스 회사는 시스템의 일부로 글로벌 테이블을 사용하여 각 고객의 직불 카드 구매 현황을 지속적으로 집계하여 해당 고객의 캐시백 보상을 계산합니다. 신규 거래는 전 세계에서 유입되어 여러 리전으로 이동합니다. 글로벌 테이블을 활용하지 않는 현재 설계에서는 고객당 하나의 Running Balance 항목을 사용합니다. 고객 작업은 ADD 식으로 잔액을 업데이트합니다. 올바른 새 값은 현재 값에 따라 달라지기 때문에 이 식은 멱등성을 가지지 않습니다. 즉, 서로 다른 리전에서 거의 동시에 동일한 잔액에 대한 쓰기 작업이 두 번 발생하면 잔액이 동기화되지 않습니다.

이 회사는 DynamoDB의 글로벌 테이블을 사용한 신중한 재설계를 통해 임의의 리전에 쓰기 모드를 달성할 수 있었습니다. 새로운 설계는 기본적으로 추가 전용 워크플로가 있는 원장인 '이벤트 스트리밍' 모델을 따를 수 있었습니다. 각 고객 작업은 해당 고객을 위해 유지 관리되는 항목 컬렉션에 새 항목을 추가합니다. 항목 컬렉션은 프라이머리 키를 공유하고 정렬 키가 서로 다른 항목 세트입니다. 고객 작업을 추가하는 각 쓰기 작업은 고객 ID를 파티션 키로, 트랜잭션 ID를 정렬 키로 사용하는 멱등적 삽입입니다. 이 설계에서는 항목을 가져온 다음 클라이언트 측 계산을 수행하기 위해 `Query`가 필요하기 때문에 잔액 계산이 더 복잡해집니다. 하지만 모든 쓰기가 멱등적이 되어 라우팅과 장애 조치가 크게 단순화된다는 이점이 있습니다. 자세한 내용은 DynamoDB 글로벌 테이블을 사용한 요청 라우팅을 참조하세요.

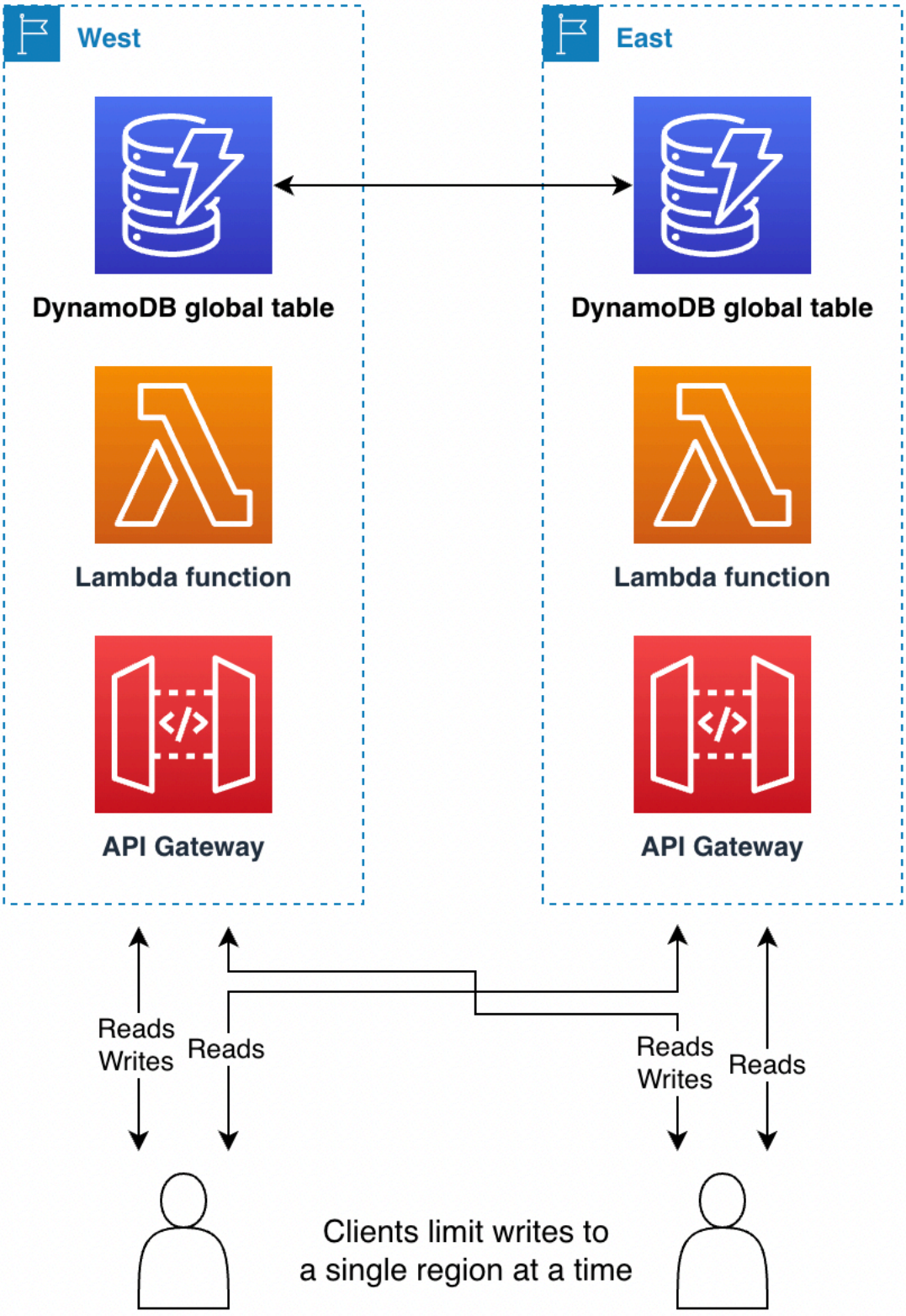
세 번째 예로 온라인 광고를 게재하는 고객이 있다고 가정해 보겠습니다. 이 고객은 임의의 리전에 쓰기 모드의 설계 단순화를 위해 낮은 데이터 손실 위험을 감수할 수 있다고 결정했습니다. 이 고객이 광고를 게재할 때는 어떤 광고를 표시할지 결정하기 위해 충분한 메타데이터를 검색한 다음 사용자에게 동일한 광고가 반복되지 않도록 광고 노출을 기록할 시간이 몇 밀리초에 불과합니다. 글로벌 테이블을 사용하면 전 세계 최종 사용자의 짧은 읽기 지연 시간과 짧은 쓰기 지연 시간을 모두 얻을 수 있습니다. 사용자의 모든 광고 노출을 단일 항목 내에 기록하고 이를 증가하는 목록으로 표시할 수 있습니다. 항목 컬렉션에 추가하는 대신 하나의 항목을 사용할 수 있습니다. 이렇게 하면 삭제 비용을 지불하지 않고도 각 쓰기의 일부로 오래된 광고 노출을 제거할 수 있기 때문입니다. 이 쓰기 작업은 멱등적이지 않으므로 동일한 최종 사용자가 거의 동시에 여러 리전에서 게재되는 광고를 보게 되면 한 광고 노출 쓰

기가 다른 광고 노출 쓰기를 덮어쓸 가능성이 있습니다. 온라인 광고 게재의 경우 반복되는 광고를 사용자가 가끔 보게 될 위험은 이 더 단순하고 효율적인 설계를 위해 감수할 만합니다.

## 단일 기본("한 리전에 쓰기")

한 리전에 쓰기 모드는 액티브-패시브이며, 모든 테이블 쓰기를 단일 활성 리전으로 라우팅합니다. 참고로 DynamoDB에는 단일 활성 리전이라는 개념이 없습니다. DynamoDB 외부의 애플리케이션 라우팅이 이를 관리합니다. 한 리전에 쓰기 모드는 한 번에 한 리전으로만 쓰기가 흐르도록 하여 쓰기 충돌을 피합니다. 이 쓰기 모드는 조건식이나 트랜잭션을 사용하고자 할 때 유용합니다. 조건식이나 트랜잭션은 최신 데이터를 기준으로 작업하고 있다는 것을 알지 못하면 작동하지 않기 때문입니다. 따라서 조건식과 트랜잭션을 사용하려면 모든 쓰기 요청을 최신 데이터가 포함된 하나의 리전으로 보내야 합니다.

최종 읽기 일관성은 지연 시간을 줄이기 위해 어떤 복제본 리전으로도 갈 수 있습니다. 강력히 일관된 읽기는 단일 기본 리전으로 가야 합니다.


West
East
DynamoDB global table
DynamoDB global table
Lambda function
Lambda function
API Gateway
API Gateway
Reads
Writes
Reads
Reads
Writes
Reads
Clients limit writes to
a single region at a time

리전 장애에 대응하여 데이터 활용을 위해 활성 리전을 변경해야 하는 경우가 있습니다. DynamoDB 글로벌 테이블을 사용한 리전 대피는 이 사용 사례의 한 예입니다. 일부 고객은 'follow-the-sun' 배포처럼 정기적인 일정에 따라 현재 활성 리전을 변경할 것입니다. 이렇게 하면 활동이 가장 많은 리전 가까이에 활성 리전이 배치되어 읽기 및 쓰기 지연 시간이 가장 짧아집니다. 또한 매일 리전을 변경하는 코드 경로를 호출하여 재해 복구 전에 제대로 테스트할 수 있다는 부수적인 이점도 있습니다.

비활성 리전은 DynamoDB 주위에 다운스케일된 인프라 세트를 유지할 수 있으며, 이 인프라는 활성 리전이 될 경우에만 구축됩니다. 파일럿 라이트 및 웜 스탠바이 설계에 대한 자세한 내용은 Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby를 참조하세요.

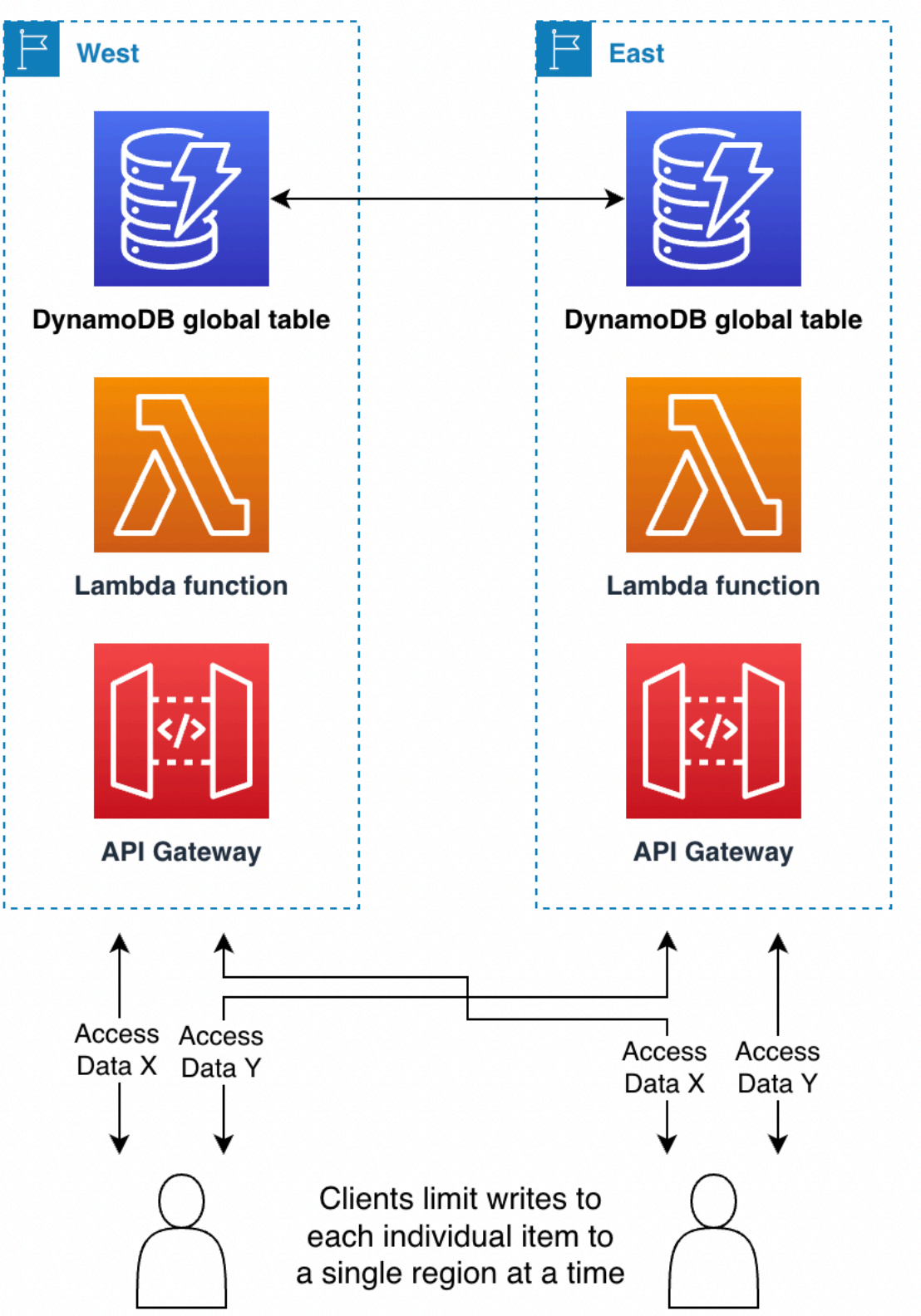
지연 시간이 짧은 글로벌 분산 읽기에 글로벌 테이블을 활용할 경우에는 한 리전에 쓰기 모드가 효과적입니다. 예를 들어 한 대형 소셜 미디어 회사는 수백만 명의 사용자와 수십억 개의 게시물을 보유하고 있습니다. 각 사용자는 계정 생성 시 자신의 위치와 지리적으로 가까운 리전에 할당됩니다. 이 비글로벌 테이블에 사용자의 모든 데이터가 들어갑니다. 이 회사는 한 리전에 쓰기 모드를 사용하여 별도의 글로벌 테이블로 사용자와 사용자 홈 리전의 매핑을 보관합니다. 전 세계에 읽기 전용 사본이 유지되므로 추가 지연 시간을 최소화하면서 각 사용자의 데이터를 직접 찾을 수 있습니다. 업데이트는 드물며 (사용자의 홈 리전을 다른 리전으로 옮길 때만) 쓰기 충돌이 발생하지 않도록 쓰기는 항상 한 리전을 거칩니다.

또 다른 예로 일일 캐시백 계산을 구현한 금융 서비스 고객을 살펴보겠습니다. 이 회사는 잔액을 계산할 때는 임의의 리전에 쓰기 모드를 사용하지만 실제 캐시백 지급액을 추적할 때는 한 리전에 쓰기 모드를 사용합니다. 하루 10달러 지출마다 1페니를 고객에게 보상하려면 전날의 모든 거래를 `Query`하고, 총 지출을 계산하고, 캐시백 결정을 새 테이블에 쓰고, 쿼리된 항목 세트를 삭제하여 소비된 것으로 표시하고, 다음 날 계산에 포함되어야 하는 나머지 금액을 저장하는 단일 항목으로 교체해야 합니다. 이 작업에는 트랜잭션이 필요하므로 한 리전에 쓰기 모드가 더 효과적입니다. 워크로드가 중복될 가능성이 없는 한 애플리케이션은 동일한 테이블에서도 쓰기 모드를 혼합할 수 있습니다.

## 혼합 기본('사용자 리전에 쓰기')

사용자 리전에 쓰기 모드는 리전마다 서로 다른 데이터 하위 집합을 할당하고 홈 리전을 통해서만 항목에 쓰기 작업을 허용합니다. 이 모드는 액티브-패시브이지만 항목을 기준으로 활성 리전을 할당합니다. 모든 리전은 중복되지 않는 자체 데이터 세트가 있는 기본 리전이며, 적절한 위치를 보장하기 위해 쓰기를 보호해야 합니다.

이 모드는 각 최종 사용자에 연결된 데이터를 해당 사용자와 가까운 네트워크에 배치할 수 있어 쓰기 지연 시간을 줄일 수 있다는 점을 제외하면 한 리전에 쓰기 모드와 비슷합니다. 또한 리전 간에 주변 인프라가 더 균등하게 분산되며, 모든 리전에 인프라의 일부가 이미 활성화되어 있기 때문에 장애 조치 시나리오 중에 인프라를 구축하는 데 드는 작업이 줄어듭니다.




항목의 홈 리전은 다음과 같은 다양한 방법으로 결정할 수 있습니다.

- 내재적: 데이터의 일부 측면(예: 파티션 키)이 홈 리전을 명확히 보여 줍니다. 예를 들어 고객 데이터 내에서 고객과 해당 고객에 대한 모든 데이터의 홈 리전이 특정 리전이라고 표시할 수 있습니다. 이 기법은 Use Region pinning to set a home Region for items in an Amazon DynamoDB global table에 설명되어 있습니다.
- 협상됨: 각 데이터 세트의 홈 리전은 할당을 관리하는 별도의 글로벌 서비스와 협상하는 것과 같이 외부적 방식으로 협상됩니다. 할당 기간이 한정될 수 있으며, 그 이후에는 재협상해야 합니다.
- 테이블 지향: 하나의 복제 글로벌 테이블 대신 복제 리전 수만큼의 글로벌 테이블을 보유하는 것입니다. 각 테이블의 이름은 해당 테이블의 홈 리전을 나타냅니다. 표준 운영에서는 모든 데이터를 홈 리전에 쓰고 다른 리전들은 읽기 전용 사본을 유지합니다. 장애 조치 중에는 해당 테이블의 쓰기 의무를 다른 리전이 일시적으로 입양합니다.

예를 들어 게임 회사에서 일하고 있다고 가정해 봅시다. 전 세계 모든 게이머의 읽기 및 쓰기 지연 시간이 짧아야 합니다. 각 게이머와 가장 가까운 리전을 홈 리전으로 할당할 수 있습니다. 이 리전이 해당 게이머의 모든 읽기 및 쓰기를 처리하므로 항상 강력한 쓰기 후 읽기 일관성이 유지됩니다. 하지만 해당 게이머가 여행 중이거나 홈 리전에 중단이 발생하는 경우, 대체 리전에서 완전한 데이터 사본을 사용할 수 있습니다. 따라서 게이머를 다른 홈 리전에 할당하는 것이 유용할 수 있습니다.

또 다른 예로 화상 회의 회사에서 일하고 있다고 가정해 보겠습니다. 각 전화 회의의 메타데이터는 특정 리전에 할당됩니다. 발신자는 가장 가까운 리전을 사용하여 지연 시간을 최소화할 수 있습니다. 리전 중단이 발생하는 경우, 글로벌 테이블을 사용하면 시스템이 통화 처리를 이미 데이터의 복제된 사본이 있는 다른 리전으로 이동할 수 있으므로 빠른 복구가 가능합니다.

## DynamoDB 글로벌 테이블을 사용한 요청 라우팅

글로벌 테이블 배포에서 가장 복잡한 부분은 아마도 요청 라우팅 관리일 것입니다. 요청은 먼저 최종 사용자에서 출발하여 어떤 방식을 통해 선택 및 라우팅되는 리전으로 가야 합니다. 요청은 AWS Lambda 함수, 컨테이너 또는 Amazon Elastic Compute Cloud(Amazon EC2) 노드가 지원하는 로드 밸런서와 다른 데이터베이스를 비롯한 기타 서비스로 구성된 컴퓨팅 계층을 포함한 해당 리전의 서비스 스택과 만나게 됩니다. 이 컴퓨팅 계층은 DynamoDB와 통신합니다. 그러려면 해당 리전의 로컬 엔드포인트를 사용해야 합니다. 글로벌 테이블의 데이터는 다른 모든 참여 리전에 복제되며, 각 리전의 DynamoDB 테이블에는 비슷한 서비스 스택이 있습니다.

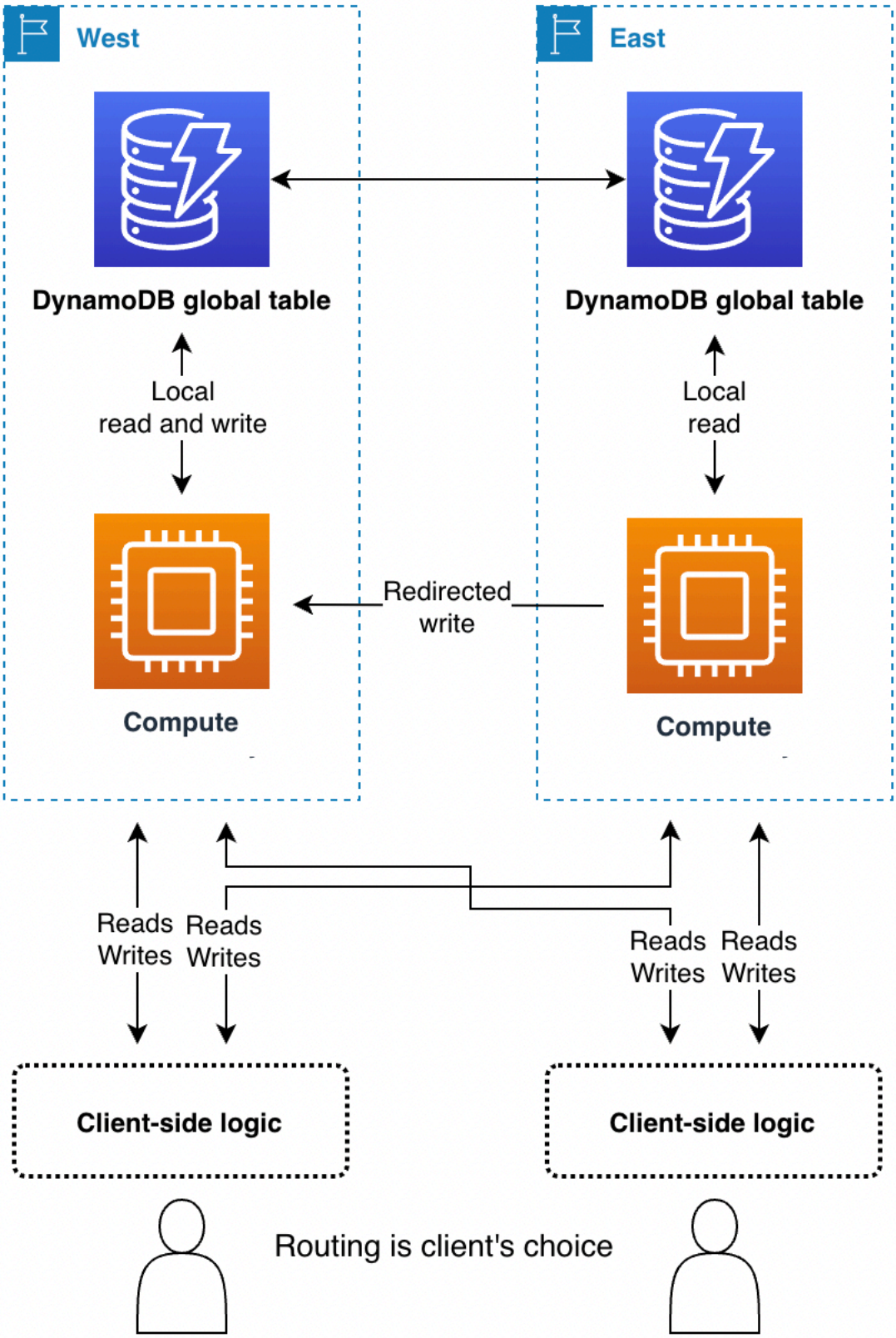
글로벌 테이블은 다양한 리전의 각 스택에 동일한 데이터의 로컬 사본을 제공합니다. 단일 리전의 단일 스택을 상정한 설계를 고려하여 로컬 DynamoDB 테이블에 문제가 생기면 보조 리전의 DynamoDB 엔드포인트에 원격 호출을 하는 방법을 예상할 수 있습니다. 이것은 모범 사례가 아닙니다. 리전 간 이동과 관련된 지연 시간은 로컬 액세스보다 100배 더 길 수 있습니다. 5개의 요청이 오고가는 것은 로컬에서 수행할 때는 몇 밀리초가 걸리지만 전 세계를 가로지를 때는 몇 초가 걸릴 수 있습니다. 처리를 위해

최종 사용자를 다른 리전으로 라우팅하는 것이 더 좋습니다. 복원력을 보장하려면 컴퓨팅 계층과 데이터 계층의 복제를 통한 여러 리전에 걸친 복제가 필요합니다.

처리를 위해 최종 사용자 요청을 리전으로 라우팅하는 대체 기법은 매우 많습니다. 최적의 선택은 쓰기 모드와 장애 조치 고려 사항에 따라 달라집니다. 이 섹션에서는 클라이언트 기반, 컴퓨팅 계층, Route 53, Global Accelerator라는 네 가지 옵션에 대해 설명합니다.

## 클라이언트 기반 요청 라우팅

클라이언트 기반 요청 라우팅에서는 애플리케이션, JavaScript가 있는 웹 페이지, 또 다른 클라이언트 같은 최종 사용자 클라이언트가 유효한 애플리케이션 엔드포인트를 추적합니다. 이 경우에는 문자 그대로의 DynamoDB 엔드포인트가 아니라 Amazon API Gateway 같은 애플리케이션 엔드포인트입니다. 최종 사용자 클라이언트는 자체 내장 로직을 사용하여 통신할 리전을 선택합니다. 무작위 선택, 관찰된 최단 지연 시간, 관찰된 최고 대역폭 측정 또는 로컬에서 수행된 상태 확인을 기반으로 리전을 선택할 수 있습니다.


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Redirected
write
Compute
Compute
Reads
Writes
Reads
Writes
Reads
Writes
Reads
Writes
Client-side logic
Client-side logic
Routing is client's choice

클라이언트 기반 요청 라우팅의 장점은 성능 저하가 감지될 경우 실제 공용 인터넷 트래픽 상황 등에 맞춰 리전을 전환할 수 있다는 것입니다. 클라이언트는 모든 잠재적 엔드포인트를 알고 있어야 하지만 새로운 리전 엔드포인트를 시작하는 경우는 드뭅니다.

임의의 리전에 쓰기 모드에서 클라이언트는 선호하는 엔드포인트를 일방적으로 선택할 수 있습니다. 한 리전에 대한 액세스가 손상되면 클라이언트는 다른 엔드포인트로 라우팅할 수 있습니다.

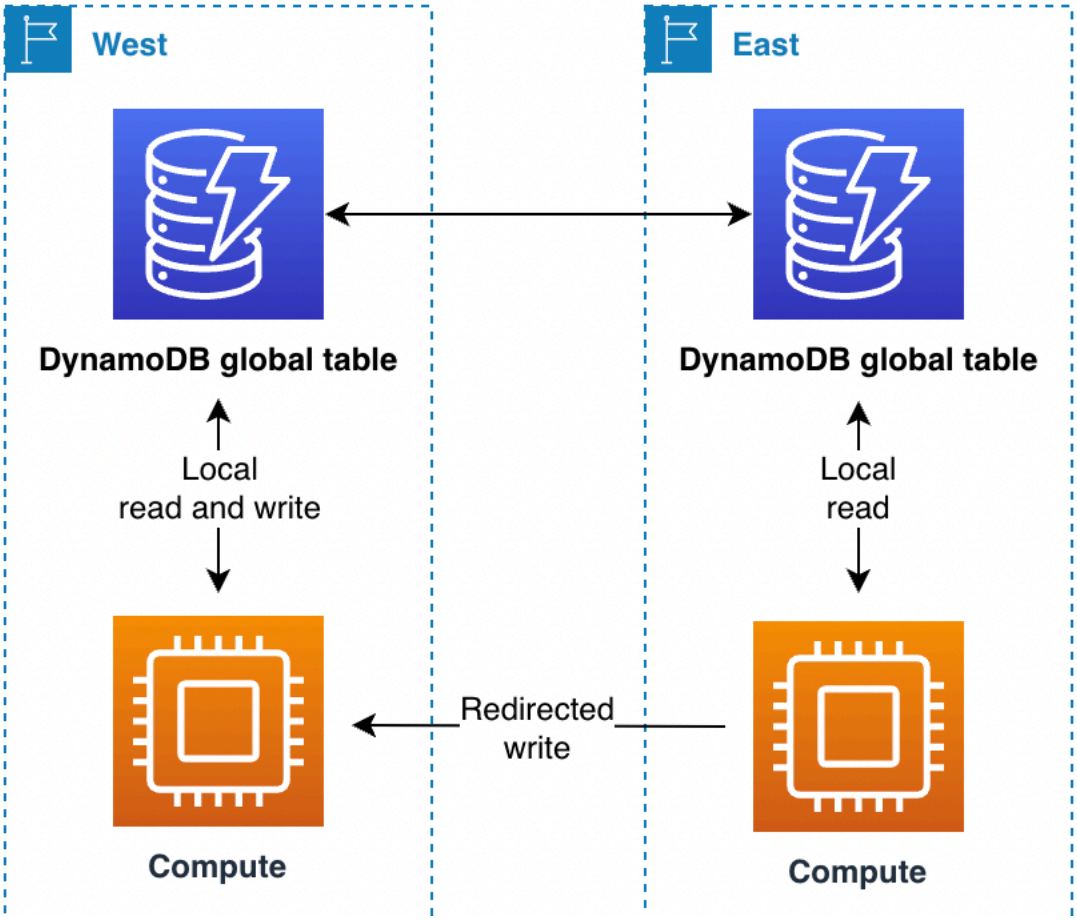
한 리전에 쓰기 모드에서 클라이언트는 쓰기를 현재 활성 리전으로 라우팅하는 메커니즘이 필요합니다. 이는 현재 어느 리전이 쓰기를 허용하고 있는지 경험적으로 테스트(쓰기 거부를 확인하고 대안으로 변경)하는 것처럼 기본적일 수도 있고, 글로벌 코디네이터를 호출하여 현재 애플리케이션 상태를 쿼리하는 것처럼 복잡할 수도 있습니다(이러한 요구 사항에 맞게 글로벌 상태를 유지하기 위해 5개 리전 쿼럼 기반 시스템을 제공하는 Route 53 애플리케이션 복구 컨트롤러(ARC) 라우팅 제어를 기반으로 구축될 수 있음). 클라이언트는 최종 일관성을 위해 읽기를 임의의 리전으로 보내도 되는지 아니면 강력한 일관성을 위해 활성 리전으로 라우팅해야 하는지 결정할 수 있습니다. 자세한 내용은 Route 53 작동 방식을 참조하세요.

사용자 리전에 쓰기 모드에서 클라이언트는 작업 중인 데이터 세트의 홈 리전을 결정해야 합니다. 예를 들어 클라이언트가 사용자 계정에 해당하고 각 사용자 계정에 홈 리전이 있는 경우, 클라이언트는 글로벌 로그인 시스템에 적절한 엔드포인트를 요청할 수 있습니다.

예를 들어 웹을 통한 사용자의 비즈니스 재무 관리를 지원하는 금융 서비스 회사는 사용자 리전에 쓰기 모드로 글로벌 테이블을 사용할 수 있습니다. 각 사용자는 중앙 서비스에 로그인해야 합니다. 이 서비스는 보안 인증 정보와 해당 보안 인증 정보가 작동하는 리전의 엔드포인트를 반환합니다. 보안 인증 정보는 짧은 기간 동안 유효합니다. 그 후 웹 페이지는 새 로그인을 자동으로 협상하여 사용자의 활동을 새 리전으로 리디렉션할 수 있는 기회를 제공합니다.

## 컴퓨팅 계층 요청 라우팅

컴퓨팅 계층 요청 라우팅의 경우, 컴퓨팅 계층에서 실행되는 코드가 요청을 로컬에서 처리할지, 다른 리전에서 실행되는 해당 코드의 사본에 전달할지 결정합니다. 한 리전에 쓰기 모드를 사용하면 컴퓨팅 계층은 리전이 활성 리전이 아님을 감지하고 로컬 읽기 작업을 허용하면서 모든 쓰기 작업을 다른 리전으로 전달할 수 있습니다. 이 컴퓨팅 계층 코드는 데이터 토폴로지 및 라우팅 규칙을 인식하고 어떤 리전이 어떤 데이터에 활성 상태인지 지정하는 최신 설정을 기반으로 이를 안정적으로 적용해야 합니다. 해당 리전 내의 외부 소프트웨어 스택은 마이크로서비스가 읽기 및 쓰기 요청을 어떻게 라우팅하는지 몰라도 됩니다. 강력한 설계에서 수신 리전은 자신이 쓰기 작업의 현재 기본 리전인지 여부를 확인합니다. 기본 리전이 아니라면 글로벌 상태를 수정해야 한다는 오류 메시지가 생성됩니다. 또한 기본 리전이 변경 중일 경우 수신 리전은 쓰기 작업을 잠시 버퍼링할 수 있습니다. 어떤 경우든 리전의 컴퓨팅 스택은 해당 로컬 DynamoDB 엔드포인트에만 쓰지만 컴퓨팅 스택은 서로 통신할 수 있습니다.

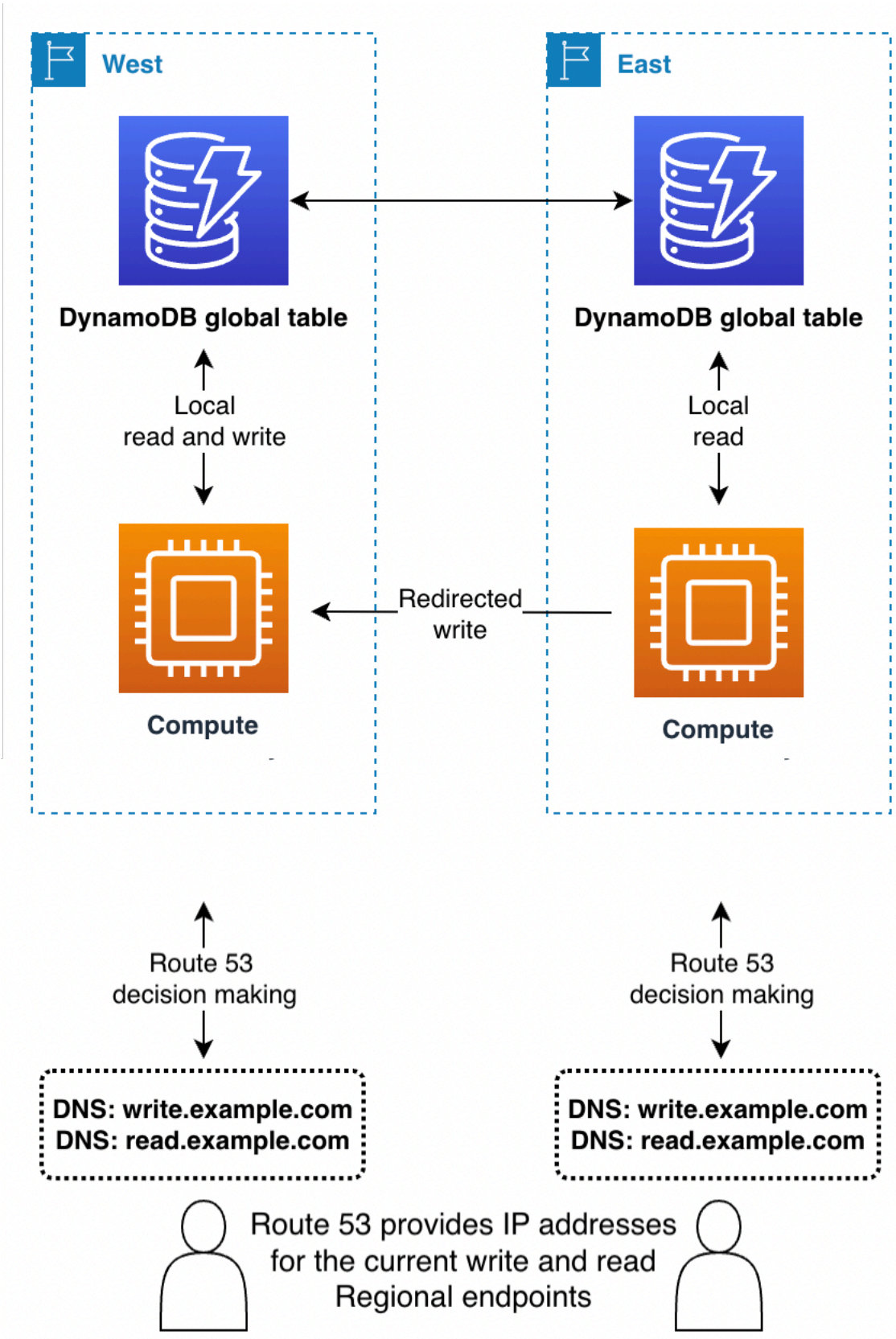



이 시나리오에서는 금융 서비스 회사가 follow-the-sun 단일 기본 모델을 사용한다고 가정해 보겠습니다. 이 회사는 이 라우팅 프로세스를 위한 시스템과 라이브러리를 사용합니다. 전체 시스템은 AWS의 ARC 라우팅 제어와 비슷하게 글로벌 상태를 유지 관리합니다. 이 회사는 글로벌 테이블을 사용하여 어느 리전이 기본 리전이고 다음 기본 리전 전환이 언제로 예약되어 있는지 추적합니다. 모든 읽기 및 쓰기 작업은 시스템에 맞게 조정되는 라이브러리를 거칩니다. 라이브러리를 통해 짧은 지연 시간으로 로컬에서 쓰기 작업을 수행할 수 있습니다. 쓰기 작업의 경우 애플리케이션은 로컬 리전이 현재 기본 리전인지 확인합니다. 현재 기본 리전인 경우 쓰기 작업이 직접 완료됩니다. 그렇지 않은 경우 라이브러리는 쓰기 작업을 현재 기본 리전에 있는 라이브러리로 전달합니다. 이 수신 라이브러리는 자신도 스스로를 기본 리전으로 간주한다는 것을 확인하고, 그렇지 않은 경우 글로벌 상태의 전파 지연을 나타내는 오류를 발생시킵니다. 이 접근 방식은 원격 DynamoDB 엔드포인트에 직접 쓰지 않으므로 검증에 도움이 됩니다.

## Route 53 요청 라우팅

Amazon Application Recovery Controller(ARC)는 도메인 이름 서비스(DNS) 기술입니다. Route 53에서 클라이언트는 잘 알려진 DNS 도메인 이름을 조회하여 엔드포인트를 요청하고, Route 53은 가장 적

절하다고 판단되는 리전 엔드포인트에 해당하는 IP 주소를 반환합니다. Route 53에는 적절한 리전을 결정하는 데 사용하는 라우팅 정책 목록이 있습니다. 또한 Route 53은 장애 조치 라우팅을 수행하여 상태 확인에 실패한 리전 외부로 트래픽을 라우팅할 수 있습니다.


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Redirected
write
Compute
Compute
Route 53
decision making
Route 53
decision making
DNS: write.example.com
DNS: read.example.com
DNS: write.example.com
DNS: read.example.com
Route 53 provides IP addresses
for the current write and read
Regional endpoints

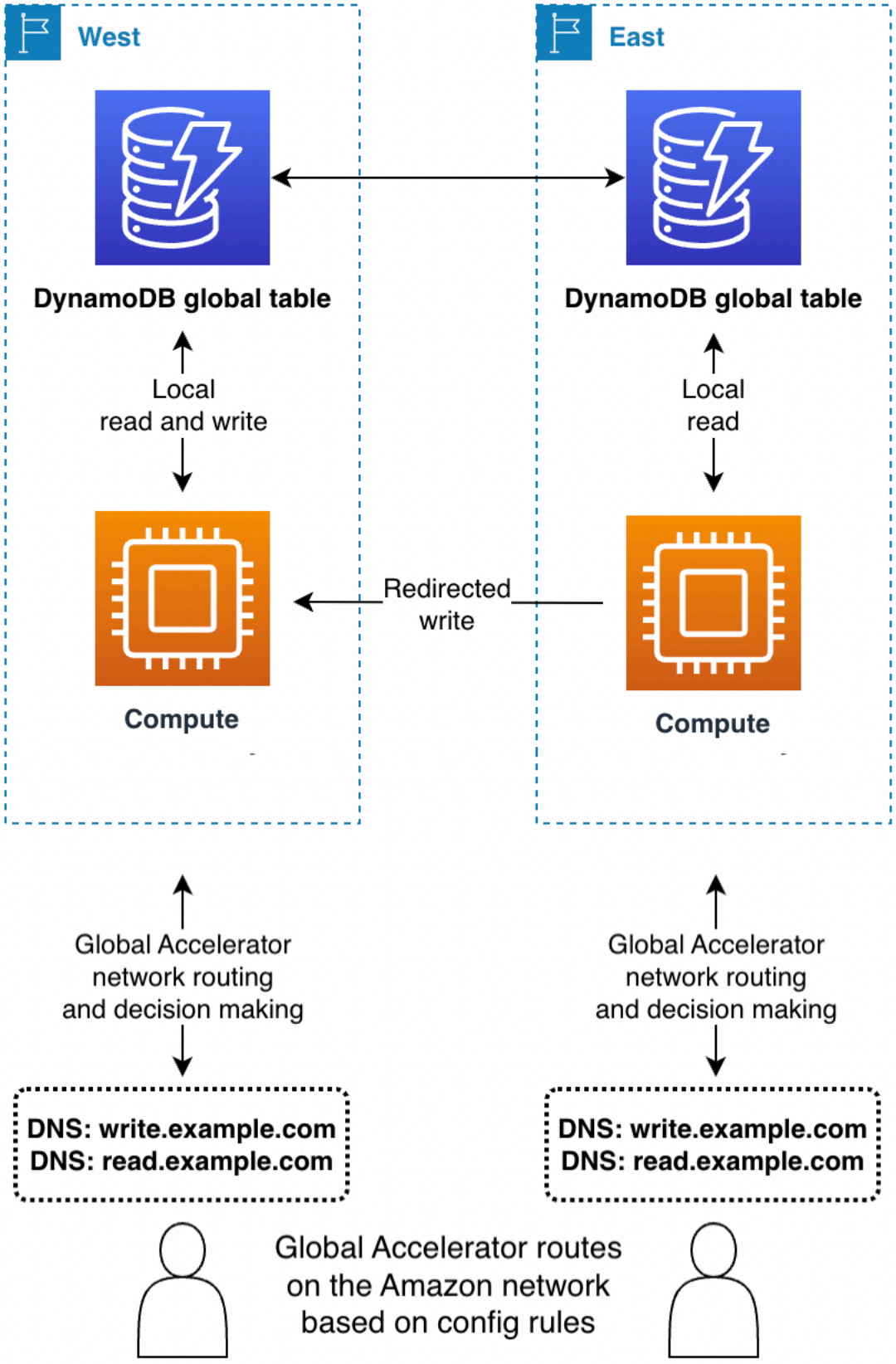
- 임의의 리전에 쓰기 모드를 사용하거나 백엔드의 컴퓨팅 계층 요청 라우팅과 결합될 경우, 네트워크와 가장 가까운 리전 또는 지리적으로 가장 가까운 리전 또는 기타 선택 등 복잡한 내부 규칙에 기반하여 리전을 반환할 수 있는 전체 액세스 권한을 Route 53에 부여할 수 있습니다.
- 한 리전에 쓰기 모드를 사용하면 현재 활성 리전을 반환하도록 Route 53을 구성할 수 있습니다 (Route 53 ARC 사용).

> **Note**
>
> 클라이언트는 Route 53의 응답에 있는 IP 주소를 도메인 이름에 TTL(Time to Live) 설정으로 표시된 시간 동안 캐시합니다. TTL이 길수록 모든 클라이언트가 새 엔드포인트를 인식할 수 있는 Recovery Time Objective(RTO)가 연장됩니다. 일반적으로 장애 조치에는 60초 값을 사용합니다. 모든 소프트웨어가 DNS TTL 만료를 완벽하게 준수하는 것은 아닙니다.

- 사용자 리전에 쓰기 모드에서는 컴퓨팅 계층 요청 라우팅도 사용하지 않는 한 Route 53을 사용하지 않는 것이 가장 좋습니다.

## Global Accelerator 요청 라우팅

클라이언트는 AWS Global Accelerator를 사용하여 잘 알려진 도메인 이름을 Route 53에서 조회합니다. 하지만 클라이언트는 리전 엔드포인트에 해당하는 IP 주소를 돌려받는 대신 가장 가까운 AWS 엣지 로케이션으로 라우팅되는 애니캐스트 고정 IP 주소를 받습니다. 모든 트래픽은 이 엣지 로케이션에서 출발하여 프라이빗 AWS 네트워크에서 Global Accelerator 내에 유지되는 라우팅 규칙에 따라 선택된 리전의 일부 엔드포인트(예: 로드 밸런서 또는 API Gateway)로 라우팅됩니다. Route 53 규칙에 기반한 라우팅과 비교할 때 Global Accelerator 요청 라우팅은 공용 인터넷의 트래픽 양을 줄이므로 지연 시간이 짧습니다. 또한 Global Accelerator는 라우팅 규칙을 변경할 때 DNS TTL 만료에 의존하지 않으므로 라우팅을 더 빠르게 조정할 수 있습니다.


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Redirected
write
Compute
Compute
Global Accelerator
network routing
and decision making
Global Accelerator
network routing
and decision making
DNS: write.example.com
DNS: read.example.com
DNS: write.example.com
DNS: read.example.com
Global Accelerator routes
on the Amazon network
based on config rules

- Global Accelerator는 임의의 리전에 쓰기 모드를 사용하거나 백엔드의 컴퓨팅 계층 요청 라우팅과 결합할 경우 원활하게 작동합니다. 클라이언트는 가장 가까운 엣지 로케이션에 연결하며, 요청을 받는 리전에 신경 쓸 필요가 없습니다.
- 한 리전에 쓰기를 사용하는 경우 Global Accelerator 라우팅 규칙은 현재 활성 리전으로 요청을 보내야 합니다. 글로벌 시스템에서 활성 리전으로 간주되지 않는 리전의 장애를 인위적으로 보고하는 상태 확인을 사용할 수 있습니다. DNS와 마찬가지로 요청이 어느 리전에서나 올 수 있다면 대체 DNS 도메인 이름을 사용하여 읽기 요청을 라우팅할 수 있습니다.
- 사용자 리전에 쓰기 모드에서는 컴퓨팅 계층 요청 라우팅을 함께 사용하지 않는 한 Global Accelerator를 사용하지 않는 것이 가장 좋습니다.

## DynamoDB 글로벌 테이블을 사용한 리전 대피

리전 대피란 읽기 및 쓰기 활동을 해당 리전 외부로 마이그레이션하는 프로세스입니다. 대부분은 쓰기 활동이며 읽기 활동인 경우도 있습니다.

### 라이브 리전 대피

여러 이유로 라이브 리전 대피 결정을 내릴 수 있습니다. 대피는 follow-the-sun 한 리전에 쓰기 모드를 사용하는 경우처럼 일반적인 비즈니스 활동의 일부일 수 있습니다. DynamoDB 외부 소프트웨어 스택의 장애에 대응하여 현재 활성 리전을 변경하기로 하는 비즈니스 결정이나 리전 내에서 발생한 평소보다 긴 지연 시간 같은 일반적인 문제 때문에 대피가 이루어질 수도 있습니다.

임의의 리전에 쓰기 모드에서는 라이브 리전 대피가 간단합니다. 라우팅 시스템을 통해 트래픽을 대체 리전으로 라우팅하고, 대피된 리전에서 이미 이루어진 쓰기 작업을 평소처럼 복제할 수 있습니다.

한 리전에 쓰기 및 사용자 리전에 쓰기 모드에서는 새 활성 리전에서 쓰기를 시작하기 전에 활성 리전에 대한 모든 쓰기가 완전히 기록되고, 스트림이 처리되고, 전역으로 전파되었는지 확인해야 합니다. 이는 향후 쓰기가 최신 버전의 데이터를 기반으로 이루어지도록 하는 데 필요합니다.

리전 A는 활성이고 지역 B는 비활성이라고 가정해 보겠습니다(전체 테이블 또는 리전 A에 있는 항목의 경우). 대피를 수행하는 일반적인 메커니즘은 A에 대한 쓰기 작업을 일시 중지하고 이러한 작업이 B로 완전히 전파될 때까지 충분히 기다린 후 B를 활성으로 인식하도록 아키텍처 스택을 업데이트한 다음 B에 쓰기 작업을 재개하는 것입니다. 리전 A의 데이터가 리전 B에 완전히 복제되었음을 100% 확실하게 나타내는 지표는 없습니다. 리전 A가 정상인 경우 리전 A에 대한 쓰기 작업을 일시 중지하고 `ReplicationLatency` 지표의 최근 최대값의 10배를 기다리면 일반적으로 복제가 완료되었는지 확인하는 데 충분합니다. 리전 A가 비정상이고 다른 영역에서 지연 시간이 길어지면 대기 시간을 더 큰 배수로 설정할 수 있습니다.

## 오프라인 리전 대피

고려해야 할 특별한 경우가 있습니다. 리전 A가 예고 없이 완전히 오프라인 상태가 되면 어떻게 될까요? 그럴 가능성은 극히 낮지만 그래도 고려하는 것이 현명합니다. 이런 경우에는 아직 전파되지 않은 리전 A의 모든 쓰기 작업은 보관되었다가 리전 A가 다시 온라인 상태가 된 후에 전파됩니다. 쓰기 작업은 손실되지 않지만 전파는 무기한 지연됩니다.

이 경우 어떻게 진행할지는 애플리케이션이 결정합니다. 비즈니스 연속성을 위해서는 새 기본 리전 B에 쓰기 작업을 계속해야 할 수도 있습니다. 하지만 리전 A로부터 항목에 대한 쓰기 작업 전파가 보류 중인 동안 리전 B의 해당 항목이 업데이트를 수신하는 경우, 최종 쓰기 우선 모델에서는 전파가 억제됩니다. 리전 B에서의 모든 업데이트는 수신되는 쓰기 요청을 억제할 수 있습니다.

임의의 리전에 쓰기 모드에서는 리전 A의 항목이 결국 리전 B로 전파될 것으로 믿고 리전 A가 다시 온라인 상태가 될 때까지 항목이 누락될 가능성을 인식하면서 리전 B에서 읽기와 쓰기를 계속할 수 있습니다. 가능하면 최근 쓰기 트래픽을 재생(예: 업스트림 이벤트 소스 사용)하여 누락될 수 있는 쓰기 작업의 공백을 메우고, 최종 쓰기 우선 충돌 해결이 수신 쓰기 작업의 최종 전파를 억제하도록 하는 방법을 고려해야 합니다.

그 밖의 쓰기 모드에서는 살짝 최신에서 뒤떨어진 데이터로 작업을 계속할 수 있는 정도를 고려해야 합니다. `ReplicationLatency`로 추적되는 짧은 기간 동안의 일부 쓰기 작업은 리전 A가 다시 온라인 상태가 될 때까지 누락됩니다. 비즈니스를 계속 진행할 수 있을까요? 진행 가능한 사용 사례도 있겠지만 추가 완화 메커니즘 없이는 가능하지 않을 수도 있습니다.

예를 들어 리전 장애 이후에도 중단 없이 사용 가능한 크레딧 잔액을 유지해야 한다고 가정해 보겠습니다. 잔액을 리전 A에 있는 항목과 리전 B에 있는 항목으로 둘로 나누고 각 항목이 사용 가능한 잔액의 절반으로 시작하도록 할 수 있습니다. 이렇게 하면 사용자 리전에 쓰기 모드를 사용하는 것입니다. 각 리전에서 처리되는 트랜잭션 업데이트는 잔액의 로컬 사본에 기록됩니다. 리전 A가 완전히 오프라인 상태가 되더라도 리전 B에서 트랜잭션 처리를 계속 진행할 수 있으며, 쓰기 작업은 리전 B에 보관된 잔액 부분으로만 제한됩니다. 이렇게 잔액을 분할하면 잔액이 낮아지거나 크레딧을 재조정해야 할 때 복잡성이 발생하지만, 보류 중인 쓰기 작업에 불확실성이 있더라도 비즈니스를 안전하게 복구할 수 있는 한 가지 예가 됩니다.

또 다른 예로 웹 양식 데이터를 캡처하는 경우를 가정해 보겠습니다. OCC(낙관적 동시성 제어)를 사용하여 데이터 항목에 버전을 할당하고 최신 버전을 웹 양식에 숨겨진 필드로 포함할 수 있습니다. 제출할 때마다 데이터베이스에 있는 버전이 양식의 작성 기준 버전과 일치하는 경우에만 쓰기 작업이 성공합니다. 버전이 일치하지 않는 경우 데이터베이스에 있는 현재 버전을 기반으로 웹 양식을 새로 고치거나 신중하게 병합할 수 있고, 사용자는 다시 진행할 수 있습니다. OCC 모델은 일반적으로 다른 클라이언트가 데이터를 덮어쓰고 새 버전의 데이터를 생성하지 못하도록 보호하지만, 클라이언트가 이전 버전의 데이터를 발견할 수 있는 장애 조치 중에도 도움이 될 수 있습니다.

타임스탬프를 버전으로 사용하고 있다고 가정해 보겠습니다. 양식이 12:00에 리전 A를 기준으로 처음 작성되었지만 장애 조치 이후 리전 B에 쓰려고 시도하다가 데이터베이스에 있는 최신 버전이 11:59임을 알게 됐다고 가정해 보겠습니다. 이 시나리오에서 클라이언트는 12:00 버전이 리전 B로 전파될 때까지 기다린 다음 이 버전을 기반으로 쓰거나, 11:59를 기반으로 빌드하고 새 12:01 버전(쓰기 후에 리전 A가 복구된 후 수신 버전을 억제)을 생성할 수 있습니다.

마지막 예인 한 금융 서비스 회사는 DynamoDB 데이터베이스에 고객 계정 및 금융 거래에 대한 데이터를 보관합니다. 이 회사는 리전 A가 완전히 중단될 경우 고객 계정과 관련된 모든 쓰기 활동을 리전 B에서 완전히 사용할 수 있도록 하거나 리전 A가 다시 온라인 상태가 될 때까지 부분적으로 알려진 고객 계정을 격리하고자 했습니다. 이 회사는 모든 업무를 일시 중지하는 대신 트랜잭션이 전파되지 않은 것으로 판단되는 극히 일부의 계정만 업무를 일시 중지하기로 결정했습니다. 이를 위해 리전 C라고 부르는 세 번째 리전을 사용했습니다. 리전 A에서 쓰기 작업을 처리하기 전에 보류 중인 작업(예: 계정의 새 트랜잭션 수)을 간략하게 요약하여 리전 C에 배치했습니다. 이 요약만으로도 리전 B가 해당 뷰가 최신 상태인지 판단하기에 충분했습니다. 이 조치로 인해 리전 C에서의 쓰기 시점부터 리전 A가 쓰기 작업을 수락하고 지역 B가 쓰기 작업을 수신할 때까지 계정이 사실상 잠겼습니다. 리전 C에 있는 데이터는 장애 조치 프로세스의 일부인 경우를 제외하고는 사용되지 않았습니다. 장애 조치 후 리전 B는 리전 C와 데이터를 교차 검증하여 최신 상태가 아닌 계정이 있는지 확인할 수 있었습니다. 이 계정들은 리전 A 복구로 부분적 데이터가 리전 B로 전파될 때까지 격리됨으로 표시됩니다.

리전 C에 장애가 발생할 경우에는 새로운 리전 D를 스핀업하여 대신 사용할 수 있습니다. 데이터는 리전 C에 아주 잠깐 머물렀고, 몇 분 후에는 진행 중인 쓰기 작업에 대한 충분히 유용한 최신 기록이 리전 D에 있게 됩니다. 리전 B에 장애가 발생할 경우 리전 A는 리전 C와 협력하여 쓰기 요청을 계속 수락할 수 있었습니다. 이 회사는 지연 시간이 더 긴 쓰기(리전 C와 리전 A에 대한 쓰기)를 받아들일 용의가 있었고, 다행히도 계정 상태를 간략하게 요약할 수 있는 데이터 모델이 있었습니다.

## DynamoDB 글로벌 테이블의 처리량 용량 계획

한 리전에서 다른 리전으로 트래픽을 마이그레이션하려면 용량과 관련된 DynamoDB 테이블 설정을 신중하게 고려해야 합니다.

쓰기 용량 관리에 대한 몇 가지 고려 사항:

- 글로벌 테이블은 온디맨드 모드이거나 Auto Scaling이 활성화된 상태로 프로비저닝되어야 합니다.
- Auto Scaling으로 프로비저닝된 경우 쓰기 설정(최소, 최대, 목표 사용률)이 여러 리전에 복제됩니다. Auto Scaling 설정은 동기화되지만 실제로 프로비저닝되는 쓰기 용량은 리전 간에 독립적으로 변동될 수 있습니다.
- 프로비저닝된 쓰기 용량이 다르게 보일 수 있는 한 가지 이유는 TTL 기능 때문입니다. DynamoDB에서 TTL을 활성화할 때 값이 항목의 만료 시간을 나타내는 속성 이름을 초 단위로 Unix epoch 시간 형

식으로 지정할 수 있습니다. 이 시간이 지나면 DynamoDB가 쓰기 비용 없이 항목을 삭제할 수 있습니다. 글로벌 테이블을 사용하는 경우 한 리전에서 TTL을 구성하면 글로벌 테이블과 연결된 다른 리전에 설정이 자동으로 복제됩니다. TTL 규칙을 통해 항목을 삭제할 수 있는 경우 모든 리전에서 이 작업을 수행할 수 있습니다. 삭제 작업은 원본 테이블의 쓰기 단위를 소비하지 않고 수행되지만 복제본 테이블은 해당 삭제 작업의 복제된 쓰기를 받게 되며 복제된 쓰기 단위 비용이 발생합니다.

- Auto Scaling을 사용하는 경우 프로비저닝된 최대 쓰기 용량 설정이 모든 쓰기 작업과 모든 잠재적 TTL 삭제 작업을 처리할 수 있을 만큼 충분히 높아야 합니다. Auto Scaling은 쓰기 사용량에 따라 각 리전을 조정합니다. 온디맨드 테이블에는 프로비저닝된 최대 쓰기 용량 설정이 없지만 테이블 수준 최대 쓰기 처리량 제한이 온디맨드 테이블이 허용하는 최대 지속 쓰기 용량을 지정합니다. 기본 제한은 40,000이지만 조정할 수 있습니다. 온디맨드 테이블에 필요할 수 있는 모든 쓰기 작업(TTL 쓰기 작업 포함)을 처리할 수 있을 만큼 이 제한을 충분히 높게 설정하는 것이 좋습니다. 글로벌 테이블을 설정할 때는 모든 참여 리전에서 이 값이 동일해야 합니다.

읽기 용량 관리에 대한 몇 가지 고려 사항:

- 리전마다 읽기 패턴이 독립적일 수 있다고 가정하기 때문에 읽기 용량 관리 설정이 리전마다 다를 수 있습니다. 테이블에 처음 글로벌 복제본을 추가하면 소스 리전의 용량이 전파됩니다. 생성 후에 읽기 용량 설정을 조정할 수 있으며, 이 설정은 반대쪽으로 전송되지 않습니다.
- DynamoDB Auto Scaling을 사용할 때는 프로비저닝된 최대 읽기 용량 설정이 모든 리전에서 모든 읽기 작업을 처리할 수 있을 만큼 충분히 높아야 합니다. 표준 운영 중에는 읽기 용량이 여러 지역에 분산될 수 있지만 장애 조치 중에는 테이블이 증가된 읽기 워크로드에 맞게 자동으로 조정될 수 있어야 합니다. 온디맨드 테이블에는 프로비저닝된 최대 읽기 용량 설정이 없지만 테이블 수준 최대 읽기 처리량 제한이 온디맨드 테이블이 허용하는 최대 지속 읽기 용량을 지정합니다. 기본 제한은 40,000이지만 조정할 수 있습니다. 모든 읽기 작업이 이 단일 리전으로 라우팅되는 경우 테이블에 필요할 수 있는 모든 읽기 작업을 처리할 수 있을 만큼 이 제한을 충분히 높게 설정하는 것이 좋습니다.
- 한 리전의 테이블이 일반적으로 읽기 트래픽을 수신하지 못하지만 장애 조치 후 많은 양의 읽기 트래픽을 흡수해야 하는 경우, 테이블의 프로비저닝된 읽기 용량을 늘리고 테이블 업데이트가 완료될 때까지 기다린 다음 테이블을 다시 프로비저닝할 수 있습니다. 테이블을 프로비저닝된 모드로 두거나 온디맨드 모드로 전환할 수 있습니다. 이렇게 하면 테이블이 사전 워밍되어 더 높은 수준의 읽기 트래픽을 수용할 수 있습니다.

ARC에 있는 준비 상태 확인 기능은 Route 53을 사용하여 요청을 라우팅하는지 여부에 관계없이 DynamoDB 리전에 유사한 테이블 설정과 계정 할당량이 있는지 확인하는 데 유용할 수 있습니다. 이러한 준비 상태 확인은 계정 수준의 할당량을 조정하여 일치하도록 하는 데도 도움이 될 수 있습니다.

# DynamoDB 글로벌 테이블 준비 체크리스트 및 자주 묻는 질문

글로벌 테이블을 배포할 때 의사 결정 및 작업에 다음 체크리스트를 사용하세요.

- 글로벌 테이블에 참여해야 하는 리전과 리전 수를 결정합니다.
- 애플리케이션의 쓰기 모드를 결정합니다. 자세한 내용은 DynamoDB 글로벌 테이블을 사용한 쓰기 모드 섹션을 참조하세요.
- 쓰기 모드에 따라 DynamoDB 글로벌 테이블을 사용한 요청 라우팅 전략을 계획합니다.
- 쓰기 모드 및 라우팅 전략에 따라

  리전 대피란 읽기 및 쓰기 활동을 해당 리전 외부로 마이그레이션하는 프로세스입니다. 대부분은 쓰기 활동이며 읽기 활동인 경우도 있습니다.

  ## 라이브 리전 대피

  여러 이유로 라이브 리전 대피 결정을 내릴 수 있습니다. 대피는 follow-the-sun 한 리전에 쓰기 모드를 사용하는 경우처럼 일반적인 비즈니스 활동의 일부일 수 있습니다. DynamoDB 외부 소프트웨어 스택의 장애에 대응하여 현재 활성 리전을 변경하기로 하는 비즈니스 결정이나 리전 내에서 발생한 평소보다 긴 지연 시간 같은 일반적인 문제 때문에 대피가 이루어질 수도 있습니다.

  임의의 리전에 쓰기 모드에서는 라이브 리전 대피가 간단합니다. 라우팅 시스템을 통해 트래픽을 대체 리전으로 라우팅하고, 대피된 리전에서 이미 이루어진 쓰기 작업을 평소처럼 복제할 수 있습니다.

  한 리전에 쓰기 및 사용자 리전에 쓰기 모드에서는 새 활성 리전에서 쓰기를 시작하기 전에 활성 리전에 대한 모든 쓰기가 완전히 기록되고, 스트림이 처리되고, 전역으로 전파되었는지 확인해야 합니다. 이는 향후 쓰기가 최신 버전의 데이터를 기반으로 이루어지도록 하는 데 필요합니다.

  리전 A는 활성이고 지역 B는 비활성이라고 가정해 보겠습니다(전체 테이블 또는 리전 A에 있는 항목의 경우). 대피를 수행하는 일반적인 메커니즘은 A에 대한 쓰기 작업을 일시 중지하고 이러한 작업이 B로 완전히 전파될 때까지 충분히 기다린 후 B를 활성으로 인식하도록 아키텍처 스택을 업데이트한 다음 B에 쓰기 작업을 재개하는 것입니다. 리전 A의 데이터가 리전 B에 완전히 복제되었음을 100% 확실하게 나타내는 지표는 없습니다. 리전 A가 정상인 경우 리전 A에 대한 쓰기 작업을 일시 중지하고 `ReplicationLatency` 지표의 최근 최대값의 10배를 기다리면 일반적으로 복제가 완료되었는지 확인하는 데 충분합니다. 리전 A가 비정상이고 다른 영역에서 지연 시간이 길어지면 대기 시간을 더 큰 배수로 설정할 수 있습니다.

## 오프라인 리전 대피

고려해야 할 특별한 경우가 있습니다. 리전 A가 예고 없이 완전히 오프라인 상태가 되면 어떻게 될까요? 그럴 가능성은 극히 낮지만 그래도 고려하는 것이 현명합니다. 이런 경우에는 아직 전파되지 않은 리전 A의 모든 쓰기 작업은 보관되었다가 리전 A가 다시 온라인 상태가 된 후에 전파됩니다. 쓰기 작업은 손실되지 않지만 전파는 무기한 지연됩니다.

이 경우 어떻게 진행할지는 애플리케이션이 결정합니다. 비즈니스 연속성을 위해서는 새 기본 리전 B에 쓰기 작업을 계속해야 할 수도 있습니다. 하지만 리전 A로부터 항목에 대한 쓰기 작업 전파가 보류 중인 동안 리전 B의 해당 항목이 업데이트를 수신하는 경우, 최종 쓰기 우선 모델에서는 전파가 억제됩니다. 리전 B에서의 모든 업데이트는 수신되는 쓰기 요청을 억제할 수 있습니다.

임의의 리전에 쓰기 모드에서는 리전 A의 항목이 결국 리전 B로 전파될 것으로 믿고 리전 A가 다시 온라인 상태가 될 때까지 항목이 누락될 가능성을 인식하면서 리전 B에서 읽기와 쓰기를 계속할 수 있습니다. 가능하면 최근 쓰기 트래픽을 재생(예: 업스트림 이벤트 소스 사용)하여 누락될 수 있는 쓰기 작업의 공백을 메우고, 최종 쓰기 우선 충돌 해결이 수신 쓰기 작업의 최종 전파를 억제하도록 하는 방법을 고려해야 합니다.

그 밖의 쓰기 모드에서는 살짝 최신에서 뒤떨어진 데이터로 작업을 계속할 수 있는 정도를 고려해야 합니다. `ReplicationLatency`로 추적되는 짧은 기간 동안의 일부 쓰기 작업은 리전 A가 다시 온라인 상태가 될 때까지 누락됩니다. 비즈니스를 계속 진행할 수 있을까요? 진행 가능한 사용 사례도 있겠지만 추가 완화 메커니즘 없이는 가능하지 않을 수도 있습니다.

예를 들어 리전 장애 이후에도 중단 없이 사용 가능한 크레딧 잔액을 유지해야 한다고 가정해 보겠습니다. 잔액을 리전 A에 있는 항목과 리전 B에 있는 항목으로 둘로 나누고 각 항목이 사용 가능한 잔액의 절반으로 시작하도록 할 수 있습니다. 이렇게 하면 사용자 리전에 쓰기 모드를 사용하는 것입니다. 각 리전에서 처리되는 트랜잭션 업데이트는 잔액의 로컬 사본에 기록됩니다. 리전 A가 완전히 오프라인 상태가 되더라도 리전 B에서 트랜잭션 처리를 계속 진행할 수 있으며, 쓰기 작업은 리전 B에 보관된 잔액 부분으로만 제한됩니다. 이렇게 잔액을 분할하면 잔액이 낮아지거나 크레딧을 재조정해야 할 때 복잡성이 발생하지만, 보류 중인 쓰기 작업에 불확실성이 있더라도 비즈니스를 안전하게 복구할 수 있는 한 가지 예가 됩니다.

또 다른 예로 웹 양식 데이터를 캡처하는 경우를 가정해 보겠습니다. OCC(낙관적 동시성 제어)를 사용하여 데이터 항목에 버전을 할당하고 최신 버전을 웹 양식에 숨겨진 필드로 포함할 수 있습니다. 제출할 때마다 데이터베이스에 있는 버전이 양식의 작성 기준 버전과 일치하는 경우에만 쓰기 작업이 성공합니다. 버전이 일치하지 않는 경우 데이터베이스에 있는 현재 버전을 기반으로 웹 양식을 새로 고치거나 신중하게 병합할 수 있고, 사용자는 다시 진행할 수 있습니다. OCC 모델은 일

반적으로 다른 클라이언트가 데이터를 덮어쓰고 새 버전의 데이터를 생성하지 못하도록 보호하지만, 클라이언트가 이전 버전의 데이터를 발견할 수 있는 장애 조치 중에도 도움이 될 수 있습니다.

타임스탬프를 버전으로 사용하고 있다고 가정해 보겠습니다. 양식이 12:00에 리전 A를 기준으로 처음 작성되었지만 장애 조치 이후 리전 B에 쓰려고 시도하다가 데이터베이스에 있는 최신 버전이 11:59임을 알게 됐다고 가정해 보겠습니다. 이 시나리오에서 클라이언트는 12:00 버전이 리전 B로 전파될 때까지 기다린 다음 이 버전을 기반으로 쓰거나, 11:59를 기반으로 빌드하고 새 12:01 버전(쓰기 후에 리전 A가 복구된 후 수신 버전을 억제)을 생성할 수 있습니다.

마지막 예인 한 금융 서비스 회사는 DynamoDB 데이터베이스에 고객 계정 및 금융 거래에 대한 데이터를 보관합니다. 이 회사는 리전 A가 완전히 중단될 경우 고객 계정과 관련된 모든 쓰기 활동을 리전 B에서 완전히 사용할 수 있도록 하거나 리전 A가 다시 온라인 상태가 될 때까지 부분적으로 알려진 고객 계정을 격리하고자 했습니다. 이 회사는 모든 업무를 일시 중지하는 대신 트랜잭션이 전파되지 않은 것으로 판단되는 극히 일부의 계정만 업무를 일시 중지하기로 결정했습니다. 이를 위해 리전 C라고 부르는 세 번째 리전을 사용했습니다. 리전 A에서 쓰기 작업을 처리하기 전에 보류 중인 작업(예: 계정의 새 트랜잭션 수)을 간략하게 요약하여 리전 C에 배치했습니다. 이 요약만으로도 리전 B가 해당 뷰가 최신 상태인지 판단하기에 충분했습니다. 이 조치로 인해 리전 C에서의 쓰기 시점부터 리전 A가 쓰기 작업을 수락하고 지역 B가 쓰기 작업을 수신할 때까지 계정이 사실상 잠겼습니다. 리전 C에 있는 데이터는 장애 조치 프로세스의 일부인 경우를 제외하고는 사용되지 않았습니다. 장애 조치 후 리전 B는 리전 C와 데이터를 교차 검증하여 최신 상태가 아닌 계정이 있는지 확인할 수 있었습니다. 이 계정들은 리전 A 복구로 부분적 데이터가 리전 B로 전파될 때까지 격리됨으로 표시됩니다.

리전 C에 장애가 발생할 경우에는 새로운 리전 D를 스핀업하여 대신 사용할 수 있습니다. 데이터는 리전 C에 아주 잠깐 머물렀고, 몇 분 후에는 진행 중인 쓰기 작업에 대한 충분히 유용한 최신 기록이 리전 D에 있게 됩니다. 리전 B에 장애가 발생할 경우 리전 A는 리전 C와 협력하여 쓰기 요청을 계속 수락할 수 있었습니다. 이 회사는 지연 시간이 더 긴 쓰기(리전 C와 리전 A에 대한 쓰기)를 받아들일 용의가 있었고, 다행히도 계정 상태를 간략하게 요약할 수 있는 데이터 모델이 있었습니다.

대피 계획을 정의합니다.

- 각 리전의 상태, 지연 시간, 오류에 대한 지표를 캡처합니다. DynamoDB 지표 목록은 AWS 블로그 게시물 Monitoring Amazon DynamoDB for Operational Awareness에 있는 관찰해야 할 지표 목록을 참조하세요. 또한 합성 카나리아(장애를 감지하도록 설계된 인위적 요청으로, 탄광에 있는 카나리아에서 유래)를 사용하고 고객 트래픽을 실시간으로 관찰해야 합니다. 모든 문제가 DynamoDB 지표에 나타나는 것은 아닙니다.
- `ReplicationLatency`의 지속적 증가에 대한 경보를 설정하세요. 증가는 글로벌 테이블의 쓰기 설정이 리전마다 다른 잘못된 구성을 나타낼 수 있습니다. 이는 복제된 요청 실패와 지연 시간 증가로

이어질 수 있습니다. 리전 중단이 있음을 나타낼 수도 있습니다. 좋은 예는 최근 평균이 180,000밀리초를 초과할 경우 알림을 생성하는 것입니다. `ReplicationLatency`가 0으로 떨어지는 것을 관찰할 수도 있습니다. 이는 복제가 중단되었음을 나타냅니다.

- 각 글로벌 테이블에 충분한 최대 읽기 및 쓰기 설정을 할당합니다.
- 리전 대피 사유를 미리 식별합니다. 결정에 사람의 판단이 수반되는 경우 모든 고려 사항을 문서화합니다. 이 작업은 압박을 받지 않는 상태에서 사전에 신중하게 수행해야 합니다.
- 리전 대피 시 취해야 하는 모든 조치를 위한 런북을 유지 관리합니다. 일반적으로 글로벌 테이블에 필요한 작업은 거의 없지만 나머지 스택을 이동하는 작업은 복잡할 수 있습니다.

> **Note**
>
> 리전 장애 중에는 일부 컨트롤 플레인 작업이 저하될 수 있으므로 데이터 플레인 작업에만 의존하고 컨트롤 플레인 작업에는 의존하지 않는 것이 모범 사례입니다.

자세한 내용은 AWS 블로그 게시물 Build resilient applications with Amazon DynamoDB global tables: Part 4를 참조하세요.

- 리전 대피를 포함하여 런북의 모든 측면을 정기적으로 테스트합니다. 테스트되지 않은 런북은 신뢰할 수 없는 런북입니다.
- Resilience Hub를 사용하여 전체 애플리케이션(글로벌 테이블 포함)의 복원력을 평가하는 것을 고려해 보세요. Resilience Hub의 대시보드를 통해 전체 애플리케이션 포트폴리오 복원력 상태를 종합적으로 파악할 수 있습니다.
- ARC 준비 상태 확인을 사용하여 애플리케이션의 현재 구성을 평가하고 모범 사례에서 벗어난 부분을 추적하는 것을 고려해 보세요.
- Route 53 또는 Global Accelerator와 함께 사용할 상태 확인을 작성할 때는 DynamoDB 엔드포인트가 작동 중이라는 핑만으로는 충분하지 않습니다. IAM 구성 오류, 코드 배포 문제, DynamoDB 외부 스택 장애, 평균보다 높은 읽기 또는 쓰기 지연 시간 등 다양한 장애 모드는 포함되지 않기 때문입니다. 전체 데이터베이스 흐름을 실행하는 일련의 호출을 수행하는 것이 가장 좋습니다.

## 글로벌 테이블 배포에 대한 자주 묻는 질문(FAQ)

DynamoDB 글로벌 테이블의 전반적 사용에 유용한 원칙에는 어떤 것이 있나요?

DynamoDB 글로벌 테이블에는 제어 노브가 거의 없지만 여전히 여러 가지를 고려해야 합니다. 쓰기 모드, 라우팅 모델, 대피 프로세스를 결정해야 합니다. 글로벌 상태 유지를 위해서는 모든 리전에서 애플리케이션을 계측하고 경로를 조정하거나 대피를 수행할 준비가 되어 있어야 합니다. 그 보상으로 읽기

및 쓰기 지연 시간이 짧고 99.999% SLA(서비스 수준 계약)를 갖춘 전 세계에 분산된 데이터 세트를 보유할 수 있습니다.

글로벌 테이블의 요금은 어떻게 되나요?

기존 DynamoDB 테이블에 대한 쓰기 요금은 쓰기 용량 단위(WCU, 프로비저닝된 테이블의 경우) 또는 쓰기 요청 단위(WRU, 온디맨드 테이블의 경우)로 책정됩니다. 5KB 항목을 쓰면 5단위의 요금이 부과됩니다. 글로벌 테이블에 대한 쓰기 요금은 복제된 쓰기 용량 단위(rWCU, 프로비저닝된 테이블의 경우) 또는 복제된 쓰기 요청 단위(rWRU, 온디맨드 테이블의 경우)로 책정됩니다.

rWCU와 rWRU에는 복제를 관리하는 데 필요한 스트리밍 인프라 비용이 포함됩니다.

복제된 쓰기 단위 요금은 항목을 직접 쓰거나 복제하여 쓰는 모든 리전에서 발생합니다.

글로벌 보조 인덱스(GSI)에 쓰기는 로컬 쓰기로 간주되며, 일반 쓰기 단위를 사용합니다.

현재 rWCU에는 예약 용량을 사용할 수 없습니다. GSI가 쓰기 단위를 소비하는 테이블의 경우 예약 용량을 구매하는 것이 여전히 유용할 수 있습니다.

글로벌 테이블에 새 리전 추가 시 초기 부트스트랩에는 복원된 데이터 GB당 복원 요금에 리전 간 데이터 전송 요금이 추가로 부과됩니다.

글로벌 테이블은 어떤 리전을 지원하나요?

글로벌 테이블 버전 2019.11.21(현재)은 모든 리전에서 사용할 수 있습니다.

글로벌 테이블에서는 GSI를 어떻게 처리하나요?

글로벌 테이블 버전 2019.11.21(현재)에서는 한 리전에서 생성한 GSI가 다른 참여 리전에도 자동으로 생성되고 자동으로 백필됩니다.

글로벌 테이블의 복제를 중지하려면 어떻게 해야 하나요?

다른 테이블을 삭제하는 것과 같은 방식으로 복제본 테이블을 삭제할 수 있습니다. 글로벌 테이블을 삭제하면 해당 리전으로의 복제가 중지되고 해당 리전에 보관된 테이블 사본이 삭제됩니다. 하지만 테이블의 사본을 독립적 엔터티로 유지하는 동안에는 복제를 중지할 수 없으며 복제를 일시 중지할 수도 없습니다.

DynamoDB Streams는 글로벌 테이블과 어떻게 상호 작용하나요?

각 글로벌 테이블은 시작 위치에 관계없이 모든 쓰기를 기반으로 독립적인 스트림을 생성합니다. 이 DynamoDB 스트림을 한 리전 또는 모든 리전에서 독립적으로 사용하도록 선택할 수 있습니다. 로컬 쓰

기 작업은 처리하되 복제된 쓰기 작업은 처리하지 않으려는 경우 쓰기 리전을 식별하는 고유한 리전 속성을 각 항목에 추가할 수 있습니다. 그런 다음 Lambda 이벤트 필터를 사용하여 로컬 리전에서의 쓰기 작업만을 위한 Lambda 함수를 호출할 수 있습니다. 이렇게 하면 삽입 및 업데이트 작업에 도움이 되지만 안타깝게도 삭제 작업에는 도움이 되지 않습니다.

글로벌 테이블은 트랜잭션을 어떻게 처리하나요?

트랜잭션 작업은 쓰기 작업이 원래 이루어진 리전에서만 ACID(원자성, 일관성, 격리 및 내구성) 보장을 제공합니다. 전역 테이블에서는 트랜잭션이 리전 간에 지원되지 않습니다. 예를 들어 미국 동부(오하이오) 및 미국 서부(오레곤) 리전에 복제본이 있는 글로벌 테이블이 있고 미국 동부(오하이오)에서 `TransactWriteItems` 작업을 수행할 경우, 변경 내용이 복제될 때 미국 서부(오레곤)에서 부분적으로 완료된 트랜잭션을 관찰할 수 있습니다. 변경 사항은 소스 리전에서 커밋된 이후에만 다른 리전에 복제됩니다.

글로벌 테이블은 DynamoDB Accelerator 캐시(DAX)와 어떻게 상호 작용하나요?

글로벌 테이블은 DynamoDB를 직접 업데이트하여 DAX를 우회하므로 DAX는 오래된 데이터를 보유하고 있다는 사실을 인식하지 못합니다. DAX 캐시는 캐시의 TTL이 만료되는 경우에만 새로 고쳐집니다.

테이블의 태그가 전파되나요?

아니요, 태그는 자동으로 전파되지 않습니다.

테이블을 모든 리전에 백업해야 하나요 아니면 한 리전에만 백업해야 하나요?

대답은 백업의 목적에 따라 달라집니다. 데이터 내구성을 보장하려는 경우 DynamoDB는 이미 그러한 보호 장치를 제공합니다. 이 서비스는 내구성을 보장합니다. 기록 레코드의 스냅샷을 보관하려는 경우(예: 규정 요구 사항 충족) 한 리전에 백업하는 것으로 충분합니다. AWS Backup를 사용하여 백업을 추가 리전에 복사할 수 있습니다. 실수로 삭제되거나 수정된 데이터를 복구하려면 한 리전에서 DynamoDB 시점 복구(PITR)를 사용하세요.

AWS CloudFormation을 사용하여 글로벌 테이블을 배포하려면 어떻게 해야 하나요?

CloudFormation은 DynamoDB 테이블과 글로벌 테이블을 두 개의 개별 리소스인 `AWS::DynamoDB::Table`과 `AWS::DynamoDB::GlobalTable`로 나타냅니다. 한 가지 방법은 `GlobalTable` 구문을 사용하여 잠재적으로 글로벌일 수 있는 모든 테이블을 생성하는 것입니다. 그러면 처음에는 이를 독립 실행형 테이블로 유지하고 필요한 경우 나중에 리전에 추가할 수 있습니다.

CloudFormation에서 각 글로벌 테이블은 복제본 수에 관계없이 단일 리전에서 단일 스택에 의해 제어됩니다. 템플릿을 배포하면 CloudFormation은 단일 스택 작업의 일부로 모든 복제본을 생성하고 업

데이트합니다. 동일한 AWS::DynamoDB::GlobalTable 리소스를 여러 리전에 배포하면 안 됩니다. 이런 배포는 오류를 유발하며 지원되지 않습니다. 여러 리전에 애플리케이션 템플릿을 배포하는 경우 조건을 사용하여 단일 리전에서 AWS::DynamoDB::GlobalTable 리소스를 생성할 수 있습니다. 또는 AWS::DynamoDB::GlobalTable 리소스를 애플리케이션 스택과 별도의 스택에 정의하고 단일 리전에 배포되도록 선택할 수 있습니다.

일반 테이블이 있고 이 테이블을 CloudFormation이 계속 관리하는 글로벌 테이블로 변환하려는 경우, 삭제 정책을 유지로 설정하고, 스택에서 테이블을 제거하고, 콘솔에서 테이블을 글로벌 테이블로 변환한 다음 글로벌 테이블을 스택에 새 리소스로 가져옵니다.

교차 계정 복제는 현재 지원되지 않습니다.

# DynamoDB의 컨트롤 플레인 관리 모범 사례

> **Note**
>
> DynamoDB에는 재시도 옵션과 함께 초당 2,500개 요청의 컨트롤 플레인  제한이 도입될 예정입니다. 자세한 내용은 아래를 참조하세요.

DynamoDB 컨트롤 플레인 작업을 통해 DynamoDB 테이블은 물론 인덱스와 같은 테이블에 종속된 객체를 관리할 수 있습니다. 이러한 작업에 대한 자세한 내용은 컨트롤 플레인 섹션을 참조하세요.

경우에 따라 조치를 취하고 컨트롤 플레인 호출에서 반환된 데이터를 비즈니스 로직의 일부로 사용해야 할 수 있습니다. 예를 들어, DescribeTable에서 반환되는 ProvisionedThroughput의 값을 알아야 할 수도 있습니다. 이러한 상황에서는 다음과 같은 모범 사례를 따르세요.

- DynamoDB 컨트롤 플레인을 과도하게 쿼리하지 않습니다.
- 동일한 코드 내에서 컨트롤 플레인 호출과 데이터 영역 호출을 혼용하지 않습니다.
- 컨트롤 플레인 요청에 대한 제한을 처리하고 백오프로 재시도합니다.
- 단일 클라이언트에서 특정 리소스를 호출하고 변경 사항을 추적합니다.
- 동일한 테이블의 데이터를 짧은 간격으로 여러 번 검색하는 대신 데이터를 캐싱하여 처리합니다.

# DynamoDB에서 대량 데이터 작업을 사용하는 모범 사례

DynamoDB는 최대 25개의 PutItem 및 DeleteItem 요청을 함께 수행할 수 있는 BatchWriteItem과 같은 배치 작업을 지원합니다. 그러나 BatchWriteItem은 UpdateItem 작

업을 지원하지 않습니다. 대량 업데이트의 경우 업데이트 요구 사항과 특성 면에서 차이가 있습니다. `TransactWriteItems`를 비롯한 다른 DynamoDB API를 사용하여 배치 크기를 최대 100까지 늘릴 수 있습니다. 더 많은 항목이 관련된 경우 AWS Glue, Amazon EMR, AWS Step Functions와 같은 서비스를 사용하거나 DynamoDB-shell과 같은 사용자 지정 스크립트 및 도구를 사용하여 대량 업데이트를 수행할 수 있습니다.

주제


- 조건부 배치 업데이트
- 효율적인 대량 작업


## 조건부 배치 업데이트

DynamoDB는 최대 25개의 `PutItem` 및 `DeleteItem` 요청을 단일 배치로 수행할 수 있는 `BatchWriteItem`과 같은 배치 작업을 지원합니다. 그러나 `BatchWriteItem`은 `UpdateItem` 작업을 지원하지 않으며 조건식을 지원하지 않습니다. 해결 방법으로 `TransactWriteItems`를 비롯한 다른 DynamoDB API를 사용하여 배치 크기를 최대 100까지 늘릴 수 있습니다.

더 많은 항목이 관련되고 주요 데이터 청크를 변경해야 하는 경우 AWS Glue, Amazon EMR, AWS Step Functions와 같은 서비스를 사용하거나 DynamoDB-shell과 같은 사용자 지정 스크립트 및 도구를 사용하여 효율적인 대량 업데이트를 수행할 수 있습니다.

이 패턴을 사용해야 하는 경우

- DynamoDB-shell은 프로덕션 사용 사례에서 지원되지 않습니다.
- `TransactWriteItems` - 조건 유무에 관계없이 최대 100개의 개별 업데이트로, 전부 아니면 전무 방식의 ACID 번들로 실행됩니다. 애플리케이션에 멱등성이 필요한 경우 `TransactWriteItems` 직접 호출을 `ClientRequestToken`과 함께 제공하여 동일한 여러 호출이 하나의 호출과 같은 효과를 갖도록 할 수 있습니다. 이렇게 하면 동일한 트랜잭션을 여러 번 실행하지 않고 잘못된 데이터 상태로 끝나지 않습니다.

  단점 - 추가 처리량이 사용됩니다. 표준 1KB 쓰기당 1WGU 대신 1KB 쓰기당 2WCU.
- PartiQL `BatchExecuteStatement` - 조건 유무에 관계없이 최대 25개의 업데이트입니다. `BatchExecuteStatement`는 항상 전체 요청에 대한 성공 응답을 반환하고 순서를 유지하는 개별 작업 응답 목록도 반환합니다.

  단점 - 더 큰 배치의 경우 요청을 25개 배치로 배포하려면 추가 클라이언트측 로직이 필요합니다. 재시도 전략을 결정하려면 개별 오류 응답을 고려해야 합니다.

## 코드 예제

이러한 코드 예제에서는 AWS SDK for Python인 boto3 라이브러리를 사용합니다. 이러한 예제에서는 boto3를 설치하고 적절한 AWS 자격 증명으로 구성했다고 가정합니다.

유럽 도시 곳곳에 여러 창고를 둔 전자 제품 공급업체의 재고 데이터베이스를 생각해 보겠습니다. 여름이 끝났기 때문에 공급업체는 탁상용 선풍기를 정리하여 다른 재고를 위한 공간을 확보하려고 합니다. 이 공급업체는 이탈리아의 창고에서 공급되는 모든 탁상용 선풍기에 대해 가격 할인을 제공하고자 하지만 탁상용 선풍기 예비 재고가 20개 있는 경우에 한해 이렇게 하려고 합니다. inventory라는 이름의 DynamoDB 테이블에 각 제품의 고유 식별자인 파티션 키 sku와 웨어하우스의 식별자인 정렬 키 warehouse의 키 스키마가 있습니다.

다음 Python 코드는 `BatchExecuteStatement` API 직접 호출을 사용하여 이 조건부 배치 업데이트를 수행하는 방법을 보여줍니다.

```
import boto3

client=boto3.client("dynamodb")

before_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("Before update: ", before_image['Items'])

response=client.batch_execute_statement(
        Statements=[
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITTUR1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM2'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM5'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
```

```
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN2'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN3'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
        ],
        ReturnConsumedCapacity='TOTAL'
    )

after_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("After update: ", after_image['Items'])
```

실행 시 샘플 데이터에 대해 아래와 같은 출력이 생성됩니다.

```
Before update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price':
 {'N': '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S':
 'WITROM2'}, 'price': {'N': '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '28'}, 'warehouse': {'S': 'WITROM5'}, 'price': {'N': '38'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '40'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price':
 {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S':
 'WITVEN2'}, 'price': {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'},
 'warehouse': {'S': 'WITVEN3'}, 'price': {'N': '35'}, 'sku': {'S': 'F123'}}]
After update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price': {'N':
 '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S': 'WITROM2'},
 'price': {'N': '35'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '28'}, 'warehouse':
 {'S': 'WITROM5'}, 'price': {'N': '33'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '35'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price': {'N': '38'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S': 'WITVEN2'}, 'price':
 {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'}, 'warehouse': {'S':
 'WITVEN3'}, 'price': {'N': '30'}, 'sku': {'S': 'F123'}}]
```

이는 내부 시스템에 대한 제한된 작업이므로 멱등성 요구 사항은 고려되지 않았습니다. 가격이 35보다 크고 40보다 작은 경우에만 가격 업데이트가 진행되어야 하는 것과 같은 추가 가드레일을 배치하여 업데이트를 더 견고하게 만들 수 있습니다.

또는 더 엄격한 멱등성 및 ACID 요구 사항이 있는 경우 TransactWriteItems를 사용하여 동일한 배치 업데이트 작업을 수행할 수 있습니다. 그러나 트랜잭션 번들의 모든 작업이 진행되거나 전체 번들이 실패한다는 점에 유의해야 합니다.

이탈리아에 폭염이 발생하여 탁상용 선풍기 수요가 급격히 증가했다고 해보겠습니다. 공급업체는 이탈리아의 모든 창고에서 출고되는 탁상용 선풍기의 가격을 20유로 인상하기를 원하지만 규제 기관은 전체 재고에서 현재 가격이 70유로 미만인 경우에 한해 이러한 가격 인상을 허용하고 있습니다. 각 창고에서 가격이 70유로 미만인 경우에만 재고 전체에 걸쳐 한 번만 가격을 업데이트해야 합니다.

다음 Python 코드는 TransactWriteItems API 직접 호출을 사용하여 이 배치 업데이트를 수행하는 방법을 보여줍니다.

```
import boto3

client=boto3.client("dynamodb")

before_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("Before update: ", before_image['Items'])

response=client.transact_write_items(
        ClientRequestToken='UUIDAWS124',
        TransactItems=[
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITTUR1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM2'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM5'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
```

```
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN2'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN3'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
        ],
        ReturnConsumedCapacity='TOTAL'
    )

after_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("After update: ", after_image['Items'])
```

실행 시 샘플 데이터에 대해 아래와 같은 출력이 생성됩니다.

```
Before update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price':
 {'N': '60'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S':
 'WITROM2'}, 'price': {'N': '55'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '28'}, 'warehouse': {'S': 'WITROM5'}, 'price': {'N': '53'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '55'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price':
 {'N': '58'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S':
 'WITVEN2'}, 'price': {'N': '58'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'},
 'warehouse': {'S': 'WITVEN3'}, 'price': {'N': '50'}, 'sku': {'S': 'F123'}}]
After update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price': {'N':
 '80'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S': 'WITROM2'},
 'price': {'N': '75'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '28'}, 'warehouse':
 {'S': 'WITROM5'}, 'price': {'N': '73'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '75'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price': {'N': '78'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S': 'WITVEN2'}, 'price':
 {'N': '78'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'}, 'warehouse': {'S':
 'WITVEN3'}, 'price': {'N': '70'}, 'sku': {'S': 'F123'}}]
```

DynamoDB에서 배치 업데이트를 수행하는 방법에는 여러 가지가 있습니다. 적절한 접근 방식은 ACID 및/또는 멱등성 요구 사항, 업데이트할 항목 수, API에 대한 친숙도와 같은 요인에 따라 달라집니다.

## 효율적인 대량 작업

이 패턴을 사용해야 하는 경우

이러한 패턴은 DynamoDB 항목에 대한 대량 업데이트를 효율적으로 수행하는 데 유용합니다.

- DynamoDB-shell은 프로덕션 사용 사례에서 지원되지 않습니다.
- `TransactWriteItems` - 조건 유무에 관계없이 최대 100개의 개별 업데이트로, 전부 아니면 전무 방식의 ACID 번들로 실행됩니다.

  단점 - 1KB 쓰기당 2WCU로, 추가 처리량이 사용됩니다.
- PartiQL `BatchExecuteStatement` - 조건 유무에 관계없이 최대 25개의 업데이트입니다.

  단점 - 요청을 25개 배치로 배포하려면 추가 로직이 필요합니다.
- AWS Step Functions - AWS Lambda에 익숙한 개발자를 위한, 속도가 제한된 대량 작업입니다.

  단점 - 실행 시간이 속도 제한에 반비례합니다. 최대 Lambda 함수 제한 시간으로 제한됩니다. 이 기능의 경우 읽기와 쓰기 사이에 발생하는 데이터 변경 사항을 덮어쓸 수 있습니다. 자세한 내용은 Amazon EMR을 사용하여 Amazon DynamoDB Time to Live 속성 채우기: 2부를 참조하세요.
- AWS Glue 및 Amazon EMR - 관리형 병렬 처리를 사용하는, 속도가 제한된 대량 작업입니다. 시간에 민감하지 않은 애플리케이션 또는 업데이트의 경우 이러한 옵션을 백그라운드에서 실행하여 소량의 처리량만 사용할 수 있습니다. 두 서비스 모두 emr-dynamodb-connector를 사용하여 DynamoDB 작업을 수행합니다. 이러한 서비스는 속도 제한 옵션을 사용하여 업데이트된 항목의 대량 읽기를 수행한 후 대량 쓰기를 수행합니다.

  단점 - 실행 시간이 속도 제한에 반비례합니다. 이 기능의 경우 읽기와 쓰기 사이에 발생하는 데이터 변경 사항을 덮어쓸 수 있습니다. 글로벌 보조 인덱스(GSI)에서는 읽을 수 없습니다. Amazon EMR을 사용하여 Amazon DynamoDB Time to Live 속성 채우기: 2부를 참조하세요.
- DynamoDB Shell - SQL과 유사한 쿼리를 사용하는, 속도가 제한된 대량 작업입니다. 효율성을 높이기 위해 GSI에서 읽을 수 있습니다.

  단점 - 실행 시간이 속도 제한에 반비례합니다. DynamoDB Shell의 속도가 제한된 대량 작업을 참조하세요.

### 패턴 사용

대량 업데이트는 특히 온디맨드 처리량 모드를 사용하는 경우 비용에 상당한 영향을 미칠 수 있습니다. 프로비저닝된 처리량 모드를 사용하는 경우 속도와 비용 간에 절충이 있습니다. 속도 제한 파라미터를 매우 엄격하게 설정하면 처리 시간이 매우 길어질 수 있습니다. 평균 항목 크기와 속도 제한을 사용하여 업데이트 속도를 대략적으로 결정할 수 있습니다.

또는 업데이트 프로세스의 예상 기간과 평균 항목 크기를 기반으로 프로세스에 필요한 처리량을 결정할 수 있습니다. 각 패턴에 대해 공유된 참조 블로그에서 패턴 사용 전략, 구현 및 제한 사항과 관련한 세부 정보를 제공합니다. 자세한 내용은 Amazon DynamoDB를 사용한 비용 효율적인 대량 처리를 참조하세요.

라이브 DynamoDB 테이블에 대해 대량 업데이트를 수행하는 방법에는 여러 가지가 있습니다. 적절한 접근 방식은 ACID 및/또는 멱등성 요구 사항, 업데이트할 항목 수, API에 대한 친숙도와 같은 요인에 따라 달라집니다. 비용과 시간 간의 균형을 고려하는 것이 중요하며, 위에서 설명한 대부분의 접근 방식은 대량 업데이트 작업에 사용되는 처리량에 대한 속도 제한 옵션을 제공합니다.

## DynamoDB에서 버전 관리를 구현하는 모범 사례

DynamoDB와 같은 분산 시스템에서 낙관적 잠금을 사용하여 항목 버전을 관리하면 업데이트 충돌을 방지할 수 있습니다. 항목 버전을 추적하고 조건부 쓰기를 사용함으로써 애플리케이션에서 동시 수정을 관리하여 동시성이 높은 환경의 데이터 무결성을 보장할 수 있습니다.

낙관적 잠금은 데이터 수정 사항이 충돌 없이 올바르게 적용되도록 사용하는 전략입니다. 비관적 잠금에서와 같이 데이터를 읽을 때 데이터를 잠그는 대신 낙관적 잠금은 데이터를 다시 쓰기 전에 데이터가 변경되었는지 확인합니다. DynamoDB에서는 매번 업데이트 시 각 항목에 포함된 식별자가 증가하는 일종의 버전 관리를 통해 이것이 이루어집니다. 항목을 업데이트할 때 해당 식별자가 애플리케이션에서 예상되는 식별자와 일치하는 경우에만 작업이 성공합니다.

### 이 패턴을 사용해야 하는 경우

이 패턴은 다음과 같은 시나리오에서 유용합니다.

- 여러 사용자 또는 프로세스가 동일한 항목을 동시에 업데이트하려고 시도할 수 있습니다.
- 데이터 무결성과 일관성을 보장하는 것이 매우 중요합니다.
- 분산 잠금 관리의 오버헤드와 복잡성을 피해야 합니다.

그러한 예는 다음과 같습니다.

- 재고 수준이 자주 업데이트되는 전자 상거래 애플리케이션.
- 여러 사용자가 동일한 데이터를 편집하는 공동 작업 플랫폼.
- 거래 기록이 일관성을 유지해야 하는 금융 시스템.

### 단점

낙관적 잠금 및 조건부 확인은 강력한 데이터 무결성을 제공하지만 다음과 같은 단점이 있습니다.

동시성 충돌

동시성이 높은 환경에서는 충돌 가능성이 증가하여 재시도 횟수와 쓰기 비용이 증가할 수 있습니다.

구현 복잡성

항목에 버전 관리를 추가하고 조건부 확인을 처리하면 애플리케이션 로직이 더 복잡해질 수 있습니다.

추가 스토리지 오버헤드

각 항목의 버전 번호를 저장하면 스토리지 요구 사항이 약간 증가합니다.

## 패턴 설계

이 패턴을 구현하려면 DynamoDB 스키마에 각 항목에 대한 버전 속성이 포함되어야 합니다. 다음은 간단한 스키마 설계입니다.

- 파티션 키 - 각 항목의 고유 식별자(예: `ItemId`)입니다.
- 속성:
  - `ItemId` – 항목의 고유 식별자입니다.
  - `Version` - 항목의 버전 번호를 나타내는 정수입니다.
  - `QuantityLeft` - 항목의 남은 재고입니다.

항목이 처음 생성되면 `Version` 속성이 1로 설정됩니다. 매번 업데이트 시 버전 번호가 1씩 증가합니다.

VersionControl

| Primary key<br>Partition key: ItemID | Attributes | |
|---|---|---|
| Bananas | Version | QuantityLeft |
| | 1 | 10 |
| Apples | Version | QuantityLeft |
| | 1 | 5 |
| Oranges | Version | QuantityLeft |
| | 1 | 7 |

## 패턴 사용

이 패턴을 구현하려면 애플리케이션 흐름에서 다음 단계를 따르세요.

1. 항목의 현재 버전을 읽습니다.

   DynamoDB에서 현재 항목을 검색하고 해당 버전 번호를 읽습니다.

```
def get_document(item_id):
    response = table.get_item(Key={'ItemID': item_id})
    return response['Item']

document = get_document('Bananas')
current_version = document['Version']
```

2. 애플리케이션 로직에서 버전 번호를 증가시킵니다. 이는 업데이트의 예상 버전입니다.

```
new_version = current_version + 1
```

3. 버전 번호가 일치하는지 확인하기 위해 조건식으로 항목을 업데이트하려고 시도합니다.

```
def update_document(item_id, qty_bought, current_version):
    try:
        response = table.update_item(
            Key={'ItemID': item_id},
            UpdateExpression="set #qty = :qty, Version = :v",
            ConditionExpression="Version = :expected_v",
            ExpressionAttributeNames={
                '#qty': 'QuantityLeft'
```

```
            },
            ExpressionAttributeValues={
                ':qty': qty_bought,
                ':v': current_version + 1,
                ':expected_v': current_version
            },
            ReturnValues="UPDATED_NEW"
        )
        return response
    except ClientError as e:
        if e.response['Error']['Code'] == 'ConditionalCheckFailedException':
            print("Update failed due to version conflict.")
        else:
            print("Unexpected error: %s" % e)
        return None

update_document('Bananas', 2, new_version)
```

업데이트에 성공하면 항목의 QuantityLeft가 2만큼 줄어듭니다.

VersionControl

| Primary key<br>Partition key: ItemID | Attributes | |
|---|---|---|
| Bananas | Version | QuantityLeft |
| | 2 | 8 |
| Apples | Version | QuantityLeft |
| | 1 | 5 |
| Oranges | Version | QuantityLeft |
| | 1 | 7 |

4. 충돌이 발생하는 경우 충돌을 처리합니다.

   충돌이 발생하는 경우(예: 마지막으로 읽은 이후 다른 프로세스에서 항목을 업데이트함) 작업을 재시도하거나 사용자에게 알리는 등 충돌을 적절하게 처리합니다.

   이렇게 하려면 각 재시도 시 항목을 추가로 읽어야 하므로 요청 루프가 완전히 실패하기 전에 허용되는 총 재시도 횟수를 제한합니다.

```
def update_document_with_retry(item_id, new_data, retries=3):
    for attempt in range(retries):
        document = get_document(item_id)
```

```
        current_version = document['Version']

        result = update_document(item_id, qty_bought, current_version)

        if result is not None:
            print("Update succeeded.")
            return result
        else:
            print(f"Retrying update... ({attempt + 1}/{retries})")

    print("Update failed after maximum retries.")
    return None

update_document_with_retry('Bananas', 2)
```

DynamoDB에서 낙관적 잠금 및 조건부 확인과 함께 항목 버전 관리를 구현하는 것은 분산 애플리케이션에서 데이터 무결성을 보장하는 강력한 패턴입니다. 약간의 복잡성과 잠재적 성능 저하가 발생하지만 강력한 동시성 제어가 필요한 시나리오에서 이는 매우 중요합니다. 스키마를 신중하게 설계하고 애플리케이션 로직에 필요한 확인을 구현하면 동시 업데이트를 효과적으로 관리하고 데이터 일관성을 유지할 수 있습니다.

DynamoDB 데이터의 버전 관리를 구현하는 방법에 대한 추가 지침 및 전략은 AWS 데이터베이스 블로그에서 확인할 수 있습니다.

# DynamoDB의 AWS 결제 및 사용량 보고서 이해에 관한 모범 사례

이 문서에서는 DynamoDB 관련 요금의 `UsageType` 결제 코드를 설명합니다.

AWS는 사용된 서비스에 대한 데이터가 포함된 비용 및 사용량 보고서(CUR)를 제공합니다. AWS Cost and Usage Report를 사용하여 결제 보고서를 Amazon S3에 CSV 형식으로 게시할 수 있습니다. CUR을 설정할 때 기간을 시간, 일 또는 월별로 구분할 수 있으며 리소스 ID별로 사용량을 구분할지를 선택할 수 있습니다. CUR 생성에 대한 자세한 내용은 Creating Cost and Usage Reports를 참조하세요.

CSV 내보내기에서 각 라인에 나열된 관련 속성을 찾을 수 있습니다. 포함될 수 있는 속성의 예는 다음과 같습니다.

- lineitem/UsageStartDate: 이 행 항목의 시작 날짜와 시간(UTC)입니다. 포괄적입니다.
- lineitem/UsageEndDate: 해당 행 항목의 종료 날짜와 시간(UTC)입니다. 배타적입니다.
- lineitem/ProductCode: DynamoDB의 경우 'AmazonDynamoDB'입니다.

- lineitem/UsageType: 이 문서에 열거된 사용량 유형에 대한 특정 설명 코드입니다.
- lineitem/Operation: 요금이 발생한 작업 이름과 같이 요금에 대한 컨텍스트를 제공하는 이름(선택 사항).
- lineitem/ResourceId: 사용량이 발생한 리소스의 식별자입니다. CUR에 리소스 ID별 분류가 포함된 경우 사용할 수 있습니다.
- lineitem/UsageAmount: 지정된 기간 동안 발생한 사용량입니다.
- lineitem/UnblendedCost: 이 사용량에 발생한 비용입니다.
- lineitem/LineItemDescription: 행 항목의 텍스트 설명입니다.

CUR 데이터 딕셔너리에 대한 자세한 내용은 Cost and Usage Report (CUR) 2.0을 참조하세요. 정확한 이름은 상황에 따라 달라진다는 점에 유의하세요.

`UsageType`은 `ReadCapacityUnit-Hrs`, `USW2-ReadRequestUnits`, `EU-WriteCapacityUnit-Hrs` 또는 `USE1-TimedPITRStorage-ByteHrs` 등의 값을 가진 문자열입니다. 각 사용량 유형은 선택적 리전 접두사로 시작합니다. 없는 경우 us-east-1 리전을 나타냅니다. 리전 접두사가 있는 경우 아래 테이블은 축약된 청구 리전 코드를 일반적인 리전 코드 및 이름에 매핑한 것입니다.

예를 들어, `USW2-ReadRequestUnits`라는 사용량은 us-west-2에서 사용된 읽기 요청 단위를 나타냅니다.

| 결제 리전 코드 | 리전 코드 | 리전 이름 |
|---|---|---|
| AFS1 | af-south-1 | 아프리카(케이프타운) |
| APE1 | ap-east-1 | 아시아 태평양(홍콩) |
| APN1 | ap-northeast-1 | 아시아 태평양(도쿄) |
| APN2 | ap-northeast-2 | 아시아 태평양(서울) |
| APN3 | ap-northeast-3 | 아시아 태평양(오사카) |
| APS1 | ap-southeast-1 | 아시아 태평양(싱가포르) |
| APS2 | ap-southeast-2 | 아시아 태평양(시드니) |
| APS3 | ap-south-1 | 아시아 태평양(뭄바이) |

| 결제 리전 코드 | 리전 코드 | 리전 이름 |
|---|---|---|
| APS4 | ap-southeast-3 | 아시아 태평양(자카르타) |
| APS5 | ap-south-2 | 아시아 태평양(하이데라바드) |
| APS6 | ap-southeast-4 | 아시아 태평양(멜버른) |
| CAN1 | ca-central-1 | 캐나다(중부) |
| EU | eu-west-1 | 유럽(아일랜드) |
| EUC1 | eu-central-1 | 유럽(프랑크푸르트) |
| EUC2 | eu-central-2 | 유럽(취리히) |
| EUN1 | eu-north-1 | 유럽(스톡홀름) |
| EUS1 | eu-south-1 | 유럽(밀라노) |
| EUS2 | eu-south-2 | 유럽(스페인) |
| EUW1 | eu-west-1 | 유럽(아일랜드) |
| EUW2 | eu-west-2 | 유럽(런던) |
| EUW3 | eu-west-3 | 유럽(파리) |
| ILC1 | Il-central-1 | 이스라엘(텔아비브) |
| MEC1 | me-central-1 | 중동(UAE) |
| MES1 | me-south-1 | 중동(바레인) |
| SAE1 | sa-east-1 | 남아메리카(상파울루) |
| USE1(기본값) | us-east-1 | 미국 동부(버지니아 북부) |
| USE2 | us-east-2 | 미국 동부(오하이오) |
| UGE1 | us-gov-east-1 | 미국 정부 동부 |

| 결제 리전 코드 | 리전 코드 | 리전 이름 |
|---|---|---|
| UGW1 | us-gov-west-1 | 미국 정부 서부 |
| USW1 | us-west-1 | 미국 서부(캘리포니아 북부) |
| USW2 | us-west-2 | 미국 서부(오리건) |

다음 섹션에서는 DynamoDB 요금을 살펴볼 때 `REG-UsageType` 패턴을 사용합니다. 여기서 REG는 사용량이 발생한 리전을 지정하고 usageType은 요금 유형의 코드입니다. 예를 들어 CSV 파일에 `USW1- ReadCapacityUnit-Hrs`에 대한 행 항목이 표시되면 US-West-1에서 프로비저닝된 읽기 용량에 대한 사용량이 발생한 것입니다. 이 경우 목록에 `REG-ReadCapacityUnit-Hrs`가 표시됩니다.

주제

- 처리량 용량
- 스트림
- 스토리지
- 백업 및 복원
- 데이터 전송
- CloudWatch Contributor Insights
- DynamoDB Accelerator(DAX)

## 처리량 용량

프로비저닝된 용량 읽기 및 쓰기

프로비저닝된 용량 모드에서 DynamoDB 테이블을 생성할 때는 애플리케이션에 필요할 것으로 예상되는 읽기 및 쓰기 용량을 지정합니다. 사용량 유형은 테이블 클래스(Standard 또는 Standard-Infrequent Access)에 따라 다릅니다. 초당 사용률을 기준으로 읽기 및 쓰기를 프로비저닝하지만 프로비저닝된 용량을 기준으로 시간당 요금이 부과됩니다.

| UsageType | 단위 | Granularity | 설명 |
|---|---|---|---|
| REG-ReadCapacityUnit-Hrs | RCU 시간 | 시간 | Standard 테이블 클래스를 사용하여 프로비 |

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| | | | 저닝된 용량 모드의 읽기 요금을 부과합니다. |
| REG-IA-ReadCapacityUnit-Hrs | RCU 시간 | 시간 | Standard-IA 테이블 클래스를 사용하여 프로비저닝된 용량 모드의 읽기 요금을 부과합니다. |
| REG-WriteCapacityUnit-Hrs | WCU 시간 | 시간 | Standard 테이블 클래스를 사용하여 프로비저닝된 용량 모드의 쓰기 요금을 부과합니다. |
| REG-IA-WriteCapacityUnit-Hrs | WCU 시간 | 시간 | Standard-IA 테이블 클래스를 사용하여 프로비저닝된 용량 모드의 쓰기 요금을 부과합니다. |

예약 용량 읽기 및 쓰기

예약 용량을 구입할 경우 선결제 요금을 1회 지불하고 일정 기간에 대해 최소 프로비저닝된 사용량 수준을 약정합니다. 예약 용량은 할인된 시간당 요금으로 청구됩니다. 예약 용량을 초과하여 프로비저닝하는 용량에는 프로비저닝된 용량 표준 요금이 청구됩니다. 예약 용량은 Standard 테이블 클래스를 사용하는 DynamoDB 테이블의 단일 리전에 프로비저닝된 읽기 및 쓰기 용량 단위(RCU 및 WCU)에 사용할 수 있습니다. 1년 및 3년 예약 용량 모두 동일한 SKU를 사용하여 청구됩니다.

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-Heavy Usage:dynamodb.read | RCU 시간 | 선결제 후 월별 | 예약 용량 읽기 요금: 선결제 금액은 한 번만 청구되며 매월 초에 해당 월의 할인된 약정 RCU 시간을 모두 포함하는 월별 요금이 부 |

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| | | | 과됩니다. 여기에 상응하는 비용이 전혀 들지 않는 REG-ReadCapacityUnit-Hrs 행 항목이 있습니다. |
| REG-HeavyUsage:dynamodb.write | WCU 시간 | 선결제 후 월별 | 예약 용량 쓰기 요금: 선결제 금액은 한 번만 청구되며 매월 초에 해당 월의 할인된 약정 WCU 시간을 모두 포함하는 월별 요금이 부과됩니다. 여기에 상응하는 비용이 전혀 들지 않는 REG-WriteCapacityUnit-Hrs 행 항목이 있습니다. |

온디맨드 용량 읽기 및 쓰기

온디맨드 용량 모드로 DynamoDB 테이블을 생성할 경우 애플리케이션에서 수행하는 읽기 및 쓰기에 대해서만 요금을 지불합니다. 읽기 및 쓰기 요청 요금은 테이블 클래스에 따라 다릅니다.

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-ReadRequestUnits | RRU | 단위 | Standard 테이블 클래스를 사용하는 온디맨드 용량 모드의 읽기 요금을 부과합니다. |
| REG-IA-ReadRequestUnits | RRU | 단위 | Standard-IA 테이블 클래스를 사용하는 온디맨드 용량 모드의 읽기 요금을 부과합니다. |

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-WriteRequestUnits | WRU | 단위 | Standard 테이블 클래스를 사용하는 온디맨드 용량 모드의 쓰기 요금을 부과합니다. |
| REG-IA-WriteRequestUnits | WRU | 단위 | Standard-IA 테이블 클래스를 사용하는 온디맨드 용량 모드의 쓰기 요금을 부과합니다. |

글로벌 테이블 읽기 및 쓰기

DynamoDB는 각 복제본 테이블에 사용된 리소스를 기준으로 글로벌 테이블 사용에 대한 요금을 부과합니다. 프로비저닝된 글로벌 테이블의 경우 글로벌 테이블에 대한 쓰기 요청은 표준 WCU 대신 복제된 WCU(rWCU)로 측정되며 글로벌 테이블의 글로벌 보조 인덱스에 대한 쓰기는 WCU로 측정됩니다. 온디맨드 글로벌 테이블의 경우 쓰기 요청은 표준 WRU 대신 복제된 WRU(rWRU) 로 측정됩니다. 복제에 사용되는 rWCU 또는 rWRU 수는 사용 중인 글로벌 테이블 버전에 따라 다릅니다. 요금은 테이블 클래스에 따라 다릅니다.

글로벌 보조 인덱스(GSI)에 대한 쓰기에는 표준 쓰기 단위(WCU 및 WRU)를 사용하여 요금이 부과됩니다. 읽기 요청 및 데이터 스토리지는 단일 리전 테이블과 동일하게 청구됩니다.

테이블 복제본을 추가하여 새 리전에서 글로벌 테이블을 생성하거나 확장하는 경우, DynamoDB는 추가된 리전의 테이블 복원 비용을 복원된 데이터의 GB당 부과합니다. 복원된 데이터는 REG-RestoreDataSize-Bytes로 요금이 부과됩니다. 자세한 내용은 DynamoDB에 대한 백업 및 복원 섹션을 참조하세요. 리전 간 복제 및 데이터가 포함된 테이블에 복제본을 추가하는 경우에도 데이터 송신 요금이 부과됩니다.

DynamoDB 글로벌 테이블에 온디맨드 용량 모드를 선택하면 애플리케이션이 각 복제본 테이블에서 사용하는 리소스에 대한 비용만 지불하면 됩니다.

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-ReplWriteCapacityUnit-Hrs | rWCU 시간 | 시간 | 글로벌 테이블, 프로비저닝됨, Standard 테이블 클래스 |

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-IA-ReplWriteCapacityUnit-Hrs | rWCU 시간 | 시간 | 글로벌 테이블, 프로비저닝됨, Standard-IA 테이블 클래스 |
| REG-ReplWriteRequestUnits | rWRU | 단위 | 글로벌 테이블, 온디맨드, Standard 테이블 클래스 |
| REG-IA-ReplWriteRequestUnits | rWRU | 단위 | 글로벌 테이블, 온디맨드, Standard-IA 테이블 클래스 |

## 스트림

DynamoDB에는 DynamoDB Streams와 Kinesis라는 두 가지 스트리밍 기술이 있습니다. 각각 별도의 요금 체계가 적용됩니다.

DynamoDB Streams는 읽기 요청 단위로 데이터 읽기에 대해 요금을 부과합니다. 각 `GetRecords` API 직접 호출은 스트림 읽기 요청으로 청구됩니다. DynamoDB 트리거의 일부로 AWS Lambda에서 또는 복제의 일부로 DynamoDB 글로벌 테이블에서 호출한 `GetRecords` API 직접 호출에 대해서는 요금이 부과되지 않습니다.

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-Streams-RequestsCount | 개수 | 단위 | DynamoDB Streams에 대한 읽기 요청 단위 |

Amazon Kinesis Data Streams에서는 변경 데이터 캡처 단위로 요금을 부과합니다. DynamoDB는 각 쓰기(최대 1KB)에 대해 변경 데이터 캡처 단위 1개를 부과합니다. 항목이 1KB보다 크면 변경 데이터 캡처 단위가 추가로 필요합니다. 테이블의 처리량 용량을 관리할 필요 없이 애플리케이션이 수행한 쓰기에 대해서만 비용을 지불하면 됩니다.

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-ChangeDataCaptureUnits-Kinesis | CDC 단위 | 단위 | Kinesis Data Streams에 대한 변경 데이터 캡처 단위 |

## 스토리지

DynamoDB는 데이터의 원시 바이트 크기와 활성화한 기능에 따라 항목당 스토리지 오버헤드를 추가하여 청구 가능한 데이터의 크기를 측정합니다.

> **Note**
>
> CUR의 스토리지 사용량 값은 `DescribeTable`을 사용할 때의 스토리지 값에 비해 더 높습니다. `DescribeTable`은 항목당 스토리지 오버헤드를 포함하지 않기 때문입니다.

스토리지는 시간당 계산되지만 시간당 요금의 평균을 바탕으로 계산된 월별 요금이 부과됩니다.

`UsageType` 스토리지는 `ByteHrs`를 접미사로 사용하지만 CUR의 스토리지 사용량은 GB 단위로 측정되며 월별 GB로 요금이 책정됩니다.

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-TimedStorage-ByteHrs | GB | 월 | Standard 테이블 클래스를 사용하는 테이블의 DynamoDB 테이블 및 인덱스가 사용하는 스토리지의 양입니다. |
| REG-IA-TimedStorage- ByteHrs | GB | 월 | Standard-IA 테이블 클래스를 사용하는 테이블의 DynamoDB 테이블 및 인덱스에 사용되는 스토리지의 양입니다. |

# 백업 및 복원

DynamoDB는 시점 복구(PITR) 백업과 온디맨드 백업이라는 두 가지 유형의 백업을 제공합니다. 또한 사용자는 이러한 백업에서 DynamoDB 테이블로 복원할 수 있습니다. 아래 요금은 백업과 복원 모두에 적용됩니다.

백업 스토리지 요금은 매월 1일에 발생하며 백업이 추가 또는 제거됨에 따라 한 달 내내 조정이 이루어집니다. 자세한 내용은 Understanding Amazon DynamoDB On-demand Backups and Billing 블로그를 참조하세요.

| UsageType | 단위 | Granularity | 설명 |
|---|---|---|---|
| REG-TimedBackupStorage-ByteHrs | GB | 월 | DynamoDB 테이블 및 로컬 보조 인덱스의 온디맨드 백업에 사용되는 스토리지입니다. |
| TimedPITRStorage-ByteHrs | GB | 월 | 시점 복구(PITR) 백업에 사용되는 스토리지입니다. DynamoDB는 PITR이 활성화되어 있는 경우 한 달 내내 지속적으로 PITR 지원 테이블 크기를 모니터링하여 백업 요금 및 청구 요금을 결정합니다. |
| REG-RestoreDataSize-Bytes | GB | 크기 | DynamoDB 백업에서 복원된 총 데이터 크기(테이블 데이터, 로컬 보조 인덱스 및 글로벌 보조 인덱스 포함)로, GB 단위로 측정됩니다. |

## AWS Backup

AWS Backup은 클라우드 및 온프레미스에서 AWS 서비스 전반의 데이터 백업을 쉽게 중앙 집중화하고 자동화할 수 있는 완전관리형 백업 서비스입니다. AWS Backup에서는 스토리지(웜 스토리지 또는 콜드 스토리지), 복원 작업 및 리전 간 데이터 전송에 대한 요금이 부과됩니다. 다음 UsageType 요금은 'AmazonDynamoDB'가 아닌 'AWSBackup' ProductCode 아래에 표시됩니다.

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-WarmStorage-ByteHrs-DynamoDB | GB | 월 | 한 달 내내 AWS Backup에서 관리하는 DynamoDB 백업에 사용되는 스토리지로, 월별 GB 단위로 측정됩니다. |
| REG-CrossRegion-WarmBytes-DynamoDB | GB | 크기 | 동일한 계정 내의 AWS 리전 또는 다른 AWS 계정으로 전송된 데이터입니다. 한 리전에서 다른 리전으로 백업을 복사할 경우 리전 간 전송 요금이 발생합니다. 요금은 항상 데이터를 발송한 계정에 청구됩니다. |
| REG-Restore-WarmBytes-DynamoDB | GB | 크기 | 웜 스토리지에서 복원된 총 데이터 크기로, GB 단위로 측정됩니다. |
| REG-ColdStorage-ByteHrs-DynamoDB | GB | 월 | 한 달 내내 AWS Backup에서 관리하는 DynamoDB 백업에 사용되는 콜드 스토리지 |

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| | | | 로, 월별 GB 단위로 측정됩니다. |
| REG-Restore-ColdBytes-DynamoDB | GB | 월 | 콜드 스토리지에서 복원된 총 데이터 크기로, GB 단위로 측정됩니다. |

## 내보내기 및 가져오기

DynamoDB에서 Amazon S3로 데이터를 내보내거나 Amazon S3에서 새 DynamoDB 테이블로 데이터를 가져올 수 있습니다.

`UsageType`은 접미사로 `Bytes`를 사용하지만 CUR의 내보내기 및 가져오기 사용량은 GB 단위로 측정되고 요금이 책정됩니다.

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-ExportDataSize-Bytes | GB | 크기 | S3로 데이터 내보내기에 대한 요금입니다. DynamoDB에서는 내보내기가 생성된 특정 시점의 DynamoDB 기본 테이블(테이블 데이터 및 로컬 보조 인덱스) 크기를 기준으로 내보낸 데이터에 대한 요금을 부과합니다. |
| REG-ImportDataSize-Bytes | GB | 크기 | S3에서 데이터 가져오기에 대한 요금입니다. 크기는 Amazon S3 내 데이터의 압축되지 않은 객체 크기를 기준으로 계산됩니다. GSI를 사용하는 테이블로 가 |

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| | | | 져오는 데는 추가 요금이 부과되지 않습니다. |
| REG-IncrementalExportDataSize-Bytes | GB | 크기 | 증분 내보내기를 생성하기 위해 연속 백업에서 처리한 데이터 크기에 대한 요금입니다. |

## 데이터 전송

데이터 전송 활동은 DynamoDB 서비스와 관련된 것으로 나타날 수 있습니다. DynamoDB는 인바운드 데이터 전송에 대해 요금을 부과하지 않으며, DynamoDB와 동일 AWS 리전 내의 다른 AWS 서비스 간에 전송된 데이터에 대해서도 요금을 부과하지 않습니다(즉, GB당 0.00 USD). AWS 리전 간(예: 미국 동부(버지니아 북부) 리전 및 EU(아일랜드) 리전의 Amazon EC2 간) 전송된 데이터는 양쪽에 요금이 부과됩니다.

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-DataTransfer-In-Bytes | GB | 단위 | 인터넷에서 DynamoDB로 전송되는 데이터입니다. |
| REG-DataTransfer-Out-Bytes | GB | 단위 | DynamoDB에서 인터넷으로 전송되는 데이터입니다. |

## CloudWatch Contributor Insights

DynamoDB에 대한 CloudWatch Contributor Insights는 DynamoDB 테이블에서 가장 자주 액세스하고 제한되는 키를 한눈에 확인할 수 있는 진단 도구입니다. 다음 `UsageType` 요금은 'AmazonDynamoDB'가 아닌 'AmazonCloudWatch' ProductCode 아래에 표시됩니다.

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-CW:ContributorEventsManaged | 처리된 이벤트 | 단위 | 처리된 DynamoDB 이벤트의 양입니다. 예를 들어 CloudWatch Contributor Insights가 활성화된 테이블의 경우 항목을 읽거나 쓸 때마다 하나의 이벤트로 계산됩니다. 테이블에 정렬 키가 있는 경우 두 이벤트에 대한 요금이 부과됩니다. |
| REG-CW:ContributorRulesManaged | 규칙 수 | 월 | DynamoDB는 CloudWatch Contributor Insights를 활성화하면 가장 많이 액세스한 항목과 가장 제한이 심한 키를 식별하는 규칙을 생성합니다. 이 요금은 CloudWatch 기여자 인사이트 로깅을 위해 구성된 각 엔터티(테이블 및 GSI)에 추가된 규칙에 대해 부과됩니다. |

## DynamoDB Accelerator(DAX)

DynamoDB Accelerator(DAX)에는 서비스에 대해 선택한 인스턴스 유형을 기준으로 시간당 요금이 부과됩니다. 아래 요금은 프로비저닝된 DynamoDB Accelerator 인스턴스를 기준으로 합니다. 다음 `UsageType` 요금은 'AmazonDynamoDB'가 아닌 'AmazonDAX' ProductCode 아래에 표시됩니다.

| UsageType | 단위 | Granularity | 설명 |
| --- | --- | --- | --- |
| REG-NodeUsage:dax-<INSTANCETYPE> | 노드 시간 | 시간 | 특정 인스턴스 유형의 시간당 사용량입니다. 요금은 노드가 시작된 시점부터 종료될 때까지 사용된 노드 시간당 요금입니다. 노드 시간을 일부 소비한 경우 1시간으로 요금이 부과됩니다. DAX는 DAX 클러스터의 각 노드에 대해 요금을 부과합니다. 여러 노드가 있는 클러스터가 있는 경우 결제 보고서에 여러 행 항목이 표시됩니다. |

인스턴스 유형은 다음 목록의 값 중 하나입니다. 노드 유형에 대한 자세한 내용은 노드 섹션을 참조하세요.

- r3.2xlarge, r4.8xlarge 또는 r5.8xlarge
- r3.4xlarge, r4.large 또는 r5.large
- r3.8xlarge, r4.xlarge 또는 r5.xlarge
- r3.2xlarge, r5.12xlarge 또는 t2.medium
- r3.4xlarge, r4.large 또는 r5.large
- r3.xlarge, r5.16xlarge 또는 t2.small
- r4.16xlarge, r5.24xlarge 또는 t3.medium
- r4.2xlarge, r5.2xlarge 또는 t3.small
- r4.4xlarge 또는 r5.4xlarge

# DynamoDB 테이블을 한 계정에서 다른 계정으로 마이그레이션

한 계정에서 다른 계정으로 Amazon DynamoDB 테이블을 마이그레이션하여 다중 계정 전략 또는 백업 전략을 구현할 수 있습니다. 테스트, 디버깅 또는 규정 준수를 위해 이 작업을 수행할 수도 있습니다. 일반적인 사용 사례는 각 환경이 서로 다른 AWS 계정을 사용하는 프로덕션, 스테이징, 테스트 및 개발 환경에서 DynamoDB 테이블을 복사하는 것입니다.

DynamoDB는 한 AWS 계정에서 다른 계정으로 테이블을 마이그레이션하는 2가지 옵션을 제공합니다.

- 교차 계정 백업 및 복구를 위한 AWS Backup: AWS Backup는 여러 AWS 서비스에서 백업을 중앙 집중식으로 관리할 수 있는 완전 관리형 백업 서비스입니다. 교차 계정 백업 및 복원 기능을 사용하면 한 계정의 DynamoDB 테이블을 백업하고 동일한 AWS Organization의 다른 계정으로 백업을 복원할 수 있습니다.
- Amazon S3로 DynamoDB 내보내기 및 가져오기: Amazon S3로 DynamoDB 내보내기 및 가져오기 기능을 사용하면 Amazon S3 버킷으로 전체 내보내기를 수행한 다음 해당 데이터를 다른 AWS 계정의 새 테이블로 가져올 수 있습니다. 이 접근 방식은 동일한 AWS Organization에 속하지 않은 계정 간에 마이그레이션해야 하거나 AWS Backup를 사용하지 않으려는 경우에 적합합니다.

> **Note**
>
> Amazon S3에서 가져오기는 로컬 보조 인덱스(LSI)가 있는 테이블을 지원하지 않지만, 글로벌 보조 인덱스(GSI)는 지원합니다. LSI 및 GSI에 대한 자세한 내용은 DynamoDB에서 보조 인덱스를 사용하여 데이터 액세스 개선 섹션을 참조하세요.

주제


- 교차 계정 백업 및 복원 시 AWS Backup을 사용하여 테이블 마이그레이션
- S3로 내보내기를 사용하여 테이블을 마이그레이션하고 S3에서 가져오기


## 교차 계정 백업 및 복원 시 AWS Backup을 사용하여 테이블 마이그레이션

사전 조건

- 소스 및 대상 AWS 계정은 AWS Organizations 서비스에서 동일한 조직에 속해 있어야 합니다.
- 유효한 AWS Identity and Access Management(IAM) 권한으로 AWS Backup 볼트를 생성하고 사용할 수 있어야 합니다.

교차 계정 백업 설정에 대한 자세한 내용은 AWS 계정 간 백업 복사본 생성을 참조하세요.

요금 정보

AWS는 백업(테이블 크기 기준), AWS 리전 간의 모든 데이터 복사(데이터 양 기준), 복원(데이터 양 기준), 사용 중인 모든 스토리지 요금을 청구합니다. 비용이 지속적으로 청구되지 않도록 하려면 복원 후 필요하지 않은 백업을 삭제하면 됩니다.

요금에 대한 자세한 내용은 AWS Backup 요금을 참조하세요.

## 1단계: DynamoDB 및 교차 계정 백업을 위한 고급 기능 활성화

1. 소스 및 대상 AWS 계정 모두에서 AWS Management Console에 액세스하고 AWS Backup 콘솔을 엽니다.
2. 설정 옵션을 선택합니다.
3. Amazon DynamoDB 백업용 고급 기능에서 고급 기능이 활성화되어 있는지 확인합니다. 활성화되어 있지 않으면 활성화를 선택합니다.
4. 교차 계정 관리에서 교차 계정 백업에 대하여 설정을 선택합니다.

## 2단계: 소스 계정 및 대상 계정에 백업 볼트 생성

1. 소스 AWS 계정에서 AWS Backup 콘솔을 엽니다.
2. 백업 볼트를 선택합니다.
3. 백업 저장소 생성을 선택합니다.
4. 생성된 백업 볼트와 대상 AWS 계정의 Amazon 리소스 이름(ARN)을 복사하고 저장합니다.
5. 계정 간에 DynamoDB 테이블 백업을 복사할 때는 소스 및 대상 백업 볼트의 ARN이 모두 필요합니다.

## 3단계: 소스 계정에 DynamoDB 테이블 백업 생성

1. AWS Backup 대시보드 페이지에서 온디맨드 백업 생성을 선택합니다.
2. 설정 섹션에서 리소스 유형으로 DynamoDB를 선택한 다음 테이블 이름을 선택합니다.
3. 백업 볼트 드롭다운 목록에서 소스 계정에서 생성한 백업 볼트를 선택합니다.
4. 원하는 보존 기간을 선택합니다.
5. 온디맨드 백업 생성을 선택합니다.

6. AWS Backup 작업 페이지의 Backup 작업 탭에서 백업 작업의 상태를 모니터링합니다.

### 4단계: DynamoDB 테이블 백업을 소스 계정에서 대상 계정으로 복사

1. 백업 작업이 완료되면 소스 계정에서 AWS Backup 콘솔을 열고 백업 볼트를 선택합니다.
2. 백업에서 DynamoDB 테이블 백업을 선택합니다. 작업을 선택하고 복사를 선택합니다.
3. 대상 계정의 AWS 리전을 입력합니다.
4. 외부 볼트 ARN의 경우 대상 계정에서 생성한 백업 볼트의 ARN을 입력합니다.
5. 대상 계정 백업 볼트에서 액세스를 활성화합니다.

### 5단계: 대상 계정에서 DynamoDB 테이블 백업 복원

1. 대상 AWS 계정에서 AWS Backup 콘솔을 열고 백업 볼트를 선택합니다.
2. 백업에서 소스 계정을 통해 복사한 백업을 선택합니다. 작업을 선택한 다음, 복원을 선택합니다.
3. 새 DynamoDB 테이블의 이름, 새 테이블에 적용할 암호화, 복원을 암호화할 키 및 기타 옵션을 입력합니다.
4. 복원이 완료되면 테이블 상태가 활성으로 표시됩니다.

## S3로 내보내기를 사용하여 테이블을 마이그레이션하고 S3에서 가져오기

사전 조건

- S3로 내보내기를 수행하려면 테이블에 대해 시점 복구(PITR)를 활성화해야 합니다. 자세한 내용은 DynamoDB에서 특정 시점 복구 사용 섹션을 참조하세요.
- 내보내기를 수행할 수 있는 유효한 IAM 권한이 있어야 합니다. 자세한 내용은 DynamoDB에서 테이블 내보내기 요청 섹션을 참조하세요.
- 가져오기를 수행하기에 충분히 유효한 IAM 권한이 있어야 합니다. 자세한 내용은 DynamoDB에서 테이블 가져오기 요청 섹션을 참조하세요.

요금 정보

AWS는 PITR에 대한 요금을 테이블 크기 및 PITR이 활성화된 기간을 기준으로 부과합니다. 내보내기 외에 PITR이 필요하지 않은 경우 내보내기가 끝난 후 PITR을 설정 해제할 수 있습니다. AWS에서는 또한 S3에 대한 요청, 내보낸 데이터를 S3에 저장, 가져오기에 대해서도 가져온 데이터의 압축되지 않은 크기 기준으로 요금을 부과합니다.

DynamoDB 요금에 대한 자세한 내용은 DynamoDB 요금을 참조하세요.

> **Note**
>
> S3에서 DynamoDB로 가져올 때는 개체 크기 및 수에 제한이 있습니다. 자세한 내용은 가져오기 할당량 섹션을 참조하세요.

## Amazon S3로 테이블 내보내기 요청

1. AWS Management Console에 로그인하고 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 S3로 내보내기(Exports to S3)를 선택합니다.
3. 소스 테이블과 대상 S3 버킷을 선택합니다. 대상 계정 버킷의 URL을 `s3://bucketname/prefix` 형식으로 입력합니다. `/prefix`는 대상 버킷을 체계적으로 구성된 상태로 유지하는 데 도움이 되는 선택적 폴더입니다.
4. 전체 내보내기를 선택합니다. 전체 내보내기는 지정한 시점의 테이블의 전체 테이블 스냅샷을 출력합니다.
   a. 최신 전체 테이블 스냅샷을 내보내려면 현재 시간을 선택합니다.
   b. 내보낸 파일 형식에서 DynamoDB JSON과 Amazon Ion 중에 선택합니다. 기본 옵션은 DynamoDB JSON입니다.
5. 내보내기 버튼을 클릭하여 내보내기를 시작합니다.
6. 소규모 테이블 내보내기는 몇 분 안에 완료되지만, 테라바이트 범위의 테이블은 1시간 이상 걸릴 수 있습니다.

## Amazon S3에서 테이블 가져오기 요청

1. AWS Management Console에 로그인하고 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 S3에서 가져오기(Import from S3)를 선택합니다.
3. 표시되는 페이지에서 S3에서 가져오기(Import from S3)를 선택합니다.
4. Amazon S3 소스 URL을 입력합니다. S3 찾아보기 버튼을 사용하여 찾을 수도 있습니다. 예상 경로의 형식은 `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/data/`입니다.
5. S3 버킷 소유자(S3 bucket owner)인지 여부를 지정합니다.
6. 가져오기 파일 압축에서 내보내기와 일치하는 GZIP을 선택합니다.
7. 가져오기 파일 형식에서 내보내기와 일치하는 DynamoDB JSON을 선택합니다.

8. 다음을 선택합니다. 테이블 세부 정보 지정에서 데이터를 저장하기 위해 생성할 새 테이블에 대한 옵션을 선택합니다.
9. 다음을 선택합니다. 테이블 설정 구성에서 해당하는 경우 추가 테이블 설정을 사용자 지정합니다.
10. 다음(Next)을 다시 선택하여 가져오기 옵션을 검토하고 가져오기(Import)를 클릭하여 가져오기 작업을 시작합니다. S3에서 가져오기 아래에 가져오는 중 상태로 새 테이블이 나열됩니다. 이 시간 동안에는 테이블에 액세스할 수 없습니다. 소규모 가져오기는 몇 분 안에 완료되지만, 테라바이트 범위의 테이블은 1시간 이상 걸릴 수 있습니다.
11. 가져오기가 완료되면 상태가 활성으로 표시되고 테이블 사용을 시작할 수 있습니다.

### 마이그레이션 중 테이블 동기화 유지

마이그레이션하는 동안 소스 테이블에 대한 쓰기 작업을 일시 중지할 수 있는 경우 마이그레이션 후 소스와 출력이 정확히 일치해야 합니다. 쓰기 작업을 일시 중지할 수 없는 경우 일반적으로 마이그레이션 후 대상 테이블이 소스보다 약간 뒤처집니다. 소스 테이블을 따라잡으려면 스트리밍(DynamoDB Streams 또는 DynamoDB용 Kinesis Data Streams)을 사용하여 백업 또는 내보내기 이후 소스 테이블에서 발생한 쓰기를 재생할 수 있습니다.

소스 테이블을 S3로 내보낸 타임스탬프 이전에 스트림 레코드 읽기를 시작해야 합니다. 예를 들어, S3로 내보내기가 오후 2시에 발생하고 대상 테이블로의 가져오기가 오후 11시에 완료되었다면 오후 1시 58분에 DynamoDB 스트림 읽기를 시작해야 합니다. 변경 데이터 캡처를 위한 스트리밍 옵션 테이블에는 각 스트리밍 모델의 기능이 간략하게 나와 있습니다.

Lambda와 함께 DynamoDB Streams를 사용하면 소스 및 대상 DynamoDB 테이블 간에 데이터를 간편하게 동기화할 수 있습니다. Lambda 함수를 사용하여 대상 테이블의 각 쓰기를 재생할 수 있습니다.

> **Note**
>
> 항목은 DynamoDB Streams에 24시간 동안 보관되므로, 해당 기간 내에 백업 및 복원 또는 내보내기와 가져오기를 완료할 계획을 세워야 합니다.

## DAX를 DynamoDB 애플리케이션과 통합하기 위한 권장 가이드

DynamoDB Accelerator(DAX)는 DynamoDB와 호환되는 캐싱 서비스로, 읽기 중심의 애플리케이션처럼 까다로운 애플리케이션에 빠른 인 메모리 성능을 제공합니다. DAX를 사용하면 자주 요청되는 데이터에 액세스하는 응답 시간을 마이크로초 단위로 높일 수 있습니다. 이 DynamoDB Accelerator 권장

가이드는 DAX를 DynamoDB 애플리케이션과 통합하는 데 필요한 포괄적인 인사이트와 모범 사례를 제공합니다.

이 가이드에는 DAX를 처음 사용하거나 기존 구성을 최적화하려는 사용자를 위한 기초 지식이 나와 있습니다. 이 가이드에서는 다양한 주제(예: DAX 사용 시기 및 DAX 클러스터 생성)를 다룹니다. 또한, 프로젝트에 DAX를 효과적으로 구현하는 데 도움이 되는 실제 예제와 자세한 설명도 포함되어 있습니다. 마지막으로, 빠르고 확장 가능한 애플리케이션을 보장하기 위해 DAX 캐싱 기능을 극대화하고자 구현해야 하는 고급 전략도 제공합니다.

주제


- 사용 사례에 대한 DAX의 적합성 평가
- DAX 클라이언트 구성
- DAX 클러스터 구성
- DAX 클러스터 크기 조정
- 클러스터 배포
- 클러스터 작업 관리
- DAX 모니터링


# 사용 사례에 대한 DAX의 적합성 평가

이 섹션에서는 DAX를 사용해야 하는 시기와 이유를 설명합니다. 이 지침을 바탕으로 DAX를 DynamoDB와 통합하는 것이 애플리케이션의 워크로드 패턴, 성능 요건, 데이터 일관성 요구 사항에 유리한지 판단할 수 있습니다. 나아가 쓰기 작업이 많은 워크로드, 자주 액세스하지 않는 데이터 등 DAX가 적합하지 않을 수 있는 시나리오도 다룹니다.

이 섹션의 내용


- DAX를 선택해야 하는 시기와 이유
- DAX를 사용하지 말아야 할 경우


## DAX를 선택해야 하는 시기와 이유

여러 시나리오에서 애플리케이션 스택에 DAX를 추가하는 것을 고려해 볼 수 있습니다. 예를 들어, DAX를 사용하여 DynamoDB에 대한 읽기 요청의 전체 지연 시간을 줄이거나 테이블에서 동일한 데이터를 반복적으로 읽는 상황을 최소화할 수 있습니다. 다음 목록은 DAX와 DynamoDB의 통합을 활용할 수 있는 시나리오의 예를 보여줍니다.

- 고성능 요구 사항
  - 지연 시간이 짧은 읽기 - 애플리케이션에서 최종적으로 일관된 읽기를 보장하는 데 마이크로초 단위의 응답 시간이 필요한 경우 DAX 사용을 고려해야 합니다. 또한, DAX는 자주 읽는 데이터에 액세스하는 데 필요한 응답 시간을 크게 단축할 수 있습니다.
- 읽기 집약적인 워크로드
  - 읽기 중심의 애플리케이션 - 예를 들어, 읽기-쓰기 비율이 10:1 이상과 같이 높은 애플리케이션의 경우 DAX를 사용하면 캐시 적중률이 증가하고 오래된 데이터가 줄어듭니다. 이렇게 하면 테이블에 대한 읽기 횟수가 줄어듭니다. 애플리케이션에 쓰기 작업량이 많은 경우 캐시에서 오래된 데이터를 읽지 않으려면 캐시의 DynamoDB에서 TTL(Time To Live) 사용을 더 낮게 설정해야 합니다.
  - 일반 쿼리 캐싱 - 애플리케이션이 동일한 데이터(예: 전자 상거래 플랫폼의 인기 제품)를 자주 읽는 경우 DAX는 캐시에서 직접 이러한 요청을 처리할 수 있습니다.
- 급증하는 트래픽 패턴
  - 원활한 테이블 규모 조정 - DAX를 사용하면 갑작스러운 트래픽 급증으로 인한 영향을 완화할 수 있습니다. DAX는 DynamoDB 테이블 용량을 단계적으로 스케일 업할 수 있는 버퍼를 제공하므로, 읽기 제한의 위험이 줄어듭니다.
  - 각 항목에 대한 더 높은 읽기 처리량 - DynamoDB는 각 항목에 개별 파티션을 할당합니다. 하지만 파티션은 항목 수가 3,000RCU(읽기 용량 단위)에 도달하면 읽기 제한을 시작합니다. DAX를 사용하면 단일 항목의 읽기를 3,000RCU 이상으로 규모를 조정할 수 있습니다.
- 비용 최적화
  - DynamoDB 비용 절감 - DAX에서 읽으면 DynamoDB 테이블로 전송되는 읽기를 줄일 수 있으며, 이는 비용에 직접적인 영향을 미칠 수 있습니다. 캐시 적중률이 높으면 감소된 테이블 읽기 비용이 DAX 클러스터 비용을 초과하여 순 비용이 절감될 수 있습니다.
- 데이터 일관성 요구 사항
  - 최종 일관성 - DAX는 최종적으로 일관된 읽기를 지원합니다. 따라서 DAX는 즉각적인 일관성이 중요하지 않은 사용 사례에 적합합니다.
  - 라이트-스루 캐싱 - DAX에 대한 쓰기는 라이트-스루입니다. DAX가 항목이 DynamoDB에 기록되었음을 확인하면 항목 캐시에 해당 항목 버전을 유지합니다. 이 라이트-스루 메커니즘은 캐시와 데이터베이스 간의 데이터 일관성을 더 엄격하게 유지하는 데 도움이 되지만, 추가 DAX 클러스터 리소스를 사용합니다.

## DAX를 사용하지 말아야 할 경우

DAX는 강력하지만, 모든 시나리오에 적합하지는 않습니다. 다음 목록은 DAX와 DynamoDB를 통합하는 것이 적합하지 않은 시나리오의 예를 보여줍니다.

- 쓰기 중심의 워크로드 - DAX의 주요 이점은 읽기 속도를 높이는 것이지만, 쓰기는 읽기보다 DAX 리소스를 더 많이 사용합니다. 애플리케이션에서 쓰기 작업이 주로 발생하는 경우 DAX가 주는 이점이 제한될 수 있습니다.
- 데이터 읽기 빈도 낮음 - 애플리케이션이 데이터에 자주 액세스하지 않거나 거의 재사용되지 않는 데이터(콜드 데이터) 범위가 넓을 경우 cache hit ratio이 낮아질 가능성이 높습니다. 이 경우 캐시 유지 관리에 수반되는 오버헤드가 성능 향상을 정당화하는 데 걸림돌이 될 수 있습니다.
- 대량 읽기 또는 쓰기 - 애플리케이션이 개별 쓰기보다 대량 쓰기를 더 많이 수행하는 경우 DAX를 중심으로 작성해야 합니다. 또한, 대량 읽기의 경우 DynamoDB 테이블에 대해 직접 전체 테이블 스캔을 실행해야 합니다.
- 강력한 일관성 또는 트랜잭션 요구 사항 - DAX는 강력히 일관된 읽기 및 TransactGetItems 호출을 DynamoDB 테이블로 전달합니다. 클러스터 리소스를 사용하지 않으려면 DAX 클러스터를 대상으로 이러한 읽기 작업을 수행해야 합니다. 이렇게 읽은 항목은 캐시되지 않으므로, DAX를 통해 이러한 항목을 라우팅하는 것은 아무 소용이 없습니다.
- 성능 요구 사항이 보통인 단순 애플리케이션 - 성능 요구 사항이 보통이고 직접적인 DynamoDB 지연 시간을 허용하는 애플리케이션의 경우 DAX를 추가하는 데 따르는 복잡성과 비용이 필요하지 않을 수 있습니다. DynamoDB는 자체적으로 높은 처리량을 처리하고 10밀리초 미만의 성능을 제공합니다.
- 키-값 액세스를 넘어서는 복잡한 쿼리 요구 사항 - DAX는 키-값 액세스 패턴에 최적화되어 있습니다. 애플리케이션에 쿼리 및 스캔 작업과 같이 단순하지 않은 필터링 기능이 포함된 복잡한 쿼리 기능이 필요한 경우 DAX 캐싱이 주는 이점이 제한될 수 있습니다.

  이러한 상황에서는 Amazon ElastiCache(Redis OSS)를 대안으로 사용하세요. ElastiCache(Redis OSS)는 목록, 집합, 해시와 같은 고급 데이터 구조를 지원합니다. 나아가 게시/구독, 지리 공간 인덱스, 스크립팅과 같은 기능까지 제공합니다.
- 규정 준수 요구 사항 - DAX는 현재 DynamoDB와 동일한 규정 준수 인증을 제공하지 않습니다. 예를 들어, DAX는 아직 SOC 인증을 획득하지 못했습니다.

## DAX 클라이언트 구성

DAX 클러스터는 다양한 DAX SDK를 사용하여 액세스할 수 있는 인스턴스 기반 클러스터입니다. 각 SDK는 특정 애플리케이션 요구 사항을 충족하도록 requestTimeout 및 connections와 같은 구성 가능한 옵션을 개발자에게 제공합니다.

DAX 클라이언트를 구성할 때 중요한 고려 사항은 클라이언트 애플리케이션의 규모, 특히 DAX 서버 인스턴스 수 대비 클라이언트 인스턴스 수(최대 11개)입니다. 대규모 클라이언트 인스턴스 플릿은 DAX

서버 인스턴스에 대한 수많은 연결을 생성하여 과부하를 일으킬 수 있습니다. 이 가이드에서는 DAX 클라이언트 구성의 모범 사례를 간략하게 설명합니다.

## 모범 사례

1. 클라이언트 인스턴스 - 요청 전반에서 인스턴스를 재사용할 수 있도록 싱글톤 클라이언트 인스턴스를 구현합니다. 구현 세부 정보는 the section called “4단계: 샘플 애플리케이션 실행” 단원을 참조하세요.
2. 요청 제한 시간 - 애플리케이션은 업스트림 시스템의 지연 시간을 최소화하기 위해 요청 제한 시간을 낮춰야 하는 경우가 많지만 제한 시간을 너무 낮게 설정하면 문제가 발생할 수 있습니다. 제한 시간이 짧으면 DAX 서버에서 일시적으로 지연 시간이 급증할 때 서버 인스턴스에 자주 재연결될 수 있습니다. 시간 초과가 발생하면 DAX 클라이언트는 기존 서버 노드 연결을 종료하고 새 연결을 설정합니다. 연결 설정은 리소스 집약적이므로 연속으로 여러 차례 연결하면 DAX 서버에 과부하가 걸릴 수 있습니다. 다음과 같이 하는 것이 좋습니다:
   - 기본 요청 제한 시간 설정 유지.
   - 제한 시간이 더 짧아야 하는 경우 더 낮은 제한 시간 값을 갖는 별도의 애플리케이션 스레드를 구현하고 지수 백오프가 있는 재시도 메커니즘을 포함합니다.
3. 연결 제한 시간 - 대부분의 애플리케이션에서는 기본 연결 제한 시간 설정을 유지하는 것이 좋습니다.
4. 동시 연결 - JavaV2와 같은 특정 SDK를 사용하면 DAX 서버에 대한 동시 연결을 조정할 수 있습니다. 주요 고려 사항:
   - DAX 서버 인스턴스는 최대 40,000개의 동시 연결을 처리할 수 있습니다.
   - 기본 설정은 대부분의 사용 사례에 적합합니다.
   - 대규모 클라이언트 인스턴스와 높은 동시 연결이 결합되면 서버에 과부하가 걸릴 수 있습니다.
   - 동시 연결 값이 낮으면 서버 과부하 위험이 줄어듭니다.
   - 성능 계산 예:
     - 요청 지연 시간이 1ms라고 가정하면 각 연결은 이론적으로 초당 1,000개의 요청을 처리할 수 있습니다.
     - 3노드 클러스터의 경우 모든 노드에 연결하는 단일 클라이언트 인스턴스가 초당 3,000개의 요청을 처리할 수 있습니다.
     - 10개의 연결을 통해 클라이언트는 초당 약 30,000개의 요청을 처리할 수 있습니다.

       권장 사항 - 더 낮은 동시 연결 설정으로 시작하고 예상 프로덕션 워크로드 패턴을 사용한 성능 테스트를 통해 검증합니다.

# DAX 클러스터 구성

DAX 클러스터는 관리형 클러스터이지만, 애플리케이션 요구 사항에 맞게 구성을 조정할 수 있습니다. DynamoDB API 작업과 긴밀하게 통합되므로, 애플리케이션을 DAX와 통합할 때는 다음 사항을 고려해야 합니다.

이 섹션의 내용


- DAX 가격 책정
- 항목 캐시 및 쿼리 캐시
- 캐시에 대한 TTL 설정 선택
- DAX 클러스터로 여러 테이블 캐싱
- DAX 및 DynamoDB 글로벌 테이블의 데이터 복제
- DAX 리전 가용성
- DAX 캐싱 동작


## DAX 가격 책정

클러스터 비용은 프로비저닝한 노드의 수와 크기에 따라 달라집니다. 모든 노드는 클러스터에서 노드를 실행하는 시간별로 요금이 청구됩니다. 자세한 내용은 Amazon DynamoDB 요금을 참조하세요.

캐시 적중은 DynamoDB 비용을 발생시키지 않지만, DAX 클러스터 리소스에 영향을 미칩니다. 캐시 누락으로 인해 DynamoDB 읽기 비용이 발생하고 DAX 리소스가 필요합니다. 쓰기는 DynamoDB 쓰기 비용을 발생시키고 쓰기를 프록시하는 DAX 클러스터 리소스에 영향을 줍니다.

## 항목 캐시 및 쿼리 캐시

DAX는 항목 캐시와 쿼리 캐시를 유지합니다. 이러한 캐시 간의 차이점을 이해하면 해당 캐시가 애플리케이션에 제공하는 성능 및 일관성 특성을 결정하는 데 도움이 될 수 있습니다.

| 캐시 특성 | 항목 캐시 | 쿼리 캐시 |
| --- | --- | --- |
| 용도 | GetItem 및 BatchGetItem API 작업의 결과를 저장합니다. | 쿼리 및 스캔 API 작업의 결과를 저장합니다. 이러한 작업은 특정 항목 키 대신 쿼리 조건에 따라 여러 항목을 반환할 수 있습니다. |

| 캐시 특성 | 항목 캐시 | 쿼리 캐시 |
| --- | --- | --- |
| 액세스 유형 | 키 기반 액세스를 사용합니다.<br><br>애플리케이션이 `GetItem` 또는 `BatchGetItem` 을 사용하여 데이터를 요청하면 DAX는 먼저 요청된 항목의 프라이머리 키를 사용하여 항목 캐시를 확인합니다. 항목이 캐시되고 만료되지 않은 경우 DAX가 DynamoDB 테이블에 액세스하지 않고 즉시 항목을 반환합니다. | 파라미터 기반 액세스를 사용합니다.<br><br>DAX는 `Query` 및 `Scan` API 작업의 결과 집합을 캐싱합니다. DAX는 캐시의 동일한 쿼리 조건, 테이블, 인덱스를 포함하는 같은 파라미터를 사용하여 후속 요청을 처리합니다. 이렇게 하면 응답 시간과 DynamoDB 읽기 처리량 소비가 크게 줄어듭니다. |
| 캐시 무효화 | DAX는 다음 시나리오에서 업데이트된 항목을 DAX 클러스터 내 노드의 항목 캐시에 자동으로 복제합니다.<br><br>• 캐시를 통해 항목 업데이트를 작성합니다.<br>• 테이블에서 업데이트된 항목 버전을 살펴보세요. | 쿼리 캐시는 항목 캐시보다 무효화하기가 더 어렵습니다. 항목 업데이트가 캐시된 쿼리 또는 스캔에 직접 매핑되지 않을 수 있습니다. 데이터 일관성을 유지하려면 쿼리 캐시 TTL을 신중하게 조정해야 합니다. DAX 또는 기본 테이블을 통한 쓰기는 TTL이 이전에 캐시된 응답을 만료하고 DAX가 DynamoDB에 대해 새 쿼리를 수행할 때까지 쿼리 캐시에 반영되지 않습니다. |
| 글로벌 보조 인덱스 | `GetItem` API 작업은 로컬 보조 인덱스 또는 글로벌 보조 인덱스에서 지원되지 않으므로 항목 캐시는 기본 테이블의 읽기만 캐시합니다. | 쿼리 캐시는 테이블과 인덱스 모두에 대해 쿼리를 캐시합니다. |

## 캐시에 대한 TTL 설정 선택

TTL은 데이터가 오래되기 전에 캐시에 저장되는 기간을 결정합니다. 이 기간이 지나면 다음 요청 시 데이터가 자동으로 새로 고쳐집니다. DAX 캐시에 적합한 TTL 설정을 선택하려면 애플리케이션 성능 최적화와 데이터 일관성 간의 균형을 맞춰야 합니다. 모든 애플리케이션에 적용되는 범용 TTL 설정은 존재하지 않으므로, 최적의 TTL 설정은 애플리케이션의 구체적인 특성 및 요구 사항에 따라 달라집니다. 이 권장 가이드를 바탕으로 보수적인 TTL 설정으로 시작하는 것이 좋습니다. 그런 다음 애플리케이션의 성능 데이터 및 인사이트를 기반으로 TTL 설정을 반복적으로 조정하세요.

DAX는 항목 캐시에 대해 가장 오랫동안 사용되지 않음(LRU) 목록을 유지합니다. LRU 목록은 항목이 캐시에 처음 기록된 시점 또는 캐시에서 마지막으로 읽은 시점을 추적합니다. DAX 노드 메모리가 가득 차면 DAX는 아직 만료되지 않았더라도 오래된 항목을 제거하여 새 항목을 위한 공간을 마련합니다. LRU 알고리즘은 항상 활성화되어 있으며 사용자가 구성할 수 없습니다.

애플리케이션에 적합한 TTL 기간을 설정하려면 다음 사항을 고려하세요.

데이터 액세스 패턴 파악

- 읽기 중심의 워크로드 - 읽기 워크로드가 많고 데이터 업데이트 빈도가 낮은 애플리케이션의 경우 TTL 기간을 더 길게 설정하여 캐시 누락 횟수를 줄이세요. TTL 기간이 길면 기본 DynamoDB 테이블에 액세스할 필요성도 줄어듭니다.
- 쓰기 중심의 워크로드 - DAX를 통해 작성되지 않고 업데이트가 잦은 애플리케이션의 경우 캐시가 데이터베이스와 일관성을 유지할 수 있도록 TTL 기간을 짧게 설정하세요. TTL 기간이 짧을수록 오래된 데이터를 제공할 위험도 줄어듭니다.

애플리케이션의 성능 요구 사항 평가

- 지연 시간 민감도 - 애플리케이션에 데이터 새로 고침보다 짧은 지연 시간이 필요한 경우 TTL 기간을 더 길게 사용하세요. TTL 기간이 길수록 캐시 적중률이 극대화되어 평균 읽기 지연 시간이 줄어듭니다.
- 처리량 및 확장성 - TTL 기간이 길수록 DynamoDB 테이블의 부하가 줄어들고 처리량과 확장성이 향상됩니다. 그러나 최신 데이터에 대한 필요성과 균형을 맞춰야 합니다.

캐시 제거 및 메모리 사용량 분석

- 캐시 메모리 제한 - DAX 클러스터의 메모리 사용량을 모니터링합니다. TTL 기간이 길어지면 캐시에 더 많은 데이터를 저장할 수 있으며, 이로 인해 메모리 한도에 도달하여 LRU 기반 제거로 이어질 수 있습니다.

지표와 모니터링을 사용하여 TTL 조정

캐시 적중률 및 누락, CPU 및 메모리 사용률과 같은 지표를 정기적으로 검토하세요. 이러한 지표를 기반으로 TTL 설정을 조정하여 성능과 데이터 최신성 간의 최적의 균형을 찾을 수 있습니다. 캐시 누락이 많고 메모리 사용률이 낮은 경우 TTL 기간을 늘려 캐시 적중률을 높이세요.

비즈니스 요구 사항 및 규정 준수 고려

데이터 보존 정책에 따라 민감한 정보 또는 개인 정보에 설정할 수 있는 최대 TTL 기간이 정해질 수 있습니다.

TTL을 0으로 설정한 경우의 캐시 동작

TTL을 0으로 설정하면 항목 캐시와 쿼리 캐시는 다음과 같은 동작을 나타냅니다.

- 항목 캐시 - LRU 제거 또는 라이트-스루 작업이 발생한 경우에만 캐시의 항목이 새로 고쳐집니다.
- 쿼리 캐시 - 쿼리 응답은 캐시되지 않습니다.

## DAX 클러스터로 여러 테이블 캐싱

개별 캐시가 필요하지 않은 작은 DynamoDB 테이블이 여러 개 있는 워크로드의 경우 단일 DAX 클러스터가 이러한 테이블에 대한 요청을 캐싱합니다. 따라서 특히 여러 테이블에 액세스하고 고성능 읽기가 필요한 애플리케이션에 DAX를 보다 유연하고 효율적으로 사용할 수 있습니다.

DynamoDB 데이터 플레인 API와 마찬가지로 DAX 요청에는 테이블 이름이 필요합니다. 동일한 DAX 클러스터에서 여러 테이블을 사용하는 경우 구체적인 구성이 필요하지 않습니다. 그러나 클러스터의 보안 권한이 모든 캐시된 테이블에 대한 액세스를 허용하는지 확인해야 합니다.

여러 테이블과 함께 DAX를 사용하기 위한 고려 사항

DAX를 여러 DynamoDB 테이블과 함께 사용할 때 다음 사항을 고려해야 합니다.

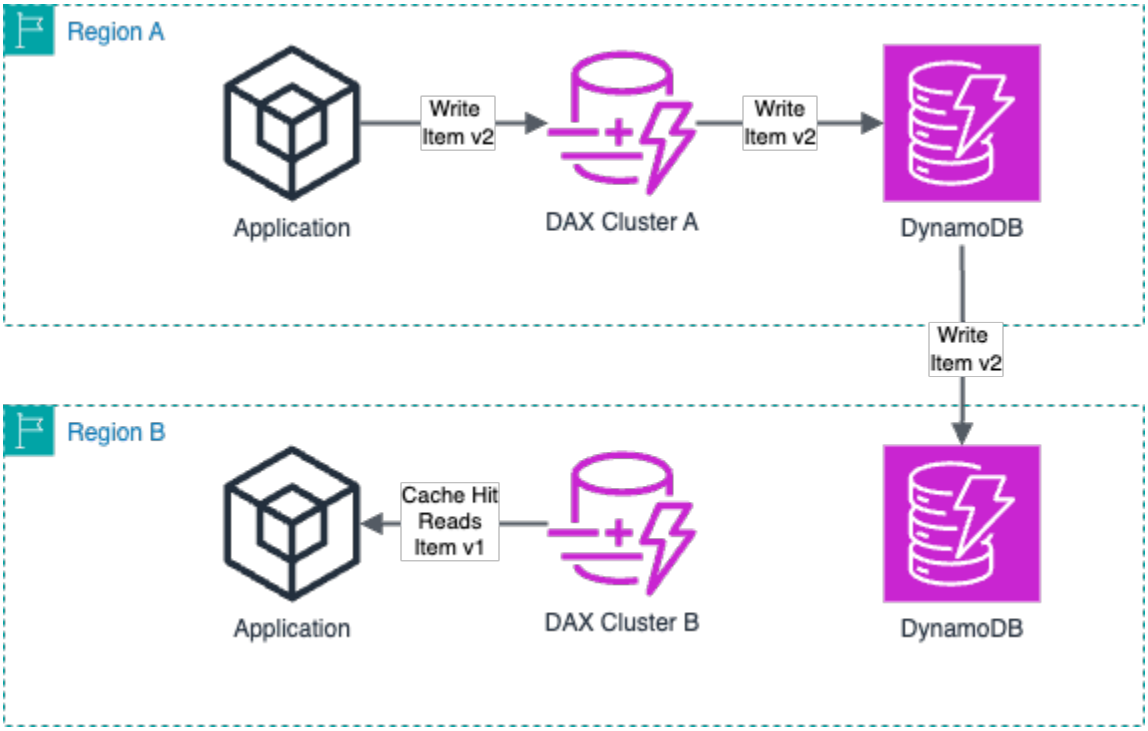
- 메모리 관리 - 여러 테이블과 함께 DAX를 사용할 때는 작업 데이터세트의 전체 크기를 고려해야 합니다. 데이터세트의 모든 테이블은 선택한 노드 유형의 동일한 메모리 공간을 공유합니다.
- 리소스 할당 - DAX 클러스터의 리소스는 캐시된 모든 테이블에서 공유됩니다. 하지만 트래픽이 많은 테이블에서는 인접한 더 작은 테이블에서 데이터가 제거될 수 있습니다.
- 규모의 경제 - 더 작은 리소스를 더 큰 DAX 클러스터로 그룹화하여 트래픽을 더 안정적인 패턴으로 평균화합니다. DAX 클러스터에 필요한 총 읽기 리소스 수를 고려하면 3개 이상의 노드를 보유하는 것도 경제적입니다. 또한, 클러스터에 있는 모든 캐시된 테이블의 가용성도 향상됩니다.

## DAX 및 DynamoDB 글로벌 테이블의 데이터 복제

DAX는 리전 기반 서비스이므로, 클러스터는 AWS 리전 내의 트래픽만 인식합니다. 글로벌 테이블은 다른 리전의 데이터를 복제할 때 캐시를 중심으로 작성합니다.

TTL 기간이 길면 오래된 데이터가 기본 리전보다 보조 리전에 더 오래 남아 있을 수 있습니다. 이로 인해 보조 리전의 로컬 캐시에서 캐시 누락이 발생할 수 있습니다.

다음 다이어그램은 소스 리전 A의 글로벌 테이블 수준에서 발생하는 데이터 복제를 보여줍니다. 리전 B의 DAX 클러스터는 소스 리전 A에서 새로 복제된 데이터를 즉시 인식하지 못합니다.




## DAX 리전 가용성

DynamoDB 테이블을 지원하는 모든 리전이 DAX 클러스터 배포를 지원하지는 않습니다. 애플리케이션에 DAX를 통한 짧은 읽기 지연 시간이 필요한 경우 먼저 DAX를 지원하는 리전 목록을 검토하세요. 그런 다음 DynamoDB 테이블의 리전을 선택합니다.

## DAX 캐싱 동작

DAX는 메타데이터와 음성 캐싱을 수행합니다. 이러한 캐싱 동작을 이해하면 캐시된 항목 및 음성 캐시 항목의 속성 메타데이터를 효과적으로 관리하는 데 도움이 됩니다.

- 메타데이터 캐싱 - DAX 클러스터는 캐시한 항목의 속성 이름에 대한 메타데이터를 무기한 유지 관리합니다. 이 메타데이터는 항목이 만료되거나 캐시에서 제거된 후에도 유지됩니다.

시간이 지남에 따라 무한한 수의 속성 이름을 사용하는 애플리케이션은 DAX 클러스터에서 메모리 부족을 야기할 수 있습니다. 이러한 제한은 최상위 속성 이름에만 적용되며, 중첩 속성 이름에는 적용되지 않습니다. 무제한 속성 이름의 예로는 타임스탬프, UUID 및 세션 ID가 포함됩니다. 타임스탬프와 세션 ID를 속성 값으로 사용할 수 있지만, 더 짧고 예측 가능한 속성 이름을 사용하는 것이 좋습니다.

- 음성 캐싱 - 캐시 누락이 발생하고 DynamoDB 테이블에서 읽은 결과 일치하는 항목이 없는 경우 DAX는 각 항목 또는 쿼리 캐시에 음성 캐시 항목을 추가합니다. 이 항목은 캐시 TTL 기간이 만료되거나 라이트-스루 작업이 발생할 때까지 유지됩니다. DAX는 향후 요청을 위해 이 음성 캐시 항목을 계속 반환합니다.

  음성 캐싱 동작이 애플리케이션 패턴에 맞지 않는 경우 DAX가 빈 결과를 반환할 때 DynamoDB 테이블을 직접 읽어보세요. 또한, 캐시에서 빈 결과가 오래 지속되지 않도록 하고 테이블과의 일관성을 높이려면 TTL 캐시 기간을 짧게 설정하는 것이 좋습니다.

## DAX 클러스터 크기 조정

DAX 클러스터의 총 용량 및 가용성은 노드 유형과 수에 따라 달라집니다. 클러스터에 노드가 많을수록 읽기 용량은 증가하지만, 쓰기 용량은 증가하지 않습니다. 노드 유형이 클수록(최대 r5.8xlarge) 더 많은 쓰기를 처리할 수 있으나, 노드가 너무 적으면 노드 장애 발생 시 가용성에 영향을 미칠 수 있습니다. DAX 클러스터 크기 조정에 대한 자세한 내용은 DAX 클러스터 크기 조정 안내서 섹션을 참조하세요.

다음 섹션에서는 확장 가능하고 비용 효율적인 클러스터를 만들기 위해 노드 유형과 수의 균형을 맞추려면 고려해야 하는 다양한 크기 조정 측면을 설명합니다.

이 섹션의 내용

- 가용성 계획
- 쓰기 처리량 계획
- 읽기 처리량 계획
- 데이터세트 크기 계획
- 대략적인 클러스터 용량 요구 사항 계산
- 노드 유형별 클러스터 처리량 용량 근사치 계산
- DAX 클러스터의 쓰기 용량 규모 조정

## 가용성 계획

DAX 클러스터의 크기를 조정할 때는 먼저 대상 가용성에 초점을 맞춰야 합니다. DAX와 같은 클러스터링된 서비스의 가용성은 클러스터에 있는 총 노드 수의 차원입니다. 단일 노드 클러스터는 장애를 용납하지 않으므로, 가용성은 노드 1개와 같습니다. 10노드 클러스터에서는 단일 노드 손실이 클러스터의 전체 용량에 미치는 영향이 미미합니다. 나머지 노드는 여전히 읽기 요청을 처리할 수 있기 때문에 손실이 생겨도 가용성에 직접적인 영향을 미치지 않습니다. DAX는 쓰기를 재개하기 위해 새 프라이머리 노드를 신속하게 지명합니다.

DAX는 VPC 기반입니다. 서브넷 그룹을 사용하여 노드를 실행할 수 있는 가용 영역과 서브넷에서 사용할 IP 주소를 결정합니다. 프로덕션 워크로드의 경우 서로 다른 가용 영역에 있는 최소 3개 노드로 구성된 DAX를 사용할 것을 적극 권장합니다. 이렇게 하면 단일 노드 또는 가용 영역에 장애가 발생하더라도 클러스터에 요청을 처리할 노드가 2개 이상 남아 있게 됩니다. 클러스터는 최대 11개의 노드를 가질 수 있으며, 여기서 하나는 프라이머리 노드이고 10개는 읽기 전용 복제본입니다.

## 쓰기 처리량 계획

모든 DAX 클러스터에는 라이트-스루 요청을 위한 프라이머리 노드가 있습니다. 클러스터의 노드 유형 크기에 따라 쓰기 용량이 결정됩니다. 읽기 전용 복제본을 추가해도 클러스터의 쓰기 용량은 증가하지 않습니다. 따라서 나중에 노드 유형을 변경할 수 없으므로, 클러스터를 생성할 때 쓰기 용량을 고려해야 합니다.

애플리케이션에서 항목 캐시를 업데이트하려고 DAX를 통해 작성해야 하는 경우 쓰기가 용이하도록 클러스터 리소스 사용을 늘리는 것이 좋습니다. DAX에 대한 쓰기는 캐시 적중 읽기보다 약 25배 더 많은 리소스를 소비합니다. 이 경우 읽기 전용 클러스터보다 더 큰 노드 유형이 필요할 수 있습니다.

라이트-스루 또는 라이트-어라운드가 애플리케이션에 가장 적합한지 여부를 결정하는 방법에 대한 추가 지침은 쓰기 전략 섹션을 참조하세요.

## 읽기 처리량 계획

DAX 클러스터의 읽기 용량은 워크로드의 캐시 적중률에 따라 달라집니다. 캐시 누락이 발생할 경우 DAX는 DynamoDB에서 데이터를 읽기 때문에 캐시 적중률보다 약 10배 더 많은 클러스터 리소스를 소비합니다. 캐시 적중률을 높이려면 캐시의 TTL 설정을 높여 항목이 캐시에 저장되는 기간을 정의하세요. 하지만 TTL 기간이 길수록 DAX를 통해 업데이트를 작성하지 않는 한 이전 항목 버전을 읽을 가능성이 높아집니다.

클러스터에 충분한 읽기 용량이 있는지 확인하려면 클러스터의 수평적 스케일링에서 설명한 대로 클러스터를 수평으로 스케일링하세요. 노드를 더 추가하면 리소스 풀에 읽기 전용 복제본이 추가되고, 노

드를 제거하면 읽기 용량이 줄어듭니다. 클러스터의 노드 수와 크기를 선택할 때는 필요한 최소 읽기 용량과 최대 읽기 용량을 모두 고려해야 합니다. 읽기 요구 사항에 맞게 더 작은 노드 유형으로 클러스터를 수평적으로 스케일링할 수 없는 경우 더 큰 노드 유형을 사용하세요.

### 데이터세트 크기 계획

사용 가능한 각 노드 유형에는 DAX가 데이터를 캐시할 수 있도록 설정된 메모리 크기가 있습니다. 노드 유형이 너무 작으면 애플리케이션이 요청하는 작업 데이터세트가 메모리에 맞지 않아 캐시 누락이 발생합니다. 큰 노드는 더 큰 캐시를 지원하므로, 캐싱해야 하는 예상 데이터세트보다 큰 노드 유형을 사용하세요. 캐시가 클수록 적중률도 향상됩니다.

반복 읽기가 거의 없는 항목을 캐싱하면 수익이 줄어들 수 있습니다. 자주 액세스하는 항목의 메모리 크기를 계산하고 캐시가 해당 데이터세트를 저장할 수 있을 만큼 충분히 큰지 확인하세요.

### 대략적인 클러스터 용량 요구 사항 계산

워크로드의 총 용량 요구량을 추정하여 클러스터 노드의 적절한 크기와 수량을 선택할 수 있습니다. 추정하려면 가변 정규화된 초당 요청 수(정규화된 RPS)를 계산하세요. 이 변수는 캐시 적중, 캐시 누락, 쓰기를 포함하여 애플리케이션이 DAX 클러스터를 지원하는 데 필요한 총 작업 단위를 나타냅니다. 정규화된 RPS를 계산하려면 다음 입력을 사용합니다.

- `ReadRPS_CacheHit` - 캐시 적중으로 이어지는 초당 읽기 수를 지정합니다.
- `ReadRPS_CacheMiss` - 캐시 누락으로 이어지는 초당 읽기 수를 지정합니다.
- `WriteRPS` - DAX를 통과할 초당 쓰기 수를 지정합니다.
- `DaxNodeCount` – DAX 클러스터에 있는 노드 수를 지정합니다.
- `Size` - 쓰거나 읽는 항목의 크기를 KB 단위로 가장 가까운 KB로 반올림하여 지정합니다.
- `10x_ReadMissFactor` - 값 10을 나타냅니다. 캐시 누락이 발생할 경우 DAX는 캐시 적중률보다 약 10배 더 많은 리소스를 사용합니다.
- `25x_WriteFactor` - DAX 라이트-스루는 캐시 적중률보다 약 25배 많은 리소스를 사용하기 때문에 값 25를 나타냅니다.

다음 공식을 사용하여 정규화된 RPS를 계산할 수 있습니다.

```
Normalized RPS = (ReadRPS_CacheHit * Size) + (ReadRPS_CacheMiss * Size *
 10x_ReadMissFactor) + (WriteRequestRate * 25x_WriteFactor * Size * DaxNodeCount)
```

예를 들어, 다음과 같은 입력 값을 고려합니다.

- `ReadRPS_CacheHit` = 50,000
- `ReadRPS_CacheMiss` = 1,000
- `ReadMissFactor` = 1
- `Size` = 2KB
- `WriteRPS` = 100
- `WriteFactor` = 1
- `DaxNodeCount` = 3

공식에서 이러한 값을 대체하면 다음과 같이 정규화된 RPS를 계산할 수 있습니다.

```
Normalized RPS = (50,000 Cache Hits/Sec * 2KB) + (1,000 Cache Misses/Sec * 2KB * 10) +
 (100 Writes/Sec * 25 * 2KB * 3)
```

이 예제에서 정규화된 RPS의 계산된 값은 135,000입니다. 하지만 이 정규화된 RPS 값은 클러스터 사용률을 100% 미만으로 유지하거나 노드 손실을 고려하지 않습니다. 추가 용량을 고려하는 것이 좋습니다. 이를 위해서는 목표 사용률과 노드 손실 허용 한도라는 2가지 배수 요소 중 더 큰 값을 결정해야 합니다. 그런 다음 정규화된 RPS에 더 큰 인수를 곱하여 초당 목표 요청 수(목표 RPS)를 구하세요.

- 목표 사용률

  성능에 영향을 미치면 캐시 누락이 증가하므로, DAX 클러스터를 100% 사용률로 실행하는 것은 권장하지 않습니다. 클러스터 사용률을 70% 이하로 유지하는 것이 가장 좋습니다. 이를 달성하려면 정규화된 RPS에 1.43을 곱하세요.
- 노드 손실 허용 한도

  노드에 장애가 발생하는 경우 애플리케이션은 해당 요청을 나머지 노드에 분산할 수 있어야 합니다. 노드 사용률을 100% 미만으로 유지하려면 장애가 발생한 노드가 다시 온라인 상태가 될 때까지 추가 트래픽을 흡수할 수 있을 만큼 큰 노드 유형을 선택하세요. 노드 수가 적은 클러스터의 경우 한 노드에 장애가 발생할 때 각 노드는 더 큰 트래픽 증가를 허용해야 합니다.

  프라이머리 노드에 장애가 발생하면 DAX가 자동으로 읽기 전용 복제본으로 장애 조치하고 이를 새로운 프라이머리 노드로 지정합니다. 복제본 노드에 장애가 발생하면 실패한 노드가 복구될 때까지 DAX 클러스터의 다른 노드가 계속 요청을 수행할 수 있습니다.

  예를 들어, 노드 장애가 발생한 3노드 DAX 클러스터의 경우 나머지 두 노드에 50%의 추가 용량이 필요합니다. 이를 위해서는 1.5를 곱해야 합니다. 반대로, 장애가 발생한 노드가 있는 11노드 클러스터의 경우 나머지 노드에 10%의 추가 용량이 필요하거나 배율 1.1을 곱해야 합니다.

다음 공식을 사용하여 목표 RPS를 계산할 수 있습니다.

```
Target RPS = Normalized RPS * CEILING(TargetUtilization, NodeLossTolerance)
```

예를 들어, 목표 RPS를 계산하려면 다음 값을 고려하세요.

- `Normalized RPS` = 135,000
- `TargetUtilization` = 1.43

  최대 70%의 클러스터 사용률을 목표로 하고 있기 때문에 `TargetUtilization`을 1.43으로 설정했습니다.
- `NodeLossTolerance` = 1.5

  3노드 클러스터를 사용하고 있으므로, `NodeLossTolerance`를 50% 용량으로 설정한다고 가정해 보겠습니다.

공식에서 이러한 값을 대체하면 다음과 같이 목표 RPS를 계산할 수 있습니다.

```
Target RPS = 135,000 * CEILING(1.43, 1.5)
```

이 예제에서는 `NodeLossTolerance`의 값이 `TargetUtilization`보다 크기 때문에 `NodeLossTolerance`를 사용하여 목표 RPS 값을 계산합니다. 이를 통해 DAX 클러스터가 지원해야 하는 총 용량인 202,500의 목표 RPS 값을 얻을 수 있습니다. 클러스터에 필요한 노드 수를 결정하려면 목표 RPS를 다음 테이블의 해당 열에 매핑하세요. 목표 RPS가 202,500인 이 예제에서는 3개의 노드가 있는 dax.r5.large 노드 유형이 필요합니다.

## 노드 유형별 클러스터 처리량 용량 근사치 계산

Target RPS formula을 사용하여 다양한 노드 유형의 클러스터 용량을 추정할 수 있습니다. 다음 테이블에는 노드 유형이 1, 3, 5, 11인 클러스터의 대략적인 용량이 나와 있습니다. 이러한 용량이 있다고 해서 자체 데이터 및 요청 패턴으로 DAX의 부하 테스트를 수행할 필요가 없어지지는 않습니다. 또한, 고정 CPU 용량이 부족하기 때문에 이러한 용량에는 t 유형 인스턴스가 포함되지 않습니다. 다음 테이블의 모든 값 단위는 정규화된 RPS입니다.

| 노드 유형(메모리) | 노드 1 | 노드 3 | 노드 5 | 노드 11 |
|---|---|---|---|---|
| dax.r5.24xlarge(768GB) | 100만 | 300만 | 500만 | 11M |
| dax.r5.16xlarge(512GB) | 100만 | 300만 | 500만 | 1,100만 |
| dax.r5.12xlarge(384GB) | 100만 | 300만 | 500만 | 1,100만 |
| dax.r5.8xlarge(256GB) | 100만 | 300만 | 500만 | 1,100만 |
| dax.r5.4xlarge(128GB) | 60만 | 180만 | 300만 | 660만 |
| dax.r5.2xlarge(64GB) | 30만 | 90만 | 150만 | 330만 |
| dax.r5.xlarge(32GB) | 15만 | 45만 | 75만 | 165만 |
| dax.r5.large(16GB) | 7만5천 | 22만5천 | 37만5천 | 82만5천 |

각 노드의 최대 한도는 1백만 초당 네트워크 작업 수(NPS)이므로, dax.r5.8xlarge 이상의 노드 유형은 추가 클러스터 용량을 제공하지 않습니다. 8xlarge보다 큰 노드 유형은 클러스터의 총 처리량 용량에 기여하지 않을 수 있습니다. 그러나 이러한 노드 유형은 더 큰 작업 데이터세트를 메모리에 저장하는 데 유용할 수 있습니다.

## DAX 클러스터의 쓰기 용량 규모 조정

DAX에 대한 각 쓰기는 모든 노드에서 25개의 정규화된 요청을 소비합니다. 각 노드에 대해 1백만 RPS 한도가 있기 때문에, DAX 클러스터는 읽기 사용량을 고려하지 않고 초당 4만개의 쓰기로 제한됩니다.

사용 사례에서 캐시에 초당 4만회 이상의 쓰기가 필요한 경우 별도의 DAX 클러스터를 사용하고 이들 간에 쓰기를 샤드해야 합니다. DynamoDB와 마찬가지로 캐시에 쓰는 데이터의 파티션 키를 해시할 수 있습니다. 그런 다음 모듈러스를 사용하여 데이터를 쓸 샤드를 결정합니다.

다음 예제에서는 입력 문자열의 해시를 계산합니다. 그런 다음 해시 값의 모듈러스를 10으로 계산합니다.

```
def hash_modulo(input_string):
    # Compute the hash of the input string
    hash_value = hash(input_string)

    # Compute the modulus of the hash value with 10
    bucket_number = hash_value % 10

    return bucket_number

#Example usage
if _name_ == "_main_":
    input_string = input("Enter a string: ")
    result = hash_modulo(input_string)
    print(f"The hash modulo 10 of '{input_string}' is: {result}.")
```

## 클러스터 배포

새 DAX 클러스터를 생성하려면 DynamoDB에 필요한 구성 이상의 구성이 필요합니다. DAX가 Amazon VPC를 기반으로 하기 때문에 이러한 구성은 특히 네트워킹에 적합합니다. 이렇게 하면 리소스 배치, 연결 및 보안을 포함하여 가상 네트워킹 환경을 완전히 제어할 수 있습니다. 이 섹션에서는 클러스터 생성 시 필요한 설정에 대한 모범 사례를 제공합니다.

클러스터 노드 선택에 대한 자세한 내용은 DAX 클러스터 크기 조정 섹션을 참조하세요.

이 섹션의 내용

- 네트워크 구성
- 보안 구성
- Parameter Group
- 유지보수 윈도우

## 네트워크 구성

DAX는 서브넷 그룹을 사용하여 노드를 실행할 수 있는 가용 영역과 서브넷에서 사용할 IP 주소를 결정합니다. 애플리케이션과 DAX 사이의 지연 시간을 최소화하려면 애플리케이션 서버와 DAX 클러스터의 서브넷 및 가용 영역이 동일해야 합니다.

DAX 노드를 여러 가용 영역에 걸쳐 분산하는 것이 좋습니다. 기본 옵션인 자동 할당이 이 작업을 자동으로 수행합니다.

VPC 설정에 대한 모범 사례는 Amazon VPC 사용 설명서의 Amazon VPC 시작하기를 참조하세요.

## 보안 구성

이 섹션에서는 DAX를 사용하는 애플리케이션에 구현해야 하는 보안 조치에 대해 설명합니다. 또한, DAX가 데이터 암호화를 위해 포함하는 지원에 대해 간략하게 설명합니다.

### IAM

DAX 및 DynamoDB의 액세스 제어 메커니즘은 별개입니다. DAX에서 DynamoDB 테이블에 액세스하려면 IAM 역할이 필요합니다. 이 역할은 최소 권한 원칙을 따르고 GetItem 및 PutItem과 같은 특정 테이블 및 DynamoDB 작업에만 액세스 권한을 부여해야 합니다. DAX에서 제공하는 액세스 제어 메커니즘에 대한 자세한 내용은 DAX 액세스 제어 섹션을 참조하세요.

### 암호화

DAX 클러스터를 생성할 때 저장 중 암호화와 전송 중 암호화를 구성합니다. 이는 기본적으로 활성화되어 있습니다. 비즈니스 요구 사항으로 인해 제한되지 않는 한 기본 암호화 설정을 유지하는 것이 좋습니다. 자세한 내용은 DAX 저장 데이터 암호화 및 DAX 전송 중 데이터 암호화 섹션을 참조하세요.

## Parameter Group

DAX는 파라미터 그룹이라고 하는 클러스터의 모든 노드에 구성 집합을 적용합니다. 클러스터를 생성한 후 이 구성을 변경할 수 있습니다.

DAX 파라미터 그룹에는 항목 캐시 및 쿼리 캐시에 대한 TTL 설정이 들어 있습니다. 기본 TTL 지속 시간은 5분입니다. TTL 지속 시간을 1밀리초 이상의 정수 값으로 재정의할 수 있습니다.

실행 중인 DAX 인스턴스에서 사용 중인 파라미터 그룹은 수정할 수 없습니다. DAX 클러스터의 가동 중지 도중 파라미터 그룹 값을 변경할 수 있습니다.

### 유지보수 윈도우

가끔씩 소프트웨어를 업그레이드하고 노드에 패치를 적용할 수 있도록 DAX 클러스터에 주간 유지 관리 기간이 구성되어 있습니다. 이 기간 동안 DAX는 노드에 대한 롤링 업데이트를 수행합니다. 노드가 2개 이상인 클러스터는 업데이트 중에 클러스터의 가용성이 유지되지만, 노드가 복구될 때까지 클러스터 용량이 감소합니다. 조직의 사용량이 적을 것으로 예상되는 기간이 있는 경우 유지 관리 기간을 이때로 수동 설정하는 것을 고려해 보세요.

## 클러스터 작업 관리

DAX가 클러스터 유지 관리 및 상태를 자동으로 처리합니다. 하지만 사용 패턴에 맞게 클러스터를 수평 또는 수직으로 스케일링하려면 작업 입력을 제공해야 합니다. 이 섹션에서는 DAX 클러스터 규모를 조정하는 데 필요한 권장 프로세스에 대해 설명합니다.

이 섹션의 내용

- 클러스터의 수평적 스케일링
- 클러스터의 수직적 스케일링

### 클러스터의 수평적 스케일링

DAX 클러스터 규모를 조정하려면 처리량 수요를 충족하도록 용량을 조정해야 합니다. 클러스터가 실행되는 동안 클러스터의 노드(복제본) 수를 늘리거나 줄이면서 조정할 수 있습니다. 수평적 스케일링이라고 하는 이 프로세스를 적용하면 더 많은 노드에 워크로드를 분산하거나 수요가 적을 때 더 적은 수의 노드로 통합할 수 있습니다.

AWS CLI에서 `decrease-replication-factor` 또는 `increase-replication-factor` 명령을 사용하여 DAX 클러스터를 수평적으로 스케일 인 및 스케일 아웃할 수 있습니다.

복제 인수 증가(스케일 아웃)

DAX 클러스터의 복제 인수를 늘리면 클러스터에 더 많은 노드가 추가됩니다. 다음 예제에서는 `increase-replication-factor` 명령의 사용법을 보여줍니다.

```
aws dax increase-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- 이 명령에서 `cluster-name` 인수는 클러스터의 이름을 지정합니다. *yourClusterName*을 예로 들 수 있습니다.

- new-replication-factor 인수는 규모 조정 후 클러스터에 추가할 총 노드 수를 지정합니다. 여기에는 프라이머리 노드와 복제본 노드가 포함됩니다. 예를 들어, 클러스터에 현재 3개의 노드가 있는데 노드를 2개 더 추가하려는 경우 new-replication-factor 값을 5로 설정합니다.

### 복제 인수 감소(스케일 인)

DAX 클러스터의 복제 인수를 줄이면 클러스터에서 노드가 제거됩니다. 노드를 제거하면 수요가 적은 기간 동안 비용을 절감하는 데 도움이 될 수 있습니다. 다음 예제에서는 decrease-replication-factor 명령의 사용법을 보여줍니다.

```
aws dax decrease-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- 이 명령에서 cluster-name 인수는 클러스터의 이름을 지정합니다. *yourClusterName*을 예로 들 수 있습니다.
- new-replication-factor 인수는 규모 조정 후 클러스터의 줄어든 노드 수를 지정합니다. 이 수는 현재 복제 인수보다 낮아야 하며 프라이머리 노드를 포함해야 합니다. 예를 들어, 클러스터에 5개의 노드가 있고 2개의 노드를 제거하려는 경우 new-replication-factor 값을 3으로 설정합니다.

### 수평적 스케일링 고려 사항

수평적 스케일링 계획을 세울 때 다음 사항을 고려하세요.

- 프라이머리 노드 - DAX 클러스터에는 프라이머리 노드가 포함되어 있습니다. 복제 인수에는 이 프라이머리 노드가 포함됩니다. 예를 들어, 복제 인수 3은 프라이머리 노드 하나와 복제 노드 2개를 의미합니다.
- 가용성 - DAX 노드를 추가하거나 제거하면 클러스터의 가용성과 내결함성이 변경됩니다. 노드가 많을수록 가용성이 향상될 수 있지만, 비용도 증가합니다.
- 데이터 마이그레이션 - 복제 인수를 늘리면 DAX가 새 노드 집합 전반의 데이터 분산을 자동으로 처리합니다. 새 노드가 트래픽을 처리하기 시작하면 해당 캐시는 이미 워밍업된 상태입니다. 하지만 이 프로세스 중에는 데이터 마이그레이션 도중 성능에 일시적인 영향이 있을 수 있습니다.

규모 조정 프로세스 도중과 이후에 DAX 클러스터를 면밀히 모니터링하여 예상대로 작동하는지 확인하고 필요에 따라 추가로 조정해야 합니다.

## 클러스터의 수직적 스케일링

기존 클러스터의 노드 크기를 수직적으로 스케일링하려면 새 클러스터를 생성하고 애플리케이션 트래픽을 새 클러스터로 마이그레이션해야 합니다. 노드가 다른 새 클러스터로 마이그레이션하려면 애플리케이션의 성능 및 가용성에 미치는 영향을 최소화하면서 원활하게 전환할 수 있도록 여러 단계를 거쳐야 합니다.

노드 크기를 수직적으로 스케일링할 수 있는 새 클러스터를 만들려면 다음 사항을 고려하세요.

- 현재 설정에 액세스 - 현재 DAX 클러스터의 지표를 검토하여 필요한 새 노드 크기와 수량을 결정하세요. 이 정보를 입력으로 사용하여 클러스터 크기를 정의합니다. 자세한 내용은 DAX 클러스터 크기 조정 섹션을 참조하세요.
- 새 DAX 클러스터 설정 - 결정한 노드 유형과 수량으로 새 DAX 클러스터를 생성합니다. 조정할 필요가 없는 한 파라미터 그룹의 기존 구성 설정을 사용할 수 있습니다.
- 데이터 동기화 - DAX는 DynamoDB의 캐싱 계층이므로, 데이터를 직접 마이그레이션할 필요가 없습니다. 그러나 새 DAX 클러스터는 트래픽을 전송하기 전까지는 작업 데이터세트를 메모리에 저장하지 않습니다.
- 애플리케이션 구성 업데이트 - 새 DAX 클러스터의 엔드포인트를 가리키도록 애플리케이션 구성을 업데이트하세요. 애플리케이션 구성에 따라 코드를 변경하거나 환경 변수를 업데이트해야 할 수 있습니다.

  새 클러스터로 전환할 때 미치는 영향을 줄이려면 애플리케이션 플릿의 일부에서 새 클러스터로 canary 트래픽을 전송하세요. 애플리케이션 업데이트를 천천히 배포하거나 DAX 엔드포인트 앞에 가중치 기반 라우팅 DNS 항목을 사용하여 이를 수행할 수 있습니다.
- 모니터링 및 최적화 - 새 DAX 클러스터로 전환한 후 성능 지표와 로그를 면밀히 모니터링하여 문제가 있는지 확인하세요. 업데이트된 워크로드 패턴에 따라 노드 수를 조정할 준비를 하세요.

  새 클러스터가 작업 데이터세트를 제대로 캐시하기 전까지는 캐시 누락률과 지연 시간이 더 높아집니다.
- 기존 클러스터 사용 중지 - 새 클러스터가 예상대로 작동한다고 확신할 경우 이전 DAX 클러스터를 안전하게 해제하여 불필요한 비용을 방지하세요.

# DAX 모니터링

캐시 적중률과 같은 주요 지표를 모니터링하여 최적의 DAX 클러스터 성능을 보장하고, 문제를 진단하고, 클러스터 규모를 조정해야 하는 시기를 결정할 수 있습니다. 주요 지표를 정기적으로 확인하면 워

크로드 요구 사항에 맞게 클러스터 규모를 조정하여 성능, 안정성 및 비용 효율성을 유지하는 데 도움이 됩니다. DAX 모니터링에 대한 자세한 내용은 프로덕션 모니터링 섹션을 참조하세요.

다음 목록은 모니터링해야 하는 몇 가지 주요 지표를 보여줍니다.

- 캐시 적중률 - DAX가 캐시된 데이터를 얼마나 효과적으로 처리하는지 보여 주므로, 기본 DynamoDB 테이블에 액세스할 필요성이 줄어듭니다. 클러스터의 캐시 누락이 거의 없다는 것은 캐싱 효율성이 양호하다는 사실을 나타냅니다. 하지만 캐시 적중률이 줄면 캐싱 TTL 설정을 다시 검토해야 할 수도 있고, 워크로드가 캐싱에 적합하지 않을 수도 있음을 시사합니다.

  Amazon CloudWatch를 사용하여 DAX 클러스터의 캐시 적중률을 계산할 수 있습니다. `ItemCacheHits`, `ItemCacheMisses`, `QueryCacheHits`, `QueryCacheMisses` 지표를 비교하여 이 비율을 구하세요. 다음 공식은 캐시 적중률 계산 방법을 보여줍니다. 이 공식을 사용하여 비율을 계산하려면 캐시 적중률을 캐시 적중률과 누락의 합계로 나누세요.

  ```
  Cache hit ratio = Cache hits / (Cache hits + Cache misses)
  ```

  캐시 적중률은 0에서 1 사이의 숫자이며 백분율로 표시됩니다. 백분율이 높을수록 전체 캐시 사용률이 높아집니다.

- ErrorRequestCount - 노드 또는 클러스터에서 보고한 사용자 오류로 이어진 요청 수입니다. `ErrorRequestCount`에는 노드 또는 클러스터에서 제한이 발생한 요청이 포함됩니다. 사용자 오류를 모니터링하면 애플리케이션의 규모 조정 구성 오류 또는 핫 항목/파티션 패턴을 식별하는 데 도움이 될 수 있습니다.
- 작동 지연 - DAX 클러스터와의 읽기 및 쓰기 작업 지연 시간을 모니터링하면 성능 병목 현상을 식별하는 데 도움이 될 수 있습니다. 지연 시간이 증가하면 DAX 클러스터 구성, 네트워크 또는 규모 조정 필요성에 문제가 있는 것일 수 있습니다.
- 네트워크 소비 - `NetworkBytesIn` 및 `NetworkBytesOut` 지표를 주시하여 DAX 클러스터의 네트워크 트래픽을 모니터링하세요. 네트워크 처리량이 예기치 않게 증가하면 클라이언트 요청이 늘어나거나 쿼리 패턴이 비효율적이라는 의미여서 더 많은 데이터가 전송될 수 있습니다.

  네트워크 소비를 모니터링하면 DAX 클러스터의 비용을 관리하는 데 도움이 됩니다. 또한, 네트워크가 클러스터 성능의 병목 현상으로 작용하지 않도록 합니다.
- 제거율 - 새 항목을 위한 공간을 확보하려고 캐시에서 항목을 제거하는 빈도를 보여줍니다. 시간이 지나면서 제거율이 증가하면 캐시가 너무 작거나 캐싱 전략이 효과적이지 않게 될 수 있습니다.

CloudWatch의 `EvictedSize` 지표를 모니터링하여 캐시 크기가 워크로드에 적합한지 확인하세요. 제거된 총 크기가 계속 증가하면 더 큰 캐시를 수용할 수 있도록 DAX 클러스터를 스케일 업해야 할 수 있습니다.

- CPU 사용률 – 노드 또는 클러스터의 CPU 사용률 백분율입니다. 이는 모든 데이터베이스 또는 캐싱 시스템에서 모니터링해야 하는 중요한 지표입니다. CPU 사용률이 높으면 DAX 클러스터에 과부하가 걸리고 늘어난 수요를 처리하기 위해 규모를 조정해야 할 수 있습니다.

  DAX 클러스터에 대한 `CPUUtilization` 지표를 모니터링하세요. CPU 사용률이 지속적으로 70~80%에 근접하거나 이를 초과하는 경우 다음 섹션에 설명된 대로 DAX 클러스터를 스케일 업해 보세요.

  DAX로 전송된 요청 수가 노드의 용량을 초과하는 경우 DAX는 추가 요청을 수락하는 속도를 제한합니다. ThrottlingException을 반환하여 이 작업을 수행합니다. DAX는 클러스터의 CPU 사용률을 지속적으로 평가하여 정상적인 클러스터 상태를 유지하면서 처리할 수 있는 요청 볼륨을 결정합니다.

  DAX가 CloudWatch에 게시하는 `ThrottledRequestCount` 지표를 모니터링할 수 있습니다. 이러한 예외가 정기적으로 표시되는 경우 클러스터를 확장하는 것이 좋습니다.

## 모니터링 데이터를 사용하여 DAX 클러스터 규모 조정

성능 지표를 모니터링하여 DAX 클러스터를 스케일 업 또는 스케일 다운해야 하는지 결정할 수 있습니다.

- 스케일 업 또는 스케일 아웃 - DAX 클러스터의 CPU 사용률이 높거나 캐시 적중률이 낮거나(캐싱 전략을 최적화한 후) 작동 지연 시간이 길면 클러스터를 스케일 업해야 합니다. 노드를 더 추가하면(스케일 아웃이라고도 함) 로드를 더 균등하게 분산하는 데 도움이 될 수 있습니다. 초당 쓰기 수가 증가하는 워크로드의 경우 더 강력한 노드를 선택(스케일 업)해야 할 수 있습니다.
- 스케일 다운 - CPU 사용률이 꾸준히 낮고 작동 지연 시간이 임계값 이하로 계속 유지되면 리소스가 과도하게 프로비저닝된 것일 수 있습니다. 이러한 경우에는 노드를 스케일 다운하여 비용을 절감하세요. 사용률이 낮은 기간에는 노드 수를 1개로 줄일 수는 있어도 클러스터를 완전히 종료할 수는 없습니다.

# 다른 AWS 서비스와 함께 DynamoDB 사용

Amazon DynamoDB는 다른 AWS 서비스와 통합되어 반복 태스크를 자동화하거나 여러 서비스를 사용하는 애플리케이션을 빌드할 수 있습니다.

주제


- DynamoDB용 Amazon Cognito를 사용하여 파일에서 AWS 자격 증명 구성
- Amazon Redshift와 통합
- Amazon EMR의 Apache Hive로 DynamoDB 데이터 처리
- Amazon S3와 DynamoDB 통합
- Amazon SageMaker Lakehouse와의 DynamoDB 제로 ETL 통합
- Amazon OpenSearch Service와 Amazon DynamoDB의 제로 ETL 통합
- DynamoDB와 Amazon EventBridge 통합
- DynamoDB를 Amazon Managed Streaming for Apache Kafka와 통합
- DynamoDB와의 통합 모범 사례


# DynamoDB용 Amazon Cognito를 사용하여 파일에서 AWS 자격 증명 구성

웹 및 모바일 애플리케이션용 AWS 자격 증명을 얻으려면 Amazon Cognito를 사용하는 것이 좋습니다. Amazon Cognito를 사용하면 파일에서 AWS 자격 증명을 하드 코딩할 필요가 없습니다. AWS Identity and Access Management(IAM) 역할을 사용하여 애플리케이션의 인증 및 미인증 사용자를 위한 임시 자격 증명을 생성합니다.

예를 들어, Amazon Cognito 미인증 역할을 사용해 Amazon DynamoDB 웹 서비스에 액세스하도록 JavaScript 파일을 구성하려면 다음을 수행합니다.

Amazon Cognito와 통합하도록 자격 증명을 구성하려면

1. 인증받지 않은 자격 증명을 허용하는 Amazon Cognito 자격 증명 풀을 생성합니다.

```
aws cognito-identity create-identity-pool \
    --identity-pool-name DynamoPool \
    --allow-unauthenticated-identities \
    --output json
```

```
{
    "IdentityPoolId": "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
    "AllowUnauthenticatedIdentities": true,
    "IdentityPoolName": "DynamoPool"
}
```

2. 다음 정책을 myCognitoPolicy.json이라는 파일에 복사합니다. 자격 증명 풀 ID(*us-west-2:12345678-1ab2-123a-1234-a12345ab12*)를 이전 단계에서 얻은 자체 IdentityPoolId으로 바꿉니다.

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "us-
west-2:12345678-1ab2-123a-1234-a12345ab12"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "unauthenticated"
        }
      }
    }
  ]
}
```

3. IAM 역할을 만들고 이전에 만든 정책을 위임합니다. 이를 통해 Amazon Cognito는 Cognito_DynamoPoolUnauth 역할을 맡을 수 있는 신뢰할 수 있는 엔터티가 됩니다.

```
aws iam create-role --role-name Cognito_DynamoPoolUnauth \
--assume-role-policy-document file://PathToFile/myCognitoPolicy.json --output json
```

4. 관리형 정책(AmazonDynamoDBFullAccess)을 연결함으로써 DynamoDB에 대한 모든 액세스 권한을 Cognito_DynamoPoolUnauth 역할에 부여합니다.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess \
--role-name Cognito_DynamoPoolUnauth
```

> **Note**
>
> 다른 방식으로 DynamoDB에 대해 세분화된 액세스를 부여할 수 있습니다. 자세한 내용은 IAM 정책 조건을 사용하여 세분화된 액세스 제어 구현을 참조하세요.

5. IAM 역할의 Amazon 리소스 이름(ARN)을 가져와 복사합니다.

```
aws iam get-role --role-name Cognito_DynamoPoolUnauth --output json
```

6. 자격 증명 풀에 DynamoPool역할Cognito_DynamoPoolUnauth을 추가합니다. 지정할 형식은 KeyName=string입니다. 여기서 KeyName는 unauthenticated이고 문자열은 이전 단계에서 얻은 역할 ARN입니다.

```
aws cognito-identity set-identity-pool-roles \
--identity-pool-id "us-west-2:12345678-1ab2-123a-1234-a12345ab12" \
--roles unauthenticated=arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth --
output json
```

7. 파일에서 Amazon Cognito 자격 증명을 지정합니다. IdentityPoolId및 RoleArn을 적절하게 변경합니다.

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
IdentityPoolId: "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
RoleArn: "arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth"
});
```

이제 Amazon Cognito 자격 증명을 사용하여 DynamoDB 웹 서비스에서 JavaScript 프로그램을 실행할 수 있습니다. 자세한 내용은 AWS SDK for JavaScript 시작 안내서에서 웹 브라우저에서 보안 인증 정보 설정을 참조하세요.

# Amazon Redshift와 통합

Amazon Redshift는 속도가 빠른 페타바이트 규모의 완전 관리형 데이터 웨어하우스 서비스로, 간편하고 비용 효율적으로 모든 데이터를 기존 비즈니스 인텔리전스 도구를 사용하여 효율적으로 분석할 수 있게 해줍니다.

DynamoDB와 Amazon Redshift를 함께 사용하여 애플리케이션 또는 데이터 에코시스템 내의 다양한 데이터 저장 및 처리 요구 사항을 해결할 수 있습니다.

DynamoDB를 Amazon Redshift와 통합하는 방법에 대한 자세한 내용은 아래 주제를 참조하세요.

주제


- CMK와 교차 계정 통합 시 고려 사항
- Amazon Redshift와 DynamoDB의 제로 ETL 통합
- COPY 명령을 사용하여 DynamoDB에서 Amazon Redshift로 데이터 로드


## CMK와 교차 계정 통합 시 고려 사항

DynamoDB에서 Amazon Redshift로 통합을 시도하면 Amazon Redshift에서 초기 작업이 시작됩니다. 적절한 권한이 없는 경우 이 작업이 자동으로 실패할 수 있습니다. 다음 섹션에서는 이 교차 계정 통합에 필요한 권한을 자세히 설명합니다.

### 필요한 AWS KMS 정책 및 권한

예시의 다음 자리 표시자를 교체합니다.

- `REDSHIFT_ACCOUNT_ID`: Amazon Redshift가 호스팅되는 AWS 계정 ID
- `DYNAMODB_ACCOUNT_ID`: DynamoDB가 호스팅되는 AWS 계정 ID
- `REDSHIFT_ROLE_NAME`: Amazon Redshift에서 사용하는 IAM 역할 이름
- `REGION`: 리소스가 위치한 AWS 리전
- `TABLE_NAME`: DynamoDB 테이블의 이름
- `KMS_KEY_ID`: KMS 키의 ID

DynamoDB 계정의 KMS 키 정책

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::DYNAMODB_ACCOUNT_ID:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow Redshift to use the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::REDSHIFT_ACCOUNT_ID:role/REDSHIFT_ROLE_NAME"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Redshift 역할에 대한 IAM 정책(Amazon Redshift 계정)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:ParitalBatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
```

```
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:REGION:DYNAMODB_ACCOUNT_ID:table/TABLE_NAME",
                "arn:aws:dynamodb:REGION:DYNAMODB_ACCOUNT_ID:table/TABLE_NAME/stream/*"
            ]
        },
        {
            "Sid": "AllowKMSAccess",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "arn:aws:kms:REGION:DYNAMODB_ACCOUNT_ID:key/KMS_KEY_ID"
        }
    ]
}
```

### Amazon Redshift 역할의 신뢰 관계

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

### DynamoDB 테이블 정책(리소스 기반 정책을 사용하는 경우)

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "AllowRedshiftAccess",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::REDSHIFT_ACCOUNT_ID:role/REDSHIFT_ROLE_NAME"
            },
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:ParitalBatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:REGION:DYNAMODB_ACCOUNT_ID:table/TABLE_NAME",
                "arn:aws:dynamodb:REGION:DYNAMODB_ACCOUNT_ID:table/TABLE_NAME/stream/*"
            ]
        }
    ]
}
```

## 중요 고려 사항

1. KMS 키가 DynamoDB 테이블과 동일한 리전에 있어야 합니다.
2. KMS 키가 AWS 관리형 키가 아닌 고객 관리형 키(CMK)여야 합니다.
3. DynamoDB 글로벌 테이블을 사용하는 경우 모든 관련 리전에 대한 권한을 구성합니다.
4. VPC 엔드포인트 또는 IP 범위에 따라 액세스를 제한하는 조건문을 추가하는 것이 좋습니다.
5. 보안 강화를 위해 `aws:PrincipalOrgID` 조건을 사용하여 조직에 대한 액세스를 제한하는 것이 좋습니다.
6. CloudTrail 및 CloudWatch 지표를 통해 KMS 키 사용을 모니터링합니다.

# Amazon Redshift와 DynamoDB의 제로 ETL 통합

Amazon Redshift와 Amazon DynamoDB의 제로 ETL 통합을 통해 코딩 없이 DynamoDB 데이터를 원활하게 분석할 수 있습니다. 이 완전 관리형 기능은 DynamoDB 테이블을 Amazon Redshift 데이터베이스에 자동으로 복제하므로 사용자가 복잡한 ETL 프로세스를 설정하지 않고도 DynamoDB 데이터에 대한 SQL 쿼리 및 분석을 실행할 수 있습니다. 통합은 DynamoDB 테이블에서 Amazon Redshift 데이터베이스로 데이터를 복제하는 방식으로 작동합니다.

통합을 설정하려면 DynamoDB 테이블을 소스로 지정하고 Amazon Redshift 데이터베이스를 대상으로 지정하면 됩니다. 활성화 시 통합은 전체 DynamoDB 테이블을 내보내 Amazon Redshift 데이터베이스를 채웁니다. 이 초기 프로세스를 완료하는 데 걸리는 시간은 DynamoDB 테이블 크기에 따라 달라집니다. 그런 다음 제로 ETL 통합은 DynamoDB 증분 내보내기를 사용하여 15~30분마다 DynamoDB에서 Amazon Redshift로 업데이트를 점진적으로 복제합니다. 따라서 Amazon Redshift에서 복제된 DynamoDB 데이터가 자동으로 최신 상태로 유지됩니다.

이 통합이 구성되면 사용자가 DynamoDB 테이블 성능에 영향을 주지 않고 표준 SQL 클라이언트 및 도구를 사용하여 Amazon Redshift에서 DynamoDB 데이터를 분석할 수 있습니다. 번거로운 ETL을 제거한 이 제로 ETL 통합은 Amazon Redshift 분석 및 기계 학습 기능을 통해 DynamoDB에서 인사이트를 빠르고 쉽게 발굴할 수 있도록 지원합니다.

주제


- Amazon Redshift와 DynamoDB 제로 ETL 통합을 생성하기 이전의 사전 조건
- Amazon Redshift와 DynamoDB의 제로 ETL 통합 사용 시 제한 사항
- Amazon Redshift와 DynamoDB의 제로 ETL 통합 생성
- Amazon Redshift와 DynamoDB의 제로 ETL 통합 보기
- Amazon Redshift와 DynamoDB의 제로 ETL 통합 삭제


## Amazon Redshift와 DynamoDB 제로 ETL 통합을 생성하기 이전의 사전 조건

1. 통합을 생성하기 전에 소스 DynamoDB 테이블과 대상 Amazon Redshift 클러스터를 생성해야 합니다. 이 정보는 1단계: 소스 DynamoDB 테이블 구성 및 2단계: Amazon Redshift 데이터 웨어하우스 생성에서 다룹니다.
2. Amazon DynamoDB와 Amazon Redshift 간의 제로 ETL 통합을 위해서는 소스 DynamoDB 테이블에 시점 복구(PITR)를 활성화해야 합니다.

3. 리소스 기반 정책의 경우 DynamoDB 테이블과 Amazon Redshift 데이터 웨어하우스가 동일한 계정에 있는 통합을 만들면 통합 만들기 단계에서 수정 옵션을 사용하여 필요한 리소스 정책을 DynamoDB와 Amazon Redshift 모두에 자동으로 적용할 수 있습니다.

   DynamoDB 테이블과 Amazon Redshift 데이터 웨어하우스가 서로 다른 AWS 계정에 있는 통합을 만들면 DynamoDB 테이블에 다음 리소스 정책을 적용해야 합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement that allows Amazon Redshift service to
 DescribeTable and ExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": [
                "dynamodb:ExportTableToPointInTime",
                "dynamodb:DescribeTable"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
                }
            }
        },
        {
            "Sid": "Statement that allows Amazon Redshift service to see all
 exports performed on the table",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "dynamodb:DescribeExport",
```

```
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/table-
name/export/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
                }
            }
        }
    ]
}
```

Amazon Redshift 데이터 웨어하우스에서 리소스 정책을 구성해야 할 수도 있습니다. 자세한 내용은 Amazon Redshift 데이터 웨어하우스에 대한 권한 부여 구성을 참조하세요.

4. ID 기반 정책의 경우:

   a. 통합을 생성하는 사용자에게 `GetResourcePolicy`, `PutResourcePolicy` 및 `UpdateContinuousBackups` 작업을 수행할 수 있도록 권한을 부여하는 ID 기반 정책이 필요합니다.

> **Note**
>
> 다음 정책 예시에서는 `arn:aws:redshift{-serverless}`로 리소스를 보여줍니다. 이는 네임스페이스가 Amazon Redshift 클러스터인지 Amazon Redshift Serverless 네임스페이스인지에 따라 arn이 `arn:aws:redshift` 또는 `arn:aws:redshift-serverless`가 될 수 있음을 보여주는 예입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables"
```

```
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetResourcePolicy",
                "dynamodb:PutResourcePolicy",
                "dynamodb:UpdateContinuousBackups"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:111122223333:table/table-name"
            ]
        },
        {
            "Sid": "AllowRedshiftDescribeIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeIntegrations"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowRedshiftCreateIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateIntegration",
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
        },
        {
            "Sid": "AllowRedshiftModifyDeleteIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:ModifyIntegration",
                "redshift:DeleteIntegration"
            ],
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:uuid"
        },
        {
            "Sid": "AllowRedshiftCreateInboundIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateInboundIntegration",
```

```
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:namespace:uuid"
        }
    ]
}
```

b. 대상 Amazon Redshift 네임스페이스를 구성하는 사용자는 PutResourcePolicy, DeleteResourcePolicy 및 GetResourcePolicy 작업을 수행할 수 있도록 권한을 부여하는 ID 기반 정책이 필요합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "redshift:PutResourcePolicy",
                "redshift:DeleteResourcePolicy",
                "redshift:GetResourcePolicy"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeInboundIntegrations"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
            ]
        }
    ]
}
```

5. 암호화 키 권한

고객 관리형 AWS KMS 키를 사용하여 소스 DynamoDB 테이블을 암호화하는 경우 KMS 키에 다음 정책을 추가해야 합니다. 이 정책은 Amazon Redshift가 KMS 키를 사용하여 암호화된 테이블에서 데이터를 내보낼 수 있도록 허용합니다.

```
{
    "Sid": "Statement to allow Amazon Redshift service to perform Decrypt operation
 on the source DynamoDB Table",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "redshift.amazonaws.com"
        ]
    },
    "Action": "kms:Decrypt",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "<account>"
        },
        "ArnEquals": {
            "aws:SourceArn": "arn:aws:redshift:<region>:<account>:integration:*"
        }
    }
}
```

Amazon Redshift 관리 가이드의 제로 ETL 통합 시작하기 단계에 따라 Amazon Redshift 네임스페이스의 권한을 구성할 수도 있습니다.

## Amazon Redshift와 DynamoDB의 제로 ETL 통합 사용 시 제한 사항

이 통합의 현재 릴리스에는 다음과 같은 일반 제한 사항이 적용됩니다. 이러한 제한 사항은 후속 릴리스에서 변경될 수 있습니다.

> **Note**
>
> 아래 제한 사항 외에도 제로 ETL 통합을 사용할 때의 일반적인 고려 사항을 검토하려면 Amazon Redshift 관리 안내서의 Amazon Redshift와 제로 ETL 통합을 사용할 때 고려할 사항을 참조하세요.

- DynamoDB 테이블과 Amazon Redshift 클러스터가 동일한 리전에 있어야 합니다.
- 소스 DynamoDB 테이블을 Amazon 소유 또는 고객 관리형 AWS KMS 키로 암호화해야 합니다. 소스 DynamoDB 테이블에는 Amazon 관리형 암호화가 지원되지 않습니다.

## Amazon Redshift와 DynamoDB의 제로 ETL 통합 생성

제로 ETL 통합을 생성하기 전에 먼저 소스 DynamoDB 테이블을 설정하고 나서 대상 Amazon Redshift 데이터 웨어하우스를 설정해야 합니다.

1단계: 소스 DynamoDB 테이블 구성

Amazon Redshift와 제로 ETL 통합을 생성하려면 테이블에서 시점 복구(PITR)를 활성화해야 합니다. PITR이 켜져 있지 않은 경우 콘솔에서 통합 설정 프로세스 중에 이 문제를 자동으로 해결할 수 있습니다. PITR을 활성화하는 방법에 대한 자세한 내용은 시점 복구를 참조하세요.

2단계: Amazon Redshift 데이터 웨어하우스 생성

Amazon Redshift 데이터 웨어하우스가 아직 없는 경우 이를 생성할 수 있습니다. Amazon Redshift Serverless 작업 그룹을 생성하려면 네임스페이스가 있는 작업 그룹 생성을 참조하세요. Amazon Redshift 클러스터를 생성하려면 클러스터 생성을 참조하세요.

통합에 성공하려면 대상 Amazon Redshift 작업 그룹 또는 클러스터에 enable_case_sensitive_identifier 파라미터가 켜져 있어야 합니다. 대소문자 구분 활성화에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 데이터 웨어하우스에 대소문자 구분 기능 사용 설정을 참조하세요.

Amazon Redshift 작업 그룹 또는 클러스터 설정이 완료되면 데이터 웨어하우스를 구성해야 합니다. 자세한 내용은 Amazon Redshift 관리 안내서의 제로 ETL 통합을 참조하세요.

3단계: DynamoDB 제로 ETL 통합 생성

제로 ETL 통합을 생성하기 전에 Amazon Redshift와 DynamoDB 제로 ETL 통합을 생성하기 이전의 사전 조건 섹션에서 작업을 완료해야 합니다. DynamoDB와 Amazon Redshift 간의 통합 생성은 2단계 프로세스입니다. 먼저 DynamoDB 에서 통합을 생성한 다음 새로 생성된 이 통합에 Amazon Redshift 데이터베이스를 연결합니다.

제로 ETL 통합 생성

1. AWS Management Console에 로그인하고 Amazon DynamoDB 콘솔(https://console.aws.amazon.com/dynamodbv2)을 엽니다.
2. 탐색 창에서 통합을 선택합니다.

3. 제로 ETL 통합 생성을 선택하고 Amazon Redshift를 선택합니다.
4. 그러면 Amazon Redshift 콘솔로 이동합니다. 절차를 계속하려면 DynamoDB의 제로 ETL 통합 생성의 DynamoDB 섹션을 참조하세요.

## Amazon Redshift와 DynamoDB의 제로 ETL 통합 보기

제로 ETL 통합의 세부 정보를 보고 구성 정보와 현재 상태를 확인할 수 있습니다.

Amazon DynamoDB 콘솔에서 제로 ETL 통합의 세부 정보를 보려면:

1. AWS Management Console에 로그인하고 Amazon DynamoDB 콘솔(https://console.aws.amazon.com/dynamodbv2)을 엽니다.
2. DynamoDB 콘솔에서 통합을 선택합니다.
3. 제로 ETL 통합 창에서 보려는 제로 ETL 통합을 선택합니다.

Amazon Redshift 콘솔에서 제로 ETL 통합의 세부 정보를 보려면:

1. AWS Management Console에 로그인하고 Amazon Redshift 콘솔(https://console.aws.amazon.com/redshiftv2)을 엽니다.
2. 제로 ETL 통합 보기의 단계를 따릅니다.

> **Note**
>
> Amazon Redshift와 제로 ETL 통합에서 나타날 수 있는 다양한 상태는 Amazon Redshift 관리 안내서의 제로 ETL 통합 보기에 나열되어 있습니다.

## Amazon Redshift와 DynamoDB의 제로 ETL 통합 삭제

제로 ETL 통합을 삭제하면 DynamoDB 또는 Amazon Redshift에서 데이터가 삭제되지 않지만 DynamoDB가 소스 테이블에서 Amazon Redshift 대상으로 데이터 전송을 중지합니다.

제로 ETL 통합을 삭제하려면

1. AWS Management Console에 로그인하고 Amazon DynamoDB 콘솔(https://console.aws.amazon.com/dynamodbv2)을 엽니다.
2. DynamoDB 콘솔에서 통합을 선택합니다.

3. 제로 ETL 통합 창에서 삭제할 제로 ETL 통합을 선택합니다.
4. 관리를 선택합니다. 그러면 통합 세부 정보 페이지로 이동합니다.
5. 삭제를 확인하려면 삭제를 선택합니다.

## COPY 명령을 사용하여 DynamoDB에서 Amazon Redshift로 데이터 로드

Amazon Redshift는 고급 비즈니스 인텔리전스 기능과 강력한 SQL 기반 인터페이스로 Amazon DynamoDB와 연동됩니다. DynamoDB 테이블에서 Amazon Redshift로 데이터를 복사할 때 Amazon Redshift 클러스터에 있는 다른 테이블과의 조인을 포함해 해당 데이터에 대한 복잡한 데이터 분석 쿼리를 수행할 수 있습니다.

프로비저닝된 처리량과 관련하여 DynamoDB 테이블에서 복사하는 작업은 해당 테이블의 읽기 용량에 포함됩니다. 데이터가 복사된 후에는 Amazon Redshift의 SQL 쿼리가 DynamoDB에 전혀 영향을 미치지 않습니다. 쿼리가 DynamoDB 자체보다는 DynamoDB에서 데이터를 복사하는 작업에 적용되기 때문입니다.

DynamoDB 테이블에서 데이터를 로드하려면 먼저 데이터의 대상으로 사용할 Amazon Redshift 테이블을 생성해야 합니다. NoSQL 환경에서 SQL 환경으로 데이터를 복사하는 것이며 다른 환경에는 적용되지 않고 하나의 환경에 적용되는 특정한 규칙이 있다는 점에 유의하세요. 고려해야 할 몇 가지 차이는 다음과 같습니다.

- DynamoDB 테이블 이름은 '.'(점) 및 '-'(대시) 문자를 비롯해 최대 255자를 포함할 수 있고 대/소문자를 구분합니다. Amazon Redshift 테이블 이름은 127자로 제한되고 점이나 대시를 포함할 수 없으며 대/소문자를 구분하지 않습니다. 또한 테이블 이름이 Amazon Redshift 예약어와 충돌해서는 안 됩니다.
- DynamoDB에서는 SQL의 NULL 개념을 지원하지 않습니다. Amazon Redshift에서 DynamoDB의 빈 속성 값을 NULL이나 빈 필드 중 무엇으로 처리할지를 지정해야 합니다.
- DynamoDB 데이터 형식이 Amazon Redshift 데이터 형식에 직접적으로 대응하지 않습니다. Amazon Redshift에 있는 각 열이 DynamoDB의 데이터를 수용할 수 있도록 데이터 형식과 크기가 적합해야 합니다.

다음은 Amazon Redshift SQL의 COPY 명령 예제입니다.

```
copy favoritemovies from 'dynamodb://my-favorite-movies-table'
credentials 'aws_access_key_id=<Your-Access-Key-ID>;aws_secret_access_key=<Your-Secret-Access-Key>'
```

```
readratio 50;
```

이 예제에서 DynamoDB의 소스 테이블은 `my-favorite-movies-table`이고, Amazon Redshift의 대상 테이블은 `favoritemovies`입니다. `readratio 50` 절은 사용하는 프로비저닝 처리량의 백분율을 조절합니다. 이 경우 `my-favorite-movies-table`에 대해 프로비저닝된 읽기 용량 단위의 50% 이내가 COPY 명령에 사용됩니다. 사용되지 않는 프로비저닝 처리량 미만의 값으로 이 비율을 설정하는 것이 좋습니다.

DynamoDB에서 Amazon Redshift로 데이터를 로드하는 방법에 대한 자세한 지침은 Amazon Redshift 데이터베이스 개발자 안내서의 다음 섹션을 참조하세요.

- DynamoDB 테이블에서 데이터 로드
- COPY 명령
- COPY 예제

# Amazon EMR의 Apache Hive로 DynamoDB 데이터 처리

Amazon DynamoDB는 Amazon EMR에서 실행되는 데이터 웨어하우징 애플리케이션인 Apache Hive와 통합됩니다. Hive는 DynamoDB 테이블에서 데이터를 읽고 쓸 수 있으므로 다음과 같은 작업이 가능합니다.

- 유사 SQL 언어(HiveQL)를 사용한 실시간 DynamoDB 데이터 쿼리
- 데이터를 DynamoDB 테이블에서 Amazon S3 버킷으로, 그리고 그 반대 방향으로 복사
- 데이터를 DynamoDB 테이블에서 Hadoop 분산 파일 시스템(HDFS)으로, 그리고 그 반대 방향으로 복사
- DynamoDB 테이블에서 조인 작업을 수행

주제

- 개요
- 자습서: Amazon DynamoDB 및 Apache Hive 작업
- Hive에서 외부 테이블 생성
- HiveQL 문 처리
- DynamoDB의 데이터 쿼리
- Amazon DynamoDB 간 데이터 복사

- 성능 튜닝

## 개요

Amazon EMR은 대량의 데이터를 쉽고 빠르고 경제적으로 처리할 수 있도록 지원하는 서비스입니다. Amazon EMR을 사용하려면 Hadoop 오픈 소스 프레임워크를 실행하는 Amazon EC2 인스턴스의 관리형 클러스터를 시작합니다. Hadoop은 MapReduce 알고리즘을 구현하는 분산 애플리케이션입니다. 이 알고리즘에서는 단일 작업이 클러스터 안의 여러 노드로 매핑됩니다. 각 코드는 지정된 작업을 다른 노드와 병렬로 처리합니다. 최종적으로 출력이 단일 노드로 환원되어 최종 결과를 생성합니다.

Amazon EMR 클러스터를 지속적으로 또는 일시적으로 실행하도록 선택할 수 있습니다.

- 지속적 클러스터는 사용자가 종료할 때까지 실행됩니다. 지속적 클러스터는 데이터 분석, 데이터 웨어하우징 또는 기타 대화식 용도에 적합합니다.
- 일시적 클러스터는 한 작업 플로우를 실행하는 데 필요한 시간 동안 실행된 후 자동으로 종료합니다. 일시적 클러스터는 스크립트 실행과 같은 주기적인 처리 작업에 적합합니다.

Amazon EMR 아키텍처 및 관리에 대한 자세한 내용은 Amazon EMR 관리 가이드를 참조하세요.

Amazon EMR 클러스터를 시작할 때 Amazon EC2 인스턴스의 초기 수와 유형을 지정합니다. 또한 해당 클러스터에서 실행할 (Hadoop 이외의) 다른 분산 애플리케이션도 지정합니다. 이러한 애플리케이션으로는 Hue, Mahout, Pig, Spark 등이 있습니다.

Amazon EMR용 애플리케이션에 대한 자세한 내용은 Amazon EMR 릴리즈 가이드를 참조하세요.

클러스터 구성에 따라 다음 유형의 노드가 하나 이상 존재합니다.

- 리더 노드 - 클러스터를 관리하여 코어 및 태스크 인스턴스 그룹으로 MapReduce 실행 파일 및 원시 데이터 하위 집합의 배포를 조정합니다. 또한 수행되는 각 작업의 상태를 추적하고 인스턴스 그룹의 상태를 모니터링합니다. 클러스터에는 리더 노드가 한 개만 있습니다.
- 코어 노드 - MapReduce 작업을 실행하고 Hadoop 분산 파일 시스템(HDFS)을 사용하여 데이터를 저장합니다.
- 작업 노드(선택 사항) - MapReduce 작업을 실행합니다.

## 자습서: Amazon DynamoDB 및 Apache Hive 작업

이 자습서에서는 Amazon EMR 클러스터를 시작한 다음 Apache Hive를 사용하여 DynamoDB 테이블에 저장된 데이터를 처리합니다.

Hive는 여러 소스의 데이터를 처리 및 분석할 수 있게 해주는, Hadoop용 데이터 웨어하우스 애플리케이션입니다. Hive는 SQL과 유사한 언어인 HiveQL을 제공합니다. 이 언어를 사용하면 Amazon EMR 클러스터에 로컬로 저장된 데이터 또는 외부 데이터 원본(예: Amazon DynamoDB)에 저장된 데이터로 작업할 수 있습니다.

자세한 내용은 Hive Tutorial을 참조하세요.

주제


- 시작하기 전 준비 사항
- 1단계: Amazon EC2 키 페어 생성
- 2단계: Amazon EMR 클러스터 시작
- 3단계: 리더 노드에 연결
- 4단계: HDFS로 데이터 로드
- 5단계: DynamoDB로 데이터 복사
- 6단계: DynamoDB 테이블의 데이터 쿼리
- 7단계: (선택 사항) 정리


## 시작하기 전 준비 사항

이 자습서를 이해하려면 다음이 필요합니다.

- AWS 계정. 계정이 없을 경우 AWS에 가입 단원을 참조하세요.
- SSH 클라이언트(Secure Shell). SSH 클라이언트를 사용하여 Amazon EMR 클러스터의 리더 노드에 연결하고 대화형 명령을 실행합니다. SSH 클라이언트는 대부분의 Linux, Unix 및 Mac OS X 설치 시 기본적으로 사용됩니다. Windows 사용자는 SSH가 지원되는 PuTTY 클라이언트를 다운로드하여 설치하면 됩니다.

다음 단계

1단계: Amazon EC2 키 페어 생성

## 1단계: Amazon EC2 키 페어 생성

이 단계에서는 Amazon EMR 리더 노드에 연결하고 Hive 명령을 실행하는 데 필요한 Amazon EC2 키 페어를 생성합니다.

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 리전을 선택합니다(예: `US West (Oregon)`). 이 리전은 DynamoDB 테이블이 있는 리전과 동일해야 합니다.
3. 탐색 창에서 [Key Pairs]를 선택합니다.
4. 키 페어 생성(Create Key Pair)을 선택합니다.
5. Key pair name에서 키 페어의 이름(예: `mykeypair`)을 입력한 다음 Create를 선택합니다.
6. 프라이빗 키 파일을 다운로드합니다. 파일 이름은 `.pem`으로 끝납니다(`mykeypair.pem` 등). 프라이빗 키 파일은 안전한 곳에 보관하세요. 이 키 페어를 사용하여 시작한 모든 Amazon EMR 클러스터에 액세스하는 데 필요합니다.

   > **Important**
   >
   > 키 페어를 잃어버린 경우에는 Amazon EMR 클러스터의 리더 노드에 연결할 수 없습니다.

   Amazon EC2 키 페어에 대한 자세한 내용은 Amazon EC2 사용 설명서에서 Amazon EC2 키 페어를 참조하세요.

다음 단계

2단계: Amazon EMR 클러스터 시작

## 2단계: Amazon EMR 클러스터 시작

이 단계에서는 Amazon EMR 클러스터를 구성하고 시작합니다. Hive와 DynamoDB용 스토리지 핸들러는 이미 클러스터에 설치되어 있을 것입니다.

1. https://console.aws.amazon.com/emr에서 Amazon EMR 콘솔을 엽니다.
2. Create Cluster를 선택합니다.
3. Create Cluster - Quick Options 페이지에서 다음을 수행합니다.
   a. Cluster name에 클러스터 이름을 입력합니다(예: `My EMR cluster`).
   b. EC2 key pair에서 앞서 생성한 키 페어를 선택합니다.

   기타 설정은 기본값을 유지합니다.

4. 클러스터 생성을 선택합니다.

클러스터를 시작하는 데 몇 분이 걸릴 수 있습니다. Amazon EMR 콘솔의 Cluster Details(클러스터 세부 정보) 페이지에서 진행 상황을 모니터링할 수 있습니다.

`Waiting` 상태로 변경된 이후에 클러스터를 사용할 수 있습니다.

클러스터 로그 파일 및 Amazon S3

Amazon EMR 클러스터는 클러스터 상태 정보와 디버깅 정보를 포함하는 로그 파일을 생성합니다. Create Cluster - Quick Options(클러스터 생성 - 빠른 옵션)의 기본 설정에는 Amazon EMR 로깅 설정이 포함됩니다.

없는 경우 AWS Management Console에서 Amazon S3 버킷을 생성합니다. 버킷 이름은 `aws-logs-`*`account-id`*`-`*`region`*이며, 여기에서 *`account-id`*는 AWS 계정 번호, *`region`*은 클러스터를 시작한 리전입니다(예: `aws-logs-123456789012-us-west-2`).

> **Note**
>
> Amazon S3 콘솔을 사용하여 로그 파일을 볼 수 있습니다. 자세한 내용은 Amazon EMR 관리 가이드의 로그 파일 보기 단원을 참조하세요.

이 버킷을 로깅 이외에 용도로 사용할 수 있습니다. 예를 들어, 버킷을 Hive 스크립트를 저장하기 위한 위치로 사용하거나 Amazon DynamoDB에서 Amazon S3로 데이터를 내보낼 때 대상으로 사용할 수 있습니다.

다음 단계

3단계: 리더 노드에 연결

## 3단계: 리더 노드에 연결

Amazon EMR 클러스터의 상태가 `Waiting`으로 바뀌면 SSH를 사용하여 리더 노드에 연결하여 명령줄 작업을 수행할 수 있습니다.

1. Amazon EMR 콘솔에서 상태를 볼 클러스터의 이름을 선택합니다.
2. Cluster Details(클러스터 세부 정보) 페이지에서 Leader public DNS(리더 퍼블릭 DNS) 필드를 찾습니다. 이 필드는 Amazon EMR 클러스터 리더 노드의 퍼블릭 DNS 이름입니다.
3. DNS 이름 오른쪽에서 SSH 링크를 선택합니다.

4. SSH를 사용하여 리더 노드에 연결의 지침을 따릅니다.

   운영 체제에 따라 Windows 탭 또는 Mac/Linux 탭을 선택하고 리더 노드를 연결하는 지침을 따릅니다.

SSH 또는 PuTTY를 사용하여 리더 노드에 연결되면 다음과 비슷한 명령 프롬프트가 나타납니다.

```
[hadoop@ip-192-0-2-0 ~]$
```

다음 단계

4단계: HDFS로 데이터 로드

## 4단계: HDFS로 데이터 로드

이 단계에서는 데이터 파일을 Hadoop 분산 파일 시스템(HDFS)으로 복사한 후 해당 데이터 파일로 매핑되는 외부 Hive 테이블을 생성합니다.

샘플 데이터 다운로드

1. 샘플 데이터 아카이브(`features.zip`)를 다운로드합니다.

   ```
   wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   features.zip
   ```

2. 아카이브에서 `features.txt` 파일을 추출합니다.

   ```
   unzip features.zip
   ```

3. `features.txt` 파일의 처음 몇 줄을 확인합니다.

   ```
   head features.txt
   ```

   결과가 다음과 비슷할 것입니다.

   ```
   1535908|Big Run|Stream|WV|38.6370428|-80.8595469|794
   875609|Constable Hook|Cape|NJ|40.657881|-74.0990309|7
   1217998|Gooseberry Island|Island|RI|41.4534361|-71.3253284|10
   26603|Boone Moore Spring|Spring|AZ|34.0895692|-111.410065|3681
   1506738|Missouri Flat|Flat|WA|46.7634987|-117.0346113|2605
   1181348|Minnow Run|Stream|PA|40.0820178|-79.3800349|1558
   ```

```
1288759|Hunting Creek|Stream|TN|36.343969|-83.8029682|1024
533060|Big Charles Bayou|Bay|LA|29.6046517|-91.9828654|0
829689|Greenwood Creek|Stream|NE|41.596086|-103.0499296|3671
541692|Button Willow Island|Island|LA|31.9579389|-93.0648847|98
```

`features.txt` 파일에는 미국 지명위원회(http://geonames.usgs.gov/domestic/download_data.htm)의 데이터의 하위 집합이 포함되어 있습니다. 각 행의 필드는 다음 정보를 표시합니다.

- 지형 ID(고유 식별자)
- 명칭
- 클래스(호수, 산림, 강 등)
- State
- 위도(각도)
- 경도(각도)
- 고도(피트)

4. 명령 프롬프트에 다음 명령을 입력합니다.

```
hive
```

명령 프롬프트 이렇게 바뀝니다. `hive>`

5. 다음 HiveQL 문을 입력하여 고유 Hive 테이블을 생성합니다.

```
CREATE TABLE hive_features
    (feature_id             BIGINT,
    feature_name            STRING ,
    feature_class           STRING ,
    state_alpha             STRING,
    prim_lat_dec            DOUBLE ,
    prim_long_dec           DOUBLE ,
    elev_in_ft              BIGINT)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '|'
    LINES TERMINATED BY '\n';
```

6. 다음 HiveQL 문을 입력하여 테이블에 데이터를 로드합니다.

```
LOAD DATA
```

```
LOCAL
INPATH './features.txt'
OVERWRITE
INTO TABLE hive_features;
```

7. 이제 고유 Hive 테이블에 `features.txt` 파일 데이터가 채워졌습니다. 확인하려면 다음 HiveQL 문을 입력합니다.

```
SELECT state_alpha, COUNT(*)
FROM hive_features
GROUP BY state_alpha;
```

출력에 주 목록과 각 주의 지형 수가 표시되어야 합니다.

다음 단계

[5단계: DynamoDB로 데이터 복사]

## 5단계: DynamoDB로 데이터 복사

이 단계에서는 Hive 테이블(`hive_features`)의 데이터를 DynamoDB의 새 테이블로 복사합니다.

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. Create Table(테이블 생성)을 선택합니다.
3. Create DynamoDB table 페이지에서 다음 작업을 수행합니다.

   a. 테이블에 **`Features`**를 입력합니다.

   b. 기본 키의 파티션 키 필드에 **`Id`**를 입력합니다. 데이터 형식을 [Number]로 설정합니다.

   Use Default Settings를 지웁니다. Provisioned Capacity에 대해 다음을 입력합니다.

   - 읽기 용량 단위 - `10`
   - 쓰기 용량 단위 - `10`

   생성(Create)을 선택합니다.

4. Hive 프롬프트에서 다음 HiveQL 문을 입력합니다.

```
CREATE EXTERNAL TABLE ddb_features
    (feature_id   BIGINT,
```

```
    feature_name  STRING,
    feature_class STRING,
    state_alpha   STRING,
    prim_lat_dec  DOUBLE,
    prim_long_dec DOUBLE,
    elev_in_ft    BIGINT)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES(
    "dynamodb.table.name" = "Features",

 "dynamodb.column.mapping"="feature_id:Id,feature_name:Name,feature_class:Class,state_alpha
);
```

이제 Hive와 DynamoDB의 Features 테이블에 사이에 매핑이 설정되었습니다.

5. 다음 HiveQL 문을 입력하여 DynamoDB로 데이터를 가져옵니다.

```
INSERT OVERWRITE TABLE ddb_features
SELECT
    feature_id,
    feature_name,
    feature_class,
    state_alpha,
    prim_lat_dec,
    prim_long_dec,
    elev_in_ft
FROM hive_features;
```

Hive가 MapReduce 작업을 제출하고, Amazon EMR 클러스터가 작업을 처리합니다. 작업을 완료하는 데 몇 분이 걸릴 수 있습니다.

6. DynamoDB에 데이터가 로드되었는지 확인합니다.

   a. DynamoDB 콘솔 탐색 창에서 Tables(테이블)를 선택합니다.

   b. Features 테이블을 선택한 다음 Items 탭을 선택하여 데이터를 봅니다.

다음 단계

6단계: DynamoDB 테이블의 데이터 쿼리

## 6단계: DynamoDB 테이블의 데이터 쿼리

이 단계에서는 HiveQL을 사용하여 DynamoDB에서 Features 테이블을 쿼리합니다. 다음 Hive 쿼리를 시도해 보세요.

1. 알파벳 순으로 모든 지형 유형(`feature_class`):

```
SELECT DISTINCT feature_class
FROM ddb_features
ORDER BY feature_class;
```

2. "M"으로 시작하는 모든 호수:

```
SELECT feature_name, state_alpha
FROM ddb_features
WHERE feature_class = 'Lake'
AND feature_name LIKE 'M%'
ORDER BY feature_name;
```

3. 고도가 1마일(5,280피트) 이상인 지형이 3개 이상인 주:

```
SELECT state_alpha, feature_class, COUNT(*)
FROM ddb_features
WHERE elev_in_ft > 5280
GROUP by state_alpha, feature_class
HAVING COUNT(*) >= 3
ORDER BY state_alpha, feature_class;
```

다음 단계

7단계: (선택 사항) 정리

## 7단계: (선택 사항) 정리

이제 자습서를 모두 마쳤습니다. 이 단원을 계속 읽으면서 Amazon EMR에서의 DynamoDB 데이터 작업에 대해 보다 자세히 알아볼 수 있습니다. 그러는 동안 Amazon EMR 클러스터를 계속 실행해야 할 수 있습니다.

더 이상 클러스터가 필요하지 않을 경우 클러스터를 종료하고 관련 리소스를 모두 제거해야 합니다. 그래야 불필요한 리소스에 대해 요금을 지불하지 않을 수 있습니다.

1. Amazon EMR 클러스터를 종료합니다.
   a. https://console.aws.amazon.com/emr에서 Amazon EMR 콘솔을 엽니다.
   b. Amazon EMR 클러스터를 선택하고 Terminate(종료)를 선택한 다음 확인합니다.
2. DynamoDB에서 Features 테이블을 삭제합니다.
   a. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
   b. 탐색 창에서 테이블을 선택합니다.
   c. Features 테이블을 선택합니다. Actions 메뉴에서 Delete Table을 선택합니다.
3. Amazon EMR 로그 파일을 포함하는 Amazon S3 버킷을 삭제합니다.
   a. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
   b. 버킷 목록에서 `aws-logs-` *`accountID`*`-`*`region`*을 선택합니다. 여기서 *`accountID`*는 AWS 계정 번호이고, *`region`*은 클러스터를 시작한 리전입니다.
   c. Action 메뉴에서 Delete를 선택합니다.

## Hive에서 외부 테이블 생성

자습서: Amazon DynamoDB 및 Apache Hive 작업에서 DynamoDB 테이블로 매핑된 외부 Hive 테이블을 생성했습니다. 외부 테이블에 대해 HiveQL 문을 실행했을 때 읽기 및 쓰기 작업이 DynamoDB 테이블로 전달되었습니다.

외부 테이블을 다른 위치에서 관리 및 저장되는 데이터 원본에 대한 포인터로 생각할 수 있습니다. 이 경우 기본 데이터 원본은 DynamoDB 테이블입니다. (테이블이 이미 있어야 합니다. Hive 내부에서 DynamoDB 테이블을 생성, 업데이트, 삭제할 수 없습니다.) `CREATE EXTERNAL TABLE` 문을 사용하여 외부 테이블을 생성합니다. 그러면 데이터가 Hive 내부에 로컬로 저장된 것처럼 HiveQL을 사용하여 DynamoDB 내 데이터로 작업할 수 있습니다.

> **Note**
>
> `INSERT` 문을 사용하여 외부 테이블에 데이터를 삽입하고, `SELECT` 문을 사용하여 외부 테이블에서 데이터를 선택할 수 있습니다. 하지만 `UPDATE` 또는 `DELETE` 문을 사용하여 테이블 내 데이터를 조작할 수는 없습니다.

더 이상 외부 테이블이 필요하지 않은 경우 DROP TABLE 문을 사용하여 제거할 수 있습니다. 이 경우 DROP TABLE 문은 Hive 내 외부 테이블만 제거합니다. 기본 DynamoDB 테이블 또는 그 안의 데이터에는 영향을 주지 않습니다.

주제

- CREATE EXTERNAL TABLE 구문
- 데이터 형식 매핑

## CREATE EXTERNAL TABLE 구문

다음은 DynamoDB 테이블로 매핑되는 외부 Hive 테이블을 생성하기 위한 HiveQL 구문입니다.

```
CREATE EXTERNAL TABLE hive_table

 (hive_column1_name hive_column1_datatype, hive_column2_name hive_column2_datatype...)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES (
    "dynamodb.table.name" = "dynamodb_table",
    "dynamodb.column.mapping" =
 "hive_column1_name:dynamodb_attribute1_name,hive_column2_name:dynamodb_attribute2_name..."
);
```

행 1은 CREATE EXTERNAL TABLE 문의 시작입니다. 여기서 생성하려는 Hive 테이블(hive_table)의 이름을 제공합니다.

행 2는 hive_table의 열 및 데이터 형식을 지정합니다. DynamoDB 테이블의 속성에 해당하는 열 및 데이터 형식을 정의해야 합니다.

행 3은 STORED BY 절입니다. 여기서 Hive와 DynamoDB 테이블 사이의 데이터 관리를 처리하는 클래스를 지정합니다. DynamoDB의 경우 STORED BY를 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'로 설정해야 합니다.

행 4는 TBLPROPERTIES 절의 시작 부분입니다. 여기서 DynamoDBStorageHandler에 대해 다음 파라미터를 정의합니다.

- dynamodb.table.name - DynamoDB 테이블의 이름
- dynamodb.column.mapping - Hive 테이블의 열 이름과 해당하는 DynamoDB 테이블 속성의 쌍. 각 쌍은 hive_column_name:dynamodb_attribute_name 형식이며 쉼표로 구분됩니다.

다음을 참조하세요.

- Hive 테이블 이름이 DynamoDB 테이블 이름과 같을 필요는 없습니다.
- Hive 테이블 열 이름이 DynamoDB 테이블의 열 이름과 같을 필요는 없습니다.
- `dynamodb.table.name`에서 지정된 테이블이 DynamoDB에 존재해야 합니다.
- `dynamodb.column.mapping`의 경우:
  - DynamoDB 테이블의 키 스키마 속성을 매핑해야 합니다. 여기에는 파티션 키와 정렬 키(있는 경우)가 포함됩니다.
  - DynamoDB 테이블의 키가 아닌 속성을 매핑할 필요는 없습니다. 하지만 Hive 테이블을 쿼리할 때 이들 속성의 데이터는 볼 수 없게 됩니다.
  - Hive 테이블 열과 DynamoDB 속성의 데이터 형식이 호환되지 않을 경우 Hive 테이블을 쿼리하면 이러한 열에 `NULL`이 표시됩니다.

> **Note**
>
> `CREATE EXTERNAL TABLE` 문은 `TBLPROPERTIES` 절에 대한 검증을 수행하지 않습니다. `dynamodb.table.name` 및 `dynamodb.column.mapping`에 지정한 값은 테이블에 액세스할 때만 `DynamoDBStorageHandler` 클래스에 의해 평가됩니다.

## 데이터 형식 매핑

다음 표에는 DynamoDB 데이터 형식 및 호환 Hvie 데이터 형식이 나와 있습니다.

| DynamoDB 데이터 형식 | Hive 데이터 형식 |
| --- | --- |
| String | `STRING` |
| 숫자 | `BIGINT` 또는 `DOUBLE` |
| 바이너리 | `BINARY` |
| 문자열 집합 | `ARRAY<STRING>` |
| 숫자 집합 | `ARRAY<BIGINT>` 또는 `ARRAY<DOUBLE>` |
| 이진수 집합 | `ARRAY<BINARY>` |

> **Note**
>
> 다음 DynamoDB 데이터 형식은 `DynamoDBStorageHandler` 클래스에서 지원하지 않으므로 `dynamodb.column.mapping`과 함께 사용할 수 없습니다.
>
> - 맵
> - 나열
> - 불
> - Null
>
> 하지만 이러한 데이터 유형을 사용해야 하는 경우 전체 DynamoDB 항목을 맵의 키와 값 모두에 대한 문자열 맵으로 나타내는 `item`이라는 단일 엔터티를 만들 수 있습니다. 자세한 내용은 열 매핑 없이 데이터 복사 단원을 참조하세요.

숫자 형식의 DynamoDB 속성을 매핑하기 원할 경우 적절한 Hive 형식을 선택해야 합니다.

- Hive `BIGINT` 형식은 부호 있는 8바이트 정수입니다. Java의 `long` 데이터 형식과 동일합니다.
- Hive `DOUBLE` 형식은 8비트 배정밀도 부동 소수점 수입니다. Java의 `double` 형식과 동일합니다.

선택한 Hive 데이터 형식보다 정밀도가 높은 수치 데이터가 DynamoDB에 저장된 경우 DynamoDB 데이터에 액세스하면 정밀도가 손실될 수 있습니다.

이진수 형식의 데이터를 DynamoDB에서 (Amazon S3) 또는 HDFS로 내보낼 경우 데이터는 Base64로 인코딩된 문자열로 저장됩니다. 데이터를 Amazon S3 또는 HDFS에서 DynamoDB 이진수 형식으로 가져올 경우 데이터가 Base64 문자열로 인코딩되었는지 확인해야 합니다.

## HiveQL 문 처리

Hive는 MapReduce 작업을 실행하기 위한 일괄 처리 중심 프레임워크인 Hadoop에서 실행되는 애플리케이션입니다. HiveQL 문을 실행하면 Hive가 결과를 즉시 반환할 수 있는지 또는 MapReduce 작업을 제출해야 하는지 판단합니다.

예를 들어 ddb_features 테이블을 고려해 봅니다(자습서: Amazon DynamoDB 및 Apache Hive 작업 참조). 다음 Hive 쿼리는 주 약어와 각 주의 산정 수를 표시합니다.

```
SELECT state_alpha, count(*)
```

```
FROM ddb_features
WHERE feature_class = 'Summit'
GROUP BY state_alpha;
```

Hive는 결과를 즉시 반환하지 않습니다. 대신, MapReduce 작업을 제출하고, Hadoop 프레임워크가 이 작업을 처리합니다. Hive는 작업이 완료될 때까지 대기했다가 쿼리 결과를 표시합니다.

```
AK  2
AL  2
AR  2
AZ  3
CA  7
CO  2
CT  2
ID  1
KS  1
ME  2
MI  1
MT  3
NC  1
NE  1
NM  1
NY  2
OR  5
PA  1
TN  1
TX  1
UT  4
VA  1
VT  2
WA  2
WY  3
Time taken: 8.753 seconds, Fetched: 25 row(s)
```

## 작업 모니터링 및 취소

Hive는 Hadoop 작업을 시작하면 해당 작업의 출력을 표시합니다. 작업이 진행되는 동안 작업 완료 상태가 업데이트됩니다. 일부 경우에는 상태가 장시간 업데이트되지 않을 수도 있습니다. (할당된 읽기 용량 설정이 낮은 대규모 DynamoDB 테이블을 쿼리하는 경우 그럴 수 있습니다.)

작업을 완료 전에 취소해야 할 경우 언제든지 **Ctrl+C**를 입력하면 됩니다.

# DynamoDB의 데이터 쿼리

다음 예제에서는 HiveQL을 사용하여 DynamoDB에 저장된 데이터를 쿼리할 수 있는 몇 가지 방법을 보여 줍니다.

이러한 예제에서는 자습서(5단계: DynamoDB로 데이터 복사)의 ddb_features 테이블을 참조합니다.

주제

- 집계 함수 사용
- GROUP BY 및 HAVING 절 사용
- 두 DynamoDB 테이블 조인
- 서로 다른 소스의 테이블 조인

## 집계 함수 사용

HiveQL은 데이터 값을 요약하기 위한 내장 함수를 제공합니다. 예를 들어 MAX 함수를 사용하여 선택한 열에서 최대 값을 찾을 수 있습니다. 다음 예제는 콜로라도 주에서 가장 높은 지형의 고도를 반환합니다.

```
SELECT MAX(elev_in_ft)
FROM ddb_features
WHERE state_alpha = 'CO';
```

## GROUP BY 및 HAVING 절 사용

GROUP BY 절을 사용하여 여러 레코드에서 데이터를 수집할 수 있습니다. 이 절은 종종 SUM, COUNT, MIN, MAX와 같은 집계 함수와 함께 사용됩니다. 또한 HAVING 절을 사용하여 특정 기준을 충족하지 않는 결과를 모두 폐기할 수도 있습니다.

다음 예제는 ddb_features 테이블에서 5개 이상의 지형이 있는 주에서 최고 고도의 목록을 반환합니다.

```
SELECT state_alpha, max(elev_in_ft)
FROM ddb_features
GROUP BY state_alpha
HAVING count(*) >= 5;
```

## 두 DynamoDB 테이블 조인

다음은 다른 Hive 테이블(east_coast_states)을 DynamoDB의 테이블로 매핑하는 예입니다. SELECT 문은 이들 두 테이블의 조인입니다. 조인은 클러스터에서 계산되어 반환됩니다. DynamoDB에서는 조인이 일어나지 않습니다.

다음 데이터를 포함하는 EastCoastStates라는 이름의 DynamoDB 테이블을 생각해 보겠습니다.

```
StateName       StateAbbrev

Maine           ME
New Hampshire   NH
Massachusetts   MA
Rhode Island    RI
Connecticut     CT
New York        NY
New Jersey      NJ
Delaware        DE
Maryland        MD
Virginia        VA
North Carolina  NC
South Carolina  SC
Georgia         GA
Florida         FL
```

이 테이블을 east_coast_states라는 Hive 외부 테이블로 사용 가능하다고 가정하겠습니다.

```
CREATE EXTERNAL TABLE ddb_east_coast_states (state_name STRING, state_alpha STRING)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "EastCoastStates",
"dynamodb.column.mapping" = "state_name:StateName,state_alpha:StateAbbrev");
```

다음 조인은 지형이 3개 이상 있는 미국 동부 연안 주를 반환합니다.

```
SELECT ecs.state_name, f.feature_class, COUNT(*)
FROM ddb_east_coast_states ecs
JOIN ddb_features f on ecs.state_alpha = f.state_alpha
GROUP BY ecs.state_name, f.feature_class
HAVING COUNT(*) >= 3;
```

### 서로 다른 소스의 테이블 조인

다음 예제에서는 s3_east_coast_states가 Amazon S3에 저장된 CSV 파일과 연결된 Hive 테이블입니다. ddb_features 테이블은 DynamoDB의 데이터와 연결됩니다. 다음 예제는 이들 두 테이블을 조인하여 이름이 "New"로 시작하는 주에서 지형을 반환합니다.

```
create external table s3_east_coast_states (state_name STRING, state_alpha STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://bucketname/path/subpath/';
```

```
SELECT ecs.state_name, f.feature_name, f.feature_class
FROM s3_east_coast_states ecs
JOIN ddb_features f
ON ecs.state_alpha = f.state_alpha
WHERE ecs.state_name LIKE 'New%';
```

## Amazon DynamoDB 간 데이터 복사

자습서: Amazon DynamoDB 및 Apache Hive 작업에서는 고유 Hive 테이블에서 외부 DynamoDB 테이블로 데이터를 복사한 다음 외부 DynamoDB 테이블을 쿼리했습니다. 이 테이블은 Hive 외부에 존재하기 때문에 외부 테이블입니다. 여기로 매핑되는 Hive 테이블을 삭제하더라도 DynamoDB 내 테이블은 영향을 받지 않습니다.

Hive는 DynamoDB 테이블, Amazon S3 버킷, 고유 Hive 테이블 및 Hadoop 분산 파일 시스템(HDFS) 사이에서 데이터를 복사하는 데 매우 유용한 솔루션입니다. 이 단원에서는 이러한 작업의 예를 제공합니다.

주제

- DynamoDB와 고유 Hive 테이블 간 데이터 복사
- DynamoDB와 Amazon S3 간 데이터 복사
- DynamoDB와 HDFS 간 데이터 복사
- 데이터 압축 사용
- 인쇄할 수 없는 UTF-8 문자 데이터 읽기

### DynamoDB와 고유 Hive 테이블 간 데이터 복사

DynamoDB 테이블에 데이터가 있을 경우 이 데이터를 고유 Hive 테이블로 복사할 수 있습니다. 그러면 복사 시점의 데이터 스냅샷이 제공됩니다.

수많은 HiveQL 쿼리를 수행해야 하지만 DynamoDB에서 프로비저닝된 처리량을 소비하지 않으려면 이렇게 할 수 있습니다. 고유 Hive 테이블의 데이터는 DynamoDB 내 데이터의 복사본이고 '실시간' 데이터가 아니므로 쿼리에서 최신 데이터를 기대할 수 없습니다.

> **Note**
>
> 이 단원의 예제는 자습서: Amazon DynamoDB 및 Apache Hive 작업의 단계를 완료했고 DynamoDB에 ddb_features라는 외부 테이블이 있다는 전제하에 작성되었습니다.

Example DynamoDB에서 고유 Hive 테이블로

다음과 같이 고유 Hive 테이블을 생성하여 ddb_features로부터 데이터를 채울 수 있습니다

```
CREATE TABLE features_snapshot AS
SELECT * FROM ddb_features;
```

그런 다음 언제라도 데이터를 새로 고칠 수 있습니다.

```
INSERT OVERWRITE TABLE features_snapshot
SELECT * FROM ddb_features;
```

이들 예제에서는 하위 쿼리 `SELECT * FROM ddb_features`가 ddb_features에서 모든 데이터를 검색합니다. 데이터의 하위 집합만 복사하려면 하위 쿼리에 `WHERE` 절을 사용할 수 있습니다.

다음 예제는 호수 및 산정 속성을 일부만 포함하는 고유 Hive 테이블을 생성합니다.

```
CREATE TABLE lakes_and_summits AS
SELECT feature_name, feature_class, state_alpha
FROM ddb_features
WHERE feature_class IN ('Lake','Summit');
```

Example 고유 Hive 테이블에서 DynamoDB로

다음 HiveQL 문을 사용하여 고유 Hive 테이블에서 ddb_features로 데이터를 복사합니다.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM features_snapshot;
```

# DynamoDB와 Amazon S3 간 데이터 복사

DynamoDB 테이블에 데이터가 있는 경우 Hive를 사용하여 이 데이터를 Amazon S3 버킷으로 복사할 수 있습니다.

DynamoDB 테이블에서 데이터 아카이브를 생성하려는 경우 이렇게 할 수 있습니다. 예를 들어 DynamoDB 내 테스트 데이터의 기준 집합으로 작업해야 하는 테스트 환경을 가정하겠습니다. 기준 데이터를 Amazon S3 버킷으로 복사한 다음 테스트를 실행할 수 있습니다. 나중에 Amazon S3 버킷에서 DynamoDB로 기준 데이터를 복원하여 테스트 환경을 재설정할 수 있습니다.

자습서: Amazon DynamoDB 및 Apache Hive 작업을 완료했다면 이미 Amazon EMR 로그를 포함하는 Amazon S3 버킷이 있습니다. 버킷의 루트 경로를 알고 있다면 이 단원의 예제에서 이 버킷을 사용할 수 있습니다.

1. https://console.aws.amazon.com/emr에서 Amazon EMR 콘솔을 엽니다.
2. Name에 대해 클러스터를 선택합니다.
3. URI은 Configuration Details(구성 세부 정보)의 Log URI(로그 URI)에 나열됩니다.
4. 버킷의 루트 경로를 기록합니다. 이름 지정 규칙은 다음과 같습니다.

   ```
   s3://aws-logs-accountID-region
   ```

   여기서 *accountID*는 AWS 계정 ID이고, region은 버킷의 AWS 리전입니다.

> **Note**
>
> 다음 예제에서는 이 예제와 같이 버킷 내 하위 경로를 사용합니다.
> `s3://aws-logs-123456789012-us-west-2/hive-test`

다음 절차는 자습서 내 단계를 완료했고 DynamoDB에 ddb_features라는 외부 테이블이 있다는 전제하에 작성되었습니다.

주제

- Hive 기본 형식을 사용하여 데이터 복사
- 사용자 지정 형식을 사용하여 데이터 복사
- 열 매핑 없이 데이터 복사
- Amazon S3에서 데이터 보기

Hive 기본 형식을 사용하여 데이터 복사

Example DynamoDB에서 Amazon S3로

`INSERT OVERWRITE` 문을 사용하여 Amazon S3에 직접 작성합니다.

```
INSERT OVERWRITE DIRECTORY 's3://aws-logs-123456789012-us-west-2/hive-test'
SELECT * FROM ddb_features;
```

Amazon S3 내 데이터 파일은 다음과 같습니다.

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

각 필드는 SOH 문자(제목 시작 부분, 0x01)로 구분됩니다. 파일에서는 SOH가 **^A**로 나타납니다.

Example Amazon S3에서 DynamoDB로

1. Amazon S3 내의 형식이 지정되지 않은 데이터를 가리키는 외부 테이블을 생성합니다.

   ```
   CREATE EXTERNAL TABLE s3_features_unformatted
       (feature_id       BIGINT,
       feature_name      STRING ,
       feature_class     STRING ,
       state_alpha       STRING,
       prim_lat_dec      DOUBLE ,
       prim_long_dec     DOUBLE ,
       elev_in_ft        BIGINT)
   LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
   ```

2. DynamoDB에 데이터를 복사합니다.

   ```
   INSERT OVERWRITE TABLE ddb_features
   SELECT * FROM s3_features_unformatted;
   ```

### 사용자 지정 형식을 사용하여 데이터 복사

자체 필드 구분자 문자를 지정하려면 Amazon S3 버킷으로 매핑되는 외부 테이블을 생성할 수 있습니다. 쉼표로 분리된 값(CSV)을 갖는 데이터 파일을 생성할 때 이 기법을 사용할 수 있습니다.

#### Example DynamoDB에서 Amazon S3로

1. Amazon S3로 매핑되는 Hive 외부 테이블을 생성합니다. 이때 데이터 형식이 DynamoDB 외부 테이블과 일치해야 합니다.

```
CREATE EXTERNAL TABLE s3_features_csv
    (feature_id       BIGINT,
    feature_name      STRING,
    feature_class     STRING,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE,
    prim_long_dec     DOUBLE,
    elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
```

2. DynamoDB에서 데이터를 복사합니다.

```
INSERT OVERWRITE TABLE s3_features_csv
SELECT * FROM ddb_features;
```

Amazon S3 내 데이터 파일은 다음과 같습니다.

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

#### Example Amazon S3에서 DynamoDB로

단일 HiveQL 문을 사용하여 DynamoDB 테이블을 Amazon S3의 데이터로 채울 수 있습니다.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_csv;
```

### 열 매핑 없이 데이터 복사

데이터 형식 또는 열 매핑을 지정하지 않고 DynamoDB의 데이터를 원본 형식으로 복사하여 Amazon S3에 쓸 수 있습니다. 이 방법을 사용하여 DynamoDB 데이터의 아카이브를 생성하고 Amazon S3에 저장할 수 있습니다.

#### Example DynamoDB에서 Amazon S3로

1. DynamoDB 테이블과 연결된 외부 테이블을 생성합니다. (이 HiveQL 문에는 `dynamodb.column.mapping`이 없습니다.)

```
CREATE EXTERNAL TABLE ddb_features_no_mapping
    (item MAP<STRING, STRING>)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "Features");
```

2. Amazon S3 버킷과 연결된 또 하나의 외부 테이블을 생성합니다.

```
CREATE EXTERNAL TABLE s3_features_no_mapping
    (item MAP<STRING, STRING>)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
```

3. DynamoDB에서 Amazon S3로 데이터를 복사합니다.

```
INSERT OVERWRITE TABLE s3_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

Amazon S3 내 데이터 파일은 다음과 같습니다.

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
```

```
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

각 필드는 STX 문자(텍스트의 시작 부분, 0x02)로 시작하고 ETX 문자(텍스트의 끝부분, 0x03)로 끝납니다. 파일에서는 STX가 **^B**로 나타나고 ETX가 **^C**로 나타납니다.

Example Amazon S3에서 DynamoDB로

단일 HiveQL 문을 사용하여 DynamoDB 테이블을 Amazon S3의 데이터로 채울 수 있습니다.

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM s3_features_no_mapping;
```

Amazon S3에서 데이터 보기

SSH를 사용하여 리더 노드에 연결하는 경우 AWS Command Line Interface(AWS CLI)를 사용하여 Hive가 Amazon S3에 쓴 데이터에 액세스할 수 있습니다.

다음 단계는 이 단원에서 설명한 절차 중 하나를 사용하여 DynamoDB에서 Amazon S3로 데이터를 복사했다는 전제하에 작성되었습니다.

1. 현재 위치가 Hive 명령 프롬프트라면 Linux 명령 프롬프트로 전환합니다.

   ```
   hive> exit;
   ```

2. Amazon S3 버킷 내 hive-test 디렉터리의 내용을 나열합니다. (Hive가 DynamoDB에서 데이터를 복사하여 저장한 디렉터리입니다.)

   ```
   aws s3 ls s3://aws-logs-123456789012-us-west-2/hive-test/
   ```

   응답이 다음과 비슷할 것입니다.

   ```
   2016-11-01 23:19:54 81983 000000_0
   ```

   파일 이름(000000_0)은 시스템에 의해 생성됩니다.

3. (선택 사항) 데이터 파일을 Amazon S3에서 리더 노드의 로컬 파일 시스템으로 복사할 수 있습니다. 그런 다음 표준 Linux 명령줄 유틸리티를 사용하여 파일 데이터에 대한 작업을 수행할 수 있습니다.

   ```
   aws s3 cp s3://aws-logs-123456789012-us-west-2/hive-test/000000_0 .
   ```

응답이 다음과 비슷할 것입니다.

```
download: s3://aws-logs-123456789012-us-west-2/hive-test/000000_0
to ./000000_0
```

> **Note**
>
> 리더 노드의 로컬 파일 시스템은 용량이 제한적입니다. 로컬 파일 시스템의 가용 용량보다 큰 파일에는 이 명령을 사용하지 마세요.

## DynamoDB와 HDFS 간 데이터 복사

DynamoDB 테이블에 데이터가 있을 경우 Hive를 사용하여 이 데이터를 Hadoop 분산 파일 시스템(HDFS)으로 복사할 수 있습니다.

DynamoDB의 데이터가 필요한 MapReduce 작업을 실행하는 경우 이렇게 할 수 있습니다. DynamoDB에서 HDFS로 데이터를 복사하는 경우 Hadoop이 Amazon EMR 클러스터에서 사용 가능한 모든 노드를 병렬로 사용하여 이 작업을 처리할 수 있습니다. MapReduce 작업이 완료되면 HDFS에서 DDB로 결과를 쓸 수 있습니다.

다음 예제에서는 Hive가 다음 HDFS 디렉터리에서 데이터를 읽고 씁니다. `/user/hadoop/hive-test`

> **Note**
>
> 이 단원의 예제는 자습서: Amazon DynamoDB 및 Apache Hive 작업의 단계를 완료했고 DynamoDB에 ddb_features라는 외부 테이블이 있다는 전제하에 작성되었습니다.

주제

- Hive 기본 형식을 사용하여 데이터 복사
- 사용자 지정 형식을 사용하여 데이터 복사
- 열 매핑 없이 데이터 복사
- HDFS의 데이터 액세스

Hive 기본 형식을 사용하여 데이터 복사

Example DynamoDB에서 HDFS로

`INSERT OVERWRITE` 문을 사용하여 HDFS에 직접 쓰기를 수행합니다.

```
INSERT OVERWRITE DIRECTORY 'hdfs:///user/hadoop/hive-test'
SELECT * FROM ddb_features;
```

HDFS 내 데이터 파일은 다음과 같습니다.

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

각 필드는 SOH 문자(제목 시작 부분, 0x01)로 구분됩니다. 파일에서는 SOH가 **^A**로 나타납니다.

Example HDFS에서 DynamoDB로

1. HDFS 내의 형식이 지정되지 않은 데이터로 매핑되는 외부 테이블을 생성합니다.

   ```
   CREATE EXTERNAL TABLE hdfs_features_unformatted
       (feature_id       BIGINT,
       feature_name      STRING ,
       feature_class     STRING ,
       state_alpha       STRING,
       prim_lat_dec      DOUBLE ,
       prim_long_dec     DOUBLE ,
       elev_in_ft        BIGINT)
   LOCATION 'hdfs:///user/hadoop/hive-test';
   ```

2. DynamoDB에 데이터를 복사합니다.

   ```
   INSERT OVERWRITE TABLE ddb_features
   SELECT * FROM hdfs_features_unformatted;
   ```

### 사용자 지정 형식을 사용하여 데이터 복사

다른 필드 구분자 문자를 지정하려면 HDFS 디렉터리로 매핑되는 외부 테이블을 생성할 수 있습니다. 쉼표로 분리된 값(CSV)을 갖는 데이터 파일을 생성할 때 이 기법을 사용할 수 있습니다.

#### Example DynamoDB에서 HDFS로

1. HDFS로 매핑되는 Hive 외부 테이블을 생성합니다. 이때 데이터 형식이 DynamoDB 외부 테이블과 일치해야 합니다.

```
CREATE EXTERNAL TABLE hdfs_features_csv
    (feature_id       BIGINT,
    feature_name      STRING ,
    feature_class     STRING ,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
    elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 'hdfs:///user/hadoop/hive-test';
```

2. DynamoDB에서 데이터를 복사합니다.

```
INSERT OVERWRITE TABLE hdfs_features_csv
SELECT * FROM ddb_features;
```

HDFS 내 데이터 파일은 다음과 같습니다.

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

#### Example HDFS에서 DynamoDB로

단일 HiveQL 문을 사용하여 DynamoDB 테이블을 HDFS의 데이터로 채울 수 있습니다.

```
INSERT OVERWRITE TABLE ddb_features
```

```
SELECT * FROM hdfs_features_csv;
```

### 열 매핑 없이 데이터 복사

데이터 형식 또는 열 매핑을 지정하지 않고 DynamoDB의 데이터를 원본 형식으로 복사하여 HDFS에 쓸 수 있습니다. 이 방법을 사용하여 DynamoDB 데이터의 아카이브를 생성하고 HDFS에 저장할 수 있습니다.

> **Note**
>
> DynamoDB 테이블에 맵, 목록, 부울 또는 Null 형식의 속성이 포함된 경우 이것이 Hive를 사용하여 DynamoDB에서 HDFS로 데이터를 복사할 수 있는 유일한 방법입니다.

#### Example DynamoDB에서 HDFS로

1. DynamoDB 테이블과 연결된 외부 테이블을 생성합니다. (이 HiveQL 문에는 `dynamodb.column.mapping`이 없습니다.)

   ```
   CREATE EXTERNAL TABLE ddb_features_no_mapping
       (item MAP<STRING, STRING>)
   STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
   TBLPROPERTIES ("dynamodb.table.name" = "Features");
   ```

2. HDFS 디렉터리와 연결된 또 하나의 외부 테이블을 생성합니다.

   ```
   CREATE EXTERNAL TABLE hdfs_features_no_mapping
       (item MAP<STRING, STRING>)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   LINES TERMINATED BY '\n'
   LOCATION 'hdfs:///user/hadoop/hive-test';
   ```

3. DynamoDB에서 HDFS로 데이터를 복사합니다.

   ```
   INSERT OVERWRITE TABLE hdfs_features_no_mapping
   SELECT * FROM ddb_features_no_mapping;
   ```

HDFS 내 데이터 파일은 다음과 같습니다.

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

각 필드는 STX 문자(텍스트의 시작 부분, 0x02)로 시작하고 ETX 문자(텍스트의 끝부분, 0x03)로 끝납니다. 파일에서는 STX가 **^B**로 나타나고 ETX가 **^C**로 나타납니다.

Example HDFS에서 DynamoDB로

단일 HiveQL 문을 사용하여 DynamoDB 테이블을 HDFS의 데이터로 채울 수 있습니다.

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM hdfs_features_no_mapping;
```

HDFS의 데이터 액세스

HDFS는 Amazon EMR 클러스터의 모든 노드에서 액세스할 수 있는 분산 파일 시스템입니다. SSH를 사용하여 리더 노드에 연결하는 경우 명령줄 도구를 사용하여 Hive가 HDFS에 쓴 데이터에 액세스할 수 있습니다.

HDFS는 리더 노드의 로컬 파일 시스템과 동일한 것이 아닙니다. 표준 Linux 명령(예: `cat`, `cp`, `mv`, `rm`)을 사용하여 HDFS 내 파일 및 디렉터리에 대한 작업을 수행할 수 없습니다. `hadoop fs` 명령을 사용해야 합니다.

다음 단계는 이 단원에서 설명한 절차 중 하나를 사용하여 DynamoDB에서 HDFS로 데이터를 복사했다는 전제하에 작성되었습니다.

1. 현재 위치가 Hive 명령 프롬프트라면 Linux 명령 프롬프트로 전환합니다.

   ```
   hive> exit;
   ```

2. HDFS 내 /user/hadoop/hive-test 디렉터리의 내용을 나열합니다. (Hive가 DynamoDB에서 데이터를 복사하여 저장한 디렉터리입니다.)

```
hadoop fs -ls /user/hadoop/hive-test
```

응답이 다음과 비슷할 것입니다.

```
Found 1 items
-rw-r--r-- 1 hadoop hadoop 29504 2016-06-08 23:40 /user/hadoop/hive-test/000000_0
```

파일 이름(000000_0)은 시스템에 의해 생성됩니다.

3. 파일 내용을 확인합니다.

```
hadoop fs -cat /user/hadoop/hive-test/000000_0
```

> **Note**
>
> 이 예제에서는 파일 용량이 비교적 작습니다(약 29KB). 용량이 매우 크거나 인쇄 불가능한 문자를 포함하는 파일에 이 명령을 사용할 때는 주의하세요.

4. (선택 사항) 데이터 파일을 HDFS에서 리더 노드의 로컬 파일 시스템으로 복사할 수 있습니다. 그런 다음 표준 Linux 명령줄 유틸리티를 사용하여 파일 데이터에 대한 작업을 수행할 수 있습니다.

```
hadoop fs -get /user/hadoop/hive-test/000000_0
```

이 명령은 파일을 덮어쓰지 않습니다.

> **Note**
>
> 리더 노드의 로컬 파일 시스템은 용량이 제한적입니다. 로컬 파일 시스템의 가용 용량보다 큰 파일에는 이 명령을 사용하지 마세요.

## 데이터 압축 사용

Hive를 사용하여 여러 데이터 원본에서 데이터를 복사하는 경우 즉시 데이터 압축을 요청할 수 있습니다. Hive는 다양한 압축 코덱을 제공합니다. Hive 세션 도중 한 가지를 선택할 수 있습니다. 그러면 데이터가 지정된 형식으로 압축됩니다.

다음 예제에서는 Lempel-Ziv-Oberhumer(LZO) 알고리즘을 사용하여 데이터를 압축합니다.

```
SET hive.exec.compress.output=true;
SET io.seqfile.compression.type=BLOCK;
SET mapred.output.compression.codec = com.hadoop.compression.lzo.LzopCodec;

CREATE EXTERNAL TABLE lzo_compression_table (line STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE lzo_compression_table SELECT *
FROM hiveTableName;
```

Amazon S3의 결과 파일은 시스템에 의해 이름이 생성되고 이름의 끝에 .lzo가 추가됩니다(예: 8d436957-57ba-4af7-840c-96c2fc7bb6f5-000000.lzo).

다음과 같은 압축 코덱을 사용할 수 있습니다.

- org.apache.hadoop.io.compress.GzipCodec
- org.apache.hadoop.io.compress.DefaultCodec
- com.hadoop.compression.lzo.LzoCodec
- com.hadoop.compression.lzo.LzopCodec
- org.apache.hadoop.io.compress.BZip2Codec
- org.apache.hadoop.io.compress.SnappyCodec

## 인쇄할 수 없는 UTF-8 문자 데이터 읽기

인쇄할 수 없는 UTF-8 문자를 읽고 쓰려면 Hive 테이블을 생성할 때 STORED AS SEQUENCEFILE 절을 사용할 수 있습니다. SequenceFile은 Hadoop 이진수 파일 형식입니다. 이 파일을 읽으려면 Hadoop을 사용해야 합니다. 다음 예제에서는 DynamoDB에서 Amazon S3로 데이터를 내보내는 방법을 보여 줍니다. 이 기능을 사용하여 인쇄할 수 없는 UTF-8 인코딩 문자를 처리할 수 있습니다.

```
CREATE EXTERNAL TABLE s3_export(a_col string, b_col bigint, c_col array<string>)
STORED AS SEQUENCEFILE
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE s3_export SELECT *
FROM hiveTableName;
```

# 성능 튜닝

DynamoDB 테이블로 매핑되는 Hive 외부 테이블을 생성할 때 DynamoDB에서 읽기 또는 쓰기 용량이 소비되지 않습니다. 하지만 Hive 테이블에 대한 읽기 및 쓰기 작업(예: `INSERT` 또는 `SELECT`)은 기본 DynamoDB 테이블에 대한 읽기 및 쓰기 작업으로 바로 변환됩니다.

Amazon EMR의 Apache Hive는 DynamoDB 테이블에 대한 I/O 로드를 밸런싱하는 자체 로직을 구현하여 테이블의 프로비저닝된 처리량을 초과할 가능성을 최소화합니다. 각 Hive 쿼리의 끝에서 Amazon EMR은 프로비저닝된 처리량이 초과된 횟수를 포함하여 런타임 지표를 .반환합니다. 이 정보를 DynamoDB 테이블에 대한 CloudWatch 지표와 함께 사용하여 후속 요청에서 성능을 개선할 수 있습니다.

Amazon EMR 콘솔은 클러스터에 대한 기본 모니터링 도구를 제공합니다. 자세한 내용은 Amazon EMR 관리 가이드의 클러스터 보기 및 모니터링 단원을 참조하세요.

또한 Hue, Ganglia, Hadoop 웹 인터페이스와 같은 웹 기반 도구를 사용하여 클러스터 및 Hadoop 작업을 모니터링할 수도 있습니다. 자세한 내용은 Amazon EMR 관리 가이드의 Amazon EMR 클러스터에 호스팅된 웹 인터페이스 보기 단원을 참조하세요.

이 단원에서는 외부 DynamoDB 테이블에서 Hive 작업 성능을 조정하기 위해 취할 수 있는 단계를 설명합니다.

주제

- DynamoDB 프로비저닝 처리량
- 매퍼 수 조정
- 추가 주제

## DynamoDB 프로비저닝 처리량

외부 DynamoDB 테이블에 대해 HiveQL 문을 실행할 때 `DynamoDBStorageHandler` 클래스가 적절한 하위 수준 DynamoDB API 요청을 생성하며 이는 프로비저닝된 처리량을 소비합니다. DynamoDB 테이블의 읽기 또는 쓰기 용량이 충분하지 않을 경우 요청에 병목 현상이 발생하여 HiveQL 성능이 느려집니다. 따라서 테이블의 처리량이 충분하지 확인해야 합니다.

예를 들어 DynamoDB 테이블에 읽기 용량 단위를 100개 프로비저닝하였다고 가정하겠습니다. 그러면 초당 409,600바이트를 읽을 수 있습니다(100 × 4KB 읽기 용량 단위 크기). 이제 테이블 내 데이터가 20GB(21,474,836,480바이트)이고 HiveQL을 사용하여 `SELECT` 문으로 모든 데이터를 선택한다고 가정할 경우, 다음과 같이 쿼리에 소요되는 시간을 추정할 수 있습니다.

21,474,836,480 / 409,600 = 52,429초 = 14.56시간

이 시나리오에서는 DynamoDB 테이블에 병목 현상이 발생합니다. Hive 처리량이 초당 409,600바이트로 제한되므로 Amazon EMR 노드를 추가해도 소용이 없습니다. SELECT 문에 소요되는 시간을 단축하는 유일한 방법은 DynamoDB 테이블의 프로비저닝된 읽기 용량을 증가하는 것입니다.

비슷한 계산을 통해 DynamoDB 테이블로 매핑된 Hive 외부 테이블로 대량 데이터를 로드할 때 소용되는 시간을 추정할 수 있습니다. 항목당 필요한 총 쓰기 용량 단위 수(1KB 미만 = 1, 1~2KB = 2 등)를 결정하고 여기에 로드할 항목 수를 곱합니다. 그러면 필요한 쓰기 용량 단위 수를 알 수 있습니다. 이 수를 초당 할당된 쓰기 용량 단위 수로 나눕니다. 그러면 테이블에 로드하는 데 소요되는 시간(초)가 구해집니다.

테이블에 대한 CloudWatch 지표를 주기적으로 모니터링해야 합니다. DynamoDB 콘솔에서 빠르게 확인하려면 테이블을 선택하고 Metrics(지표) 탭을 선택합니다. 여기에서 소비된 읽기 및 쓰기 용량 단위와 병목 현상이 발생한 읽기 및 쓰기 요청을 볼 수 있습니다.

읽기 용량

Amazon EMR은 테이블의 프로비저닝된 처리량 설정에 따라 DynamoDB 테이블에 대한 요청 로드를 관리합니다. 하지만 작업 출력에서 `ProvisionedThroughputExceeded` 메시지가 다수 확인될 경우 기본 읽기 속도를 조정할 수 있습니다. 이를 위해 `dynamodb.throughput.read.percent` 구성 변수를 수정할 수 있습니다. Hive 명령 프롬프트에서 SET 명령을 사용하여 이 변수를 설정할 수 있습니다.

```
SET dynamodb.throughput.read.percent=1.0;
```

이 변수는 현재 Hive 세션 동안만 지속됩니다. Hive를 종료했다 나중에 다시 시작할 경우 `dynamodb.throughput.read.percent`가 기본값으로 되돌아갑니다.

`dynamodb.throughput.read.percent` 값은 `0.1`부터 `1.5`까지 가능합니다. `0.5`는 기본 읽기 속도입니다. 즉, Hive가 테이블 읽기 용량의 절반을 소비하려고 시도합니다. 값을 `0.5`보다 높게 설정할 경우 Hive가 요청 속도를 높이고 `0.5`보다 낮게 설정할 경우 요청 속도를 낮춥니다. (실제 읽기 속도는 DynamoDB 테이블의 키 분포가 균일한지 여부와 같은 요소에 따라 달라집니다.)

Hive가 빈번히 테이블의 할당 읽기 용량을 소진하거나 읽기 요청에서 병목 현상이 너무 자주 발생할 경우 `dynamodb.throughput.read.percent`를 `0.5`보다 낮게 설정해 보세요. 테이블 읽기 용량이 충분하고 HiveQL 작업의 응답성을 높이고자 할 경우 이 값을 `0.5`보다 높게 설정할 수 있습니다.

### 쓰기 용량

Amazon EMR은 테이블의 프로비저닝된 처리량 설정에 따라 DynamoDB 테이블에 대한 요청 로드를 관리합니다. 하지만 작업 출력에서 `ProvisionedThroughputExceeded` 메시지가 다수 확인될 경우 기본 쓰기 속도를 조정할 수 있습니다. 이를 위해 `dynamodb.throughput.write.percent` 구성 변수를 수정할 수 있습니다. Hive 명령 프롬프트에서 `SET` 명령을 사용하여 이 변수를 설정할 수 있습니다.

```
SET dynamodb.throughput.write.percent=1.0;
```

이 변수는 현재 Hive 세션 동안만 지속됩니다. Hive를 종료했다 나중에 다시 시작할 경우 `dynamodb.throughput.write.percent`가 기본값으로 되돌아갑니다.

`dynamodb.throughput.write.percent` 값은 `0.1`부터 `1.5`까지 가능합니다. `0.5`는 기본 쓰기 속도입니다. 즉, Hive가 테이블 쓰기 용량의 절반을 소비하려고 시도합니다. 값을 `0.5`보다 높게 설정할 경우 Hive가 쓰기 요청 속도를 높이고 `0.5`보다 낮게 설정할 경우 쓰기 요청 속도를 낮춥니다. (실제 쓰기 속도는 DynamoDB 테이블의 키 분포가 균일한지 여부와 같은 요소에 따라 달라집니다.)

Hive가 빈번히 테이블의 할당 읽기 용량을 소진하거나 읽기 요청에서 병목 현상이 너무 자주 발생할 경우 `dynamodb.throughput.write.percent`를 `0.5`보다 낮게 설정해 보세요. 테이블의 용량이 충분하고 HiveQL 작업의 응답성을 높이고자 할 경우 이 값을 `0.5`보다 높게 설정할 수 있습니다.

Hive를 사용하여 DynamoDB에 데이터를 쓸 때는 쓰기 용량 단위 수가 클러스터의 매퍼 수보다 큰지 확인해야 합니다. 예를 들어, 10개 m1.xlarge 노드로 구성된 Amazon EMR 클러스터가 있다고 가정해 보겠습니다. m1.xlarge 노드 유형은 8개의 매퍼 작업을 제공하므로 클러스터 내 매퍼는 총 80개(10 × 8)가 됩니다. DynamoDB 테이블의 쓰기 용량 단위가 80 미만인 경우 Hive 쓰기 작업 1개가 테이블의 쓰기 처리량을 모두 소비할 수 있습니다.

Amazon EMR 노드 유형의 매퍼 수를 확인하려면 Amazon EMR 개발자 안내서의 태스크 구성 단원을 참조하세요.

매퍼에 대한 자세한 내용은 매퍼 수 조정 단원을 참조하세요.

## 매퍼 수 조정

Hive는 Hadoop 작업을 시작하면 작업은 하나 이상의 매퍼 작업에 의해 처리됩니다. DynamoDB 테이블의 처리량이 충분하다는 전제하에, 클러스터 내 매퍼 수를 수정하여 성능을 개선할 수도 있습니다.

> **Note**
>
> Hadoop 작업에 사용되는 매퍼 작업 수는 Hadoop이 데이터를 논리 블록으로 분할하는 입력 분할에 의해 영향을 받습니다. Hadoop이 충분한 입력 분할을 수행하지 않을 경우 쓰기 작업이 DynamoDB 테이블에서 사용 가능한 쓰기 처리량을 모두 소비하지 못할 수 있습니다.

### 매퍼 수 증가

Amazon EMR의 각 매퍼는 최대 읽기 속도가 초당 1MiB입니다. 클러스터 내 매퍼 수는 클러스터 내 노드 크기에 따라 결정됩니다. (노드 크기 및 노드당 매퍼 수에 대한 정보는 Amazon EMR 개발자 안내서의 태스크 구성 단원을 참조하세요.)

DynamoDB 테이블의 읽기 처리량이 충분할 경우 다음 중 한 방법으로 매퍼 수를 늘릴 수 있습니다.

- 클러스터 내 노드의 크기를 증가시킵니다. 예를 들어 클러스터가 m1.large 노드(노드당 매퍼 3개)를 사용하는 경우 m1.xlarge 노드(노드당 매퍼 8개)로 업그레이드할 수 있습니다.
- 클러스터 내 노드 수를 증가시킵니다. 예를 들어 클러스터에 m1.xlarge 노드가 3개일 경우 총 24개 매퍼를 사용할 수 있습니다. 동일한 노드 유형에서 클러스터 크기를 두 배로 늘리면 매퍼는 48개로 증가합니다.

AWS Management Console을 사용하여 클러스터 내 노드 크기 또는 수를 관리할 수 있습니다. (이 설정이 적용되려면 클러스터를 재시작해야 합니다.)

매퍼 수를 늘리는 또 한 가지 방법은 `mapred.tasktracker.map.tasks.maximum` Hadoop 구성 파라미터를 수정하는 것입니다. (이것은 Hive 파라미터가 아니라 Hadoop 파라미터입니다. 명령 프롬프트에서 대화형으로 이 파라미터를 수정할 수 없습니다.). `mapred.tasktracker.map.tasks.maximum` 값을 높일 경우 노드 크기 또는 수를 증가시키지 않고 매퍼 수를 늘릴 수 있습니다. 하지만 이 값을 너무 높게 설정할 경우 클러스터 노드 메모리가 부족해질 수 있습니다.

`mapred.tasktracker.map.tasks.maximum` 값은 처음 Amazon EMR 클러스터를 시작할 때 부트스트랩 작업으로 설정합니다. 자세한 내용은 Amazon EMR 관리 가이드드 (선택 사항) 부트스트랩 작업을 생성하여 추가 소프트웨어 설치 단원을 참조하세요.

### 매퍼 수 감소

`SELECT` 문을 사용하여 DynamoDB로 매핑되는 외부 Hive 테이블에서 데이터를 선택할 경우 Hadoop 작업은 필요에 따라 클러스터 내 최대 매퍼 수까지 작업을 사용할 수 있습니다. 이 시나리오에서는 장

시간 실행되는 Hive 쿼리가 DynamoDB 테이블의 프로비저닝된 읽기 용량을 모두 소비하여 다른 사용자에게 부정적인 영향을 미칠 수 있습니다.

dynamodb.max.map.tasks 파라미터를 사용하여 매퍼 작업 수 상한을 설정할 수 있습니다.

```
SET dynamodb.max.map.tasks=1
```

이 값은 1보다 크거나 같아야 합니다. Hive가 쿼리를 처리할 때 Hadoop 작업이 DynamoDB 테이블에서 데이터를 읽으면서 dynamodb.max.map.tasks보다 많은 작업을 사용하지 않습니다.

## 추가 주제

다음은 Hive가 DynamoDB에 액세스하기 위해 사용하는 애플리케이션을 조정하기 위해 추가로 사용할 수 있는 몇 가지 방법입니다.

### 재시도 기간

기본적으로 Hive는 DynamoDB에서 2분 이내에 결과가 반환되지 않으면 Hadoop 작업을 재실행합니다. dynamodb.retry.duration 파라미터를 수정하여 이 간격을 조정할 수 있습니다.

```
SET dynamodb.retry.duration=2;
```

이 값은 0이 아닌 정수여야 하며, 재시도 주기를 분 단위로 지정합니다. dynamodb.retry.duration 기본값은 2(분)입니다.

### 병렬 데이터 요청

한 명 이상의 사용자 또는 하나 이상의 애플리케이션에서 단일 테이블로 데이터 요청이 다수 이루어지면 읽기 할당된 처리량이 한 번에 소비되어 성능이 느려질 수 있습니다.

### 프로세스 기간

DynamoDB의 데이터 일관성은 각 노드의 읽기 및 쓰기 작업 순서에 따라 달라집니다. Hive 쿼리가 진행 중일 때는 다른 애플리케이션이 새로운 데이터를 DynamoDB 테이블에 로드하거나 기존 데이터를 변경 또는 삭제하기도 합니다. 이 경우 쿼리 실행 중 데이터 변경 사항은 Hive 쿼리 결과에 반영되지 않습니다.

### 요청 시간

DynamoDB 테이블에 대한 수요가 낮은 시간에 Hive 쿼리가 DynamoDB 테이블에 액세스하도록 일정을 조정하면 성능이 향상됩니다. 예를 들어 애플리케이션 사용자 대부분이 샌프란시스코에 거주한다

면 대다수가 잠드는 시간인 새벽 4시(PST)에 DynamoDB 데이터베이스의 레코드 업데이트 없이 1일 데이터를 내보내는 것도 좋은 방법입니다.

# Amazon S3와 DynamoDB 통합

Amazon DynamoDB 가져오기 및 내보내기 기능을 사용하면 코드를 작성하지 않고도 Amazon S3와 DynamoDB 테이블 간에 데이터를 간단하고 효율적으로 이동할 수 있습니다.

DynamoDB 가져오기 및 내보내기 기능을 통해 DynamoDB 테이블 계정을 이동, 변환 및 복사할 수 있습니다. S3 소스에서 가져올 수 있으며, DynamoDB 테이블 데이터를 Amazon S3로 내보내고 Athena, Amazon SageMaker AI 및 AWS Lake Formation 등의 AWS 서비스를 사용하여 데이터를 분석하고 유용한 인사이트를 추출할 수 있습니다. 또한 데이터를 새로운 DynamoDB 테이블로 직접 가져와서 대규모로 10밀리초 미만의 성능으로 새 애플리케이션을 구축하고, 테이블과 계정 간의 데이터 공유를 용이하게 하며, 재해 복구 및 비즈니스 연속성 계획을 간소화할 수 있습니다.

주제

- Amazon S3에서 DynamoDB 데이터 가져오기: 작동 방식
- Amazon S3로 DynamoDB 데이터 내보내기: 작동 방식

## Amazon S3에서 DynamoDB 데이터 가져오기: 작동 방식

데이터를 DynamoDB로 가져오려면 데이터가 CSV, DynamoDB JSON 또는 Amazon Ion 형식으로 Amazon S3 버킷에 있어야 합니다. 데이터는 ZSTD 또는 GZIP 형식으로 압축하거나 압축되지 않은 형식으로 직접 가져올 수 있습니다. 소스 데이터는 단일 Amazon S3 객체이거나 동일한 접두사를 사용하는 여러 Amazon S3 객체일 수 있습니다.

데이터는 가져오기 요청을 시작할 때 생성되는 새 DynamoDB 테이블로 가져오게 됩니다. 보조 인덱스를 사용하여 이 테이블을 만든 다음, 가져오기가 완료되는 즉시 모든 기본 및 보조 인덱스에서 데이터를 쿼리하고 업데이트할 수 있습니다. 가져오기가 완료된 후 전역 테이블 복제본을 추가할 수도 있습니다.

> **Note**
>
> Amazon S3 가져오기 프로세스 중에 DynamoDB는 가져올 새 대상 테이블을 생성합니다. 기존 테이블로의 가져오기는 현재 이 기능에서 지원되지 않습니다.

Amazon S3에서 가져오기는 새 테이블의 쓰기 용량을 소비하지 않으므로 DynamoDB로 데이터를 가져오기 위해 추가 용량을 프로비저닝할 필요가 없습니다. 데이터 가져오기 요금은 가져오기의 결과로 처리되는 Amazon S3의 압축되지 않은 소스 데이터 크기를 기준으로 합니다. 처리되었지만 소스 데이터의 형식 또는 기타 불일치로 인해 테이블에 로드되지 않는 항목도 가져오기 프로세스의 일부로 비용이 청구됩니다. 자세한 정보는 Amazon DynamoDB 요금을 참조하세요.

해당 버킷에서 읽을 수 있는 올바른 권한이 있으면 다른 계정이 소유한 Amazon S3 버킷에서 데이터를 가져올 수 있습니다. 새 테이블은 소스 Amazon S3 버킷과 다른 리전에 있을 수도 있습니다. 자세한 내용은 Amazon Simple Storage Service 설정 및 권한을 참조하세요.

가져오기 시간은 Amazon S3의 데이터 특성과 직접적인 관련이 있습니다. 여기에는 데이터 크기, 데이터 형식, 압축 체계, 데이터 배포의 균일성, Amazon S3 객체 수 및 기타 관련 변수가 포함됩니다. 특히 키가 균일하게 분포된 데이터 세트는 왜곡된 데이터 세트보다 가져오기 속도가 더 빠릅니다. 예를 들어 보조 인덱스의 키가 파티셔닝에 그해의 월을 사용하고 있고 모든 데이터가 12월의 데이터인 경우 이 데이터를 가져오는 데 훨씬 더 오래 걸릴 수 있습니다.

키와 연결된 속성은 기본 테이블에서 고유해야 합니다. 고유하지 않은 키가 있는 경우 가져오기는 마지막 덮어쓰기만 남을 때까지 연결된 항목을 덮어씁니다. 예를 들어 프라이머리 키가 월이고 여러 항목이 9월로 설정된 경우 각각의 새 항목은 이전에 작성된 항목을 덮어쓰며 프라이머리 키 '월'이 9월로 설정된 항목 하나만 남게 됩니다. 이러한 경우 가져오기 테이블 설명에서 처리된 항목 수가 대상 테이블의 항목 수와 일치하지 않습니다.

AWS CloudTrail은 테이블 가져오기의 콘솔 및 API 작업을 모두 로깅합니다. 자세한 내용은 AWS CloudTrail을 사용하여 DynamoDB 작업 로깅 섹션을 참조하세요.

다음 동영상은 Amazon S3에서 DynamoDB로 직접 가져오는 방법을 소개합니다.

Amazon S3에서 가져오기

주제

- DynamoDB에서 테이블 가져오기 요청
- DynamoDB용 Amazon S3 가져오기 형식
- 가져오기 형식 할당량 및 유효성 검사
- Amazon S3에서 DynamoDB로 가져오기 모범 사례

# DynamoDB에서 테이블 가져오기 요청

DynamoDB 가져오기를 사용하면 Amazon S3 버킷에서 새 DynamoDB 테이블로 데이터를 가져올 수 있습니다. DynamoDB 콘솔, CLI, CloudFormation 또는 DynamoDB API를 사용하여 테이블 가져오기를 요청할 수 있습니다.

AWS CLI를 사용하려면 먼저 구성을 해야 합니다. 자세한 내용은 DynamoDB 액세스 섹션을 참조하세요.

> **Note**
>
> - 테이블 가져오기 기능은 Amazon S3 및 CloudWatch와 같은 다양한 AWS 서비스와 상호 작용합니다. 가져오기를 시작하기 전에 가져오기 API를 호출하는 사용자 또는 역할에 해당 기능에서 사용되는 모든 서비스 및 리소스에 대한 권한이 있는지 확인합니다.
> - 가져오기가 진행 중인 동안에는 Amazon S3 객체를 수정하지 마세요. 수정하면 작업이 실패하거나 취소될 수 있습니다.
>
> 오류 및 문제 해결에 대한 자세한 내용은 가져오기 형식 할당량 및 유효성 검사 섹션을 참조하세요.

주제


- IAM 권한 설정
- AWS Management Console을 사용하여 가져오기 요청
- AWS Management Console에서 이전 가져오기에 대한 세부 정보 확인
- AWS CLI을 사용하여 가져오기 요청
- AWS CLI에서 이전 가져오기에 대한 세부 정보 확인


## IAM 권한 설정

읽기 권한이 있는 Amazon S3 버킷에서 데이터를 가져올 수 있습니다. 소스 버킷이 소스 테이블과 같은 리전에 있거나 소유자가 동일하지 않아도 됩니다. AWS Identity and Access Management(IAM)에는 소스 Amazon S3 버킷에 대한 관련 작업과 디버깅 정보를 제공하는 데 필요한 CloudWatch 권한이 포함되어야 합니다. 예제 정책은 다음과 같습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDynamoDBImportAction",
      "Effect": "Allow",
      "Action": [
        "dynamodb:ImportTable",
        "dynamodb:DescribeImport"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table*"
    },
    {
      "Sid": "AllowS3Access",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::your-bucket/*",
        "arn:aws:s3:::your-bucket"
      ]
    },
    {
      "Sid": "AllowCloudwatchAccess",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:PutRetentionPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group/aws-dynamodb/*"
    },
    {
      "Sid": "AllowDynamoDBListImports",
      "Effect": "Allow",
      "Action": "dynamodb:ListImports",
```

```
      "Resource": "*"
    }
  ]
}
```

Amazon S3 권한

다른 계정이 소유한 Amazon S3 버킷 소스에서 가져오기를 시작할 때는 해당 역할 또는 사용자에게 Amazon S3 객체에 대한 액세스 권한이 있는지 확인합니다. Amazon S3 `GetObject` 명령을 실행하고 보안 인증 정보를 사용하여 확인할 수 있습니다. API를 사용할 때 Amazon S3 버킷 소유자 파라미터는 기본적으로 현재 사용자의 계정 ID로 설정됩니다. 교차 계정 가져오기의 경우 이 파라미터가 버킷 소유자의 계정 ID로 올바르게 채워졌는지 확인합니다. 다음 코드는 소스 계정에 있는 Amazon S3 버킷 정책의 예입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {"Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

AWS Key Management Service

가져올 새 테이블을 생성할 때 DynamoDB가 소유하지 않은 저장 데이터 암호화 키를 선택하는 경우 고객 관리형 키로 암호화된 DynamoDB 테이블을 작동하는 데 필요한 AWS KMS 권한을 제공해야 합니다. 자세한 내용은 AWS KMS 키 사용 권한 부여를 참조하세요. Amazon S3 객체가 서버 측 암호화 KMS(SSE-KMS)로 암호화된 경우 가져오기를 시작하는 역할 또는 사용자에게 AWS KMS 키를 사용하

여 암호를 해독할 수 있는 액세스 권한이 있는지 확인합니다. 이 기능은 고객 제공 암호화 키(SSE-C)로 암호화된 Amazon S3 객체는 지원하지 않습니다.

CloudWatch 권한

가져오기를 시작하는 역할 또는 사용자에게는 가져오기와 관련된 로그 그룹 및 로그 스트림에 대한 생성 및 관리 권한이 필요합니다.

AWS Management Console을 사용하여 가져오기 요청

다음은 DynamoDB 콘솔을 사용하여 `MusicCollection`이라는 새 테이블에 기존 데이터를 가져오는 방법을 보여 주는 예입니다.

테이블 가져오기를 요청하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 S3에서 가져오기(Import from S3)를 선택합니다.
3. 표시되는 페이지에서 S3에서 가져오기(Import from S3)를 선택합니다.
4. S3에서 가져오기(Import from S3)를 선택합니다.
5. 소스 S3 URL에 Amazon S3 소스 URL을 입력합니다.

   소스 버킷을 소유하고 있는 경우 S3 찾아보기를 선택하여 해당 버킷을 검색합니다. 또는 `s3://bucket/prefix` 형식으로 버킷의 URL을 입력합니다. `prefix`는 Amazon S3 키 접두사입니다. 이는 가져오려는 Amazon S3 객체 이름 또는 가져오려는 모든 Amazon S3 객체에서 공유하는 키 접두사입니다.

   > **Note**
   >
   > DynamoDB 내보내기 요청과 동일한 접두사를 사용할 수 없습니다. 내보내기 기능은 모든 내보내기에서 폴더 구조와 매니페스트 파일을 생성합니다. 동일한 Amazon S3 경로를 사용하는 경우 오류가 발생합니다.
   > 대신 특정 내보내기의 데이터가 들어 있는 폴더로 가져오기를 진행합니다. 이 경우 올바른 경로의 형식은 `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/Data/`입니다. 여기서 `XXXXXXXX-XXXXXX`는 내보내기 ID입니다.
   > 내보내기 ID는 내보내기 ARN에서 찾을 수 있습니다. 내보내기 ARN의 형식은 `arn:aws:dynamodb:<Region>:<AccountID>:table/<TableName>/export/<XXXXXXXX-XXXXXX>`입니다. 예를 들어 `arn:aws:dynamodb:us-`

*east-1*:*123456789012*:table/*ProductCatalog*/export/*01234567890123-a1b2c3d4*입니다.

6. S3 버킷 소유자(S3 bucket owner)인지 여부를 지정합니다. 다른 계정이 소스 버킷을 소유한 경우 A different AWS account(다른 AWS 계정)을 선택합니다. 그런 다음 버킷 소유자의 계정 ID를 입력합니다.
7. 가져오기 파일 압축(Import file compression)에서 압축 없음(No compression), GZIP 또는 ZSTD를 적절하게 선택합니다.
8. 적절한 가져오기 파일 형식을 선택합니다. 옵션은 DynamoDB JSON, Amazon Ion 또는 CSV입니다. CSV를 선택하면 CSV 헤더(CSV header)와 CSV 구분 기호 문자(CSV delimiter character)의 두 가지 추가 옵션이 제공됩니다.

   CSV 헤더(CSV header)에서 헤더를 파일의 첫 번째 줄에서 가져올지 아니면 사용자 지정할지를 선택합니다. 헤더 사용자 지정(Customize your headers)을 선택하는 경우 가져올 헤더 값을 지정할 수 있습니다. 이 메서드로 지정된 CSV 헤더는 대소문자를 구분하며 대상 테이블의 키를 포함해야 합니다.

   CSV 구분 기호 문자(CSV delimiter character)에서 항목을 구분하는 문자를 설정합니다. 쉼표가 기본적으로 선택됩니다. 사용자 지정 구분 기호 문자를 선택하는 경우 구분 기호는 정규식 패턴과 일치해야 합니다. `[,;:|\t ]`.
9. 다음(Next) 버튼을 클릭하고 데이터를 저장하기 위해 생성될 새 테이블에 대한 옵션을 선택합니다.

   > **Note**
   >
   > 프라이머리 키 및 정렬 키는 파일의 속성과 일치해야 합니다. 그렇지 않으면 가져오기가 실패합니다. 속성은 대소문자를 구분합니다.

10. 다음(Next)을 다시 선택하여 가져오기 옵션을 검토하고 가져오기(Import)를 클릭하여 가져오기 작업을 시작합니다. 먼저 '생성 중(Creating)' 상태로 '테이블(Tables)'에 나열된 새 테이블을 볼 수 있습니다. 현재는 테이블에 액세스할 수 없습니다.
11. 가져오기가 완료되면 상태가 '활성(Active)'으로 표시되고 테이블 사용을 시작할 수 있습니다.

### AWS Management Console에서 이전 가져오기에 대한 세부 정보 확인

이전에 실행한 가져오기 태스크에 대한 정보는 탐색 사이드바에서 S3에서 가져오기(Import from S3)를 클릭한 다음 가져오기(Imports) 탭을 선택하여 확인할 수 있습니다. 가져오기 패널에는 지난 90일간 생

성한 모든 가져오기의 목록이 포함되어 있습니다. 가져오기 탭에 나열된 태스크의 ARN을 선택하면 선택한 고급 구성 설정을 포함하여 해당 가져오기에 대한 정보가 검색됩니다.

AWS CLI을 사용하여 가져오기 요청

다음 예제에서는 prefix라는 접두사가 있는 bucket이라는 S3 버킷에서 target-table이라는 새 테이블로 CSV 형식의 데이터를 가져옵니다.

```
aws dynamodb import-table --s3-bucket-source S3Bucket=bucket,S3KeyPrefix=prefix \
            --input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
            [{"AttributeName":"hk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"hk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}' \
            --input-format-options '{"Csv": {"HeaderList": ["hk", "title", "artist",
 "year_of_release"], "Delimiter": ";"}}'
```

> **Note**
>
> AWS Key Management Service(AWS KMS)에 의해 보호되는 키를 사용하여 가져오기를 암호화하도록 선택하는 경우 해당 키는 대상 Amazon S3 버킷과 동일한 리전에 있어야 합니다.

AWS CLI에서 이전 가져오기에 대한 세부 정보 확인

이전에 실행한 가져오기 태스크에 대한 정보는 `list-imports` 명령을 사용하여 확인할 수 있습니다. 이 명령은 지난 90일간 생성한 모든 가져오기의 목록을 반환합니다. 가져오기 태스크 메타데이터는 90일이 지나면 만료되고 그보다 오래된 작업은 더 이상 이 목록에 없지만, DynamoDB는 Amazon S3 버킷의 객체 또는 가져오기 중에 생성된 테이블을 삭제하지 않습니다.

```
aws dynamodb list-imports
```

특정 가져오기 태스크에 대해 고급 구성 설정을 포함하여 자세한 정보를 검색하려면 `describe-import` 명령을 사용합니다.

```
aws dynamodb describe-import \
    --import-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/exp
```

# DynamoDB용 Amazon S3 가져오기 형식

DynamoDB는 CSV, DynamoDB JSON 및 Amazon Ion의 세 가지 형식으로 데이터를 가져올 수 있습니다.

주제

- CSV
- DynamoDB Json
- Amazon Ion

## CSV

CSV 형식의 파일은 줄 바꿈으로 구분된 여러 항목으로 구성됩니다. 기본적으로 DynamoDB는 가져오기 파일의 첫 줄을 헤더로 해석하고 열을 쉼표로 구분해야 합니다. 파일의 열 수와 일치하는 한, 적용할 헤더를 정의할 수도 있습니다. 헤더를 명시적으로 정의하면 파일의 첫 번째 줄을 값으로 가져오게 됩니다.

> **Note**
>
> CSV 파일에서 가져올 때 기본 테이블 및 보조 인덱스의 해시 범위와 키를 제외한 모든 열을 DynamoDB 문자열로 가져오게 됩니다.

### 큰따옴표 이스케이프

CSV 파일에 있는 모든 큰따옴표 문자는 이스케이프 처리해야 합니다. 다음 예제에서처럼 이스케이프 처리하지 않으면 가져오기가 실패합니다.

```
id,value
"123",Women's Full Lenth Dress
```

두 세트의 큰따옴표로 따옴표를 이스케이프 처리하면 이와 동일한 가져오기가 성공합니다.

```
id,value
"""123""",Women's Full Lenth Dress
```

텍스트를 올바르게 이스케이프 처리하여 가져오면 원본 CSV 파일에서와 동일하게 표시됩니다.

```
id,value
```

```
"123",Women's Full Lenth Dress
```

### DynamoDB Json

DynamoDB JSON 형식의 파일은 여러 항목 객체로 구성될 수 있습니다. 개별 객체는 DynamoDB의 표준 마샬링된 JSON 형식이므로 줄 바꿈이 항목 구분 기호로 사용됩니다. 추가된 기능으로 특정 시점에서의 내보내기는 기본적으로 가져오기 소스로 지원됩니다.

> **Note**
>
> 새 행은 DynamoDB JSON 형식의 파일에 대한 항목 구분자로 사용되며 항목 객체 내에서 사용해서는 안 됩니다.

```
[{
    "Item": {
        "Authors": {
            "SS": ["Author1", "Author2"]
        },
        "Dimensions": {
            "S": "8.5 x 11.0 x 1.5"
        },
        "ISBN": {
            "S": "333-3333333333"
        },
        "Id": {
            "N": "103"
        },
        "InPublication": {
            "BOOL": false
        },
        "PageCount": {
            "N": "600"
        },
        "Price": {
            "N": "2000"
        },
        "ProductCategory": {
            "S": "Book"
        },
        "Title": {
            "S": "Book 103 Title"
```

```
        }
    }
}]
```

> **Note**
>
> 새 행은 DynamoDB JSON 형식의 파일에 대한 항목 구분자로 사용되며 항목 객체 내에서 사용해서는 안 됩니다.

```
[{
    "Item": {
        "Authors": {
            "SS": ["Author1", "Author2"]
        },
        "Dimensions": {
            "S": "8.5 x 11.0 x 1.5"
        },
        "ISBN": {
            "S": "333-3333333333"
        },
        "Id": {
            "N": "103"
        },
        "InPublication": {
            "BOOL": false
        },
        "PageCount": {
            "N": "600"
        },
        "Price": {
            "N": "2000"
        },
        "ProductCategory": {
            "S": "Book"
        },
        "Title": {
            "S": "Book 103 Title"
        }
    }
},{
    "Item": {
```

```
        "Authors": {
            "SS": ["Author1", "Author2"]
        },
        "Dimensions": {
            "S": "8.5 x 11.0 x 1.5"
        },
        "ISBN": {
            "S": "444-444444444"
        },
        "Id": {
            "N": "104"
        },
        "InPublication": {
            "BOOL": false
        },
        "PageCount": {
            "N": "600"
        },
        "Price": {
            "N": "2000"
        },
        "ProductCategory": {
            "S": "Book"
        },
        "Title": {
            "S": "Book 104 Title"
        }
    }
},{
    "Item": {
        "Authors": {
            "SS": ["Author1", "Author2"]
        },
        "Dimensions": {
            "S": "8.5 x 11.0 x 1.5"
        },
        "ISBN": {
            "S": "555-5555555555"
        },
        "Id": {
            "N": "105"
        },
        "InPublication": {
            "BOOL": false
```

```
        },
        "PageCount": {
            "N": "600"
        },
        "Price": {
            "N": "2000"
        },
        "ProductCategory": {
            "S": "Book"
        },
        "Title": {
            "S": "Book 105 Title"
        }
    }
}]
```

### Amazon Ion

Amazon Ion은 대규모 서비스 중심 아키텍처를 엔지니어링할 때 매일 직면하는 신속한 개발, 분리, 효율성 문제를 해결하기 위해 빌드된 서식 있는 자기 기술형 계층적 데이터 직렬화 형식입니다.

Ion 형식으로 데이터를 가져오면 Ion 데이터 유형이 새 DynamoDB 테이블의 DynamoDB 데이터 유형에 매핑됩니다.

| 일련 번호 | Ion에서 DynamoDB로 데이터 유형 변환 | B |
|---|---|---|
| 1 | `Ion Data Type` | `DynamoDB Representation` |
| 2 | `string` | `String (s)` |
| 3 | `bool` | `Boolean (BOOL)` |
| 4 | `decimal` | `Number (N)` |
| 5 | `blob` | `Binary (B)` |
| 6 | `list (with type annotation $dynamodb` | `Set (SS, NS, BS)` |

| 일련 번호 | Ion에서 DynamoDB로 데이터 유형 변환 | B |
|---|---|---|
| | _SS, $dynamodb_NS, or $dynamodb_BS) | |
| 7 | list | List |
| 8 | struct | Map |

Ion 파일의 항목은 줄 바꿈으로 구분됩니다. 각 줄은 Ion 버전 마커로 시작되고 그 다음에 Ion 형식의 항목이 표시됩니다.

> **Note**
>
> 다음 예제에서는 가독성을 높이고자 Ion 형식 파일의 항목이 여러 줄 형식으로 되어 있습니다.

```
$ion_1_0
[
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"333-3333333333",
      Id:103.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 103 Title"
    }
  },
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"444-4444444444",
      Id:104.,
      InPublication:false,
      PageCount:6d2,
```

```
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 104 Title"
    }
  },
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"555-5555555555",
      Id:105.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 105 Title"
    }
  }
]
```

## 가져오기 형식 할당량 및 유효성 검사

### 가져오기 할당량

Amazon S3에서 DynamoDB 가져오기는 us-east-1, us-west-2 및 eu-west-1 리전에서 한 번에 총 가져오기 소스 객체 크기가 15TB인 최대 50개의 동시 가져오기 작업을 지원할 수 있습니다. 다른 모든 리전에서는 총 크기가 1TB인 최대 50개의 동시 가져오기 작업이 지원됩니다. 각 가져오기 작업은 모든 리전에서 최대 5만 개의 Amazon S3 객체를 가져올 수 있습니다. 이러한 기본 할당량은 모든 계정에 적용됩니다. 이러한 할당량을 수정해야 한다고 생각되면 계정 팀에 문의하세요. 이것은 사례별로 검토됩니다. DynamoDB 제한에 대한 자세한 내용은 Service Quotas를 참조하세요.

### 유효성 검사 오류

가져오기 프로세스 중에 DynamoDB에서 데이터를 구문 분석하는 동안 오류가 발생할 수 있습니다. 각 오류에 대해 DynamoDB는 CloudWatch 로그를 내보내며 발생한 총 오류 수를 로깅합니다. Amazon S3 객체 자체의 형식이 잘못되었거나 해당 콘텐츠가 DynamoDB 항목을 형성할 수 없는 경우 객체의 나머지 부분 처리를 건너뛸 수 있습니다.

> **Note**
>
> Amazon S3 데이터 소스에 동일한 키를 공유하는 여러 항목이 있는 경우 하나가 남을 때까지 항목을 덮어씁니다. 이렇게 하면 1개의 항목을 가져오고 다른 항목은 무시된 것처럼 나타날 수

있습니다. 중복 항목은 임의의 순서로 덮어써지고 오류로 계산되지 않으며 CloudWatch 로그로 내보내지지 않습니다.
가져오기가 완료되면 가져온 총 항목 수, 총 오류 수 및 처리된 총 항목 수를 볼 수 있습니다. 추가 문제 해결을 위해 가져온 항목의 총 크기와 처리된 데이터의 총 크기를 확인할 수도 있습니다.

가져오기 오류에는 API 유효성 검사 오류, 데이터 유효성 검사 오류 및 구성 오류의 세 가지 범주가 있습니다.

API 유효성 검사 오류

API 유효성 검사 오류는 동기화 API의 항목 수준 오류입니다. 일반적인 원인은 권한 문제, 필수 파라미터 누락 및 파라미터 유효성 검사 실패입니다. API 호출이 실패한 이유에 대한 세부 정보는 `ImportTable` 요청에서 발생한 예외에 포함되어 있습니다.

데이터 유효성 검사 오류

데이터 유효성 검사 오류는 항목 수준 또는 파일 수준에서 발생할 수 있습니다. 가져오는 동안 대상 테이블로 가져오기 전에 DynamoDB 규칙에 따라 항목의 유효성이 검사됩니다. 항목이 유효성 검사에 실패하여 항목을 가져오지 않는 경우 가져오기 작업이 해당 항목을 건너뛰고 다음 항목으로 계속 진행됩니다. 작업이 끝나면 가져오기 상태가 FAILED로 설정되고 FailureCode, ItemValidationError 및 FailureMessage "Some of the items failed validation checks and were not imported(유효성 검사에 실패하여 일부 항목을 가져오지 않았습니다.) Please check CloudWatch error logs for more details(자세한 내용은 CloudWatch 오류 로그를 참조하십시오)"가 표시됩니다.

데이터 유효성 검사 오류의 일반적인 원인에는 객체를 구문 분석할 수 없거나, 객체의 형식이 잘못되었거나(입력은 DYNAMODB_JSON을 지정하지만 객체가 DYNAMODB_JSON에 없음), 스키마가 지정된 소스 테이블 키와 일치하지 않는 것 등이 포함됩니다.

구성 오류

구성 오류는 일반적으로 권한 유효성 검사로 인한 워크플로 오류입니다. 가져오기 워크플로는 요청을 수락한 후 일부 권한을 확인합니다. Amazon S3 또는 CloudWatch와 같은 필수 종속 항목을 호출하는 데 문제가 있는 경우 프로세스에서 가져오기 상태를 FAILED로 표시합니다. `failureCode` 및 `failureMessage`는 실패 이유를 가리킵니다. 해당하는 경우 실패 메시지에는 CloudTrail에서 실패 이유를 조사하는 데 사용할 수 있는 요청 ID도 포함됩니다.

일반적인 구성 오류에는 Amazon S3 버킷의 URL이 잘못되었거나 Amazon S3 버킷, CloudWatch Logs 및 Amazon S3 객체의 암호를 해독하는 데 사용되는 AWS KMS 키에 액세스할 수 있는 권한이 없는 것 등이 포함됩니다. 자세한 내용은 사용 및 데이터 키를 참조하세요.

소스 Amazon S3 객체 유효성 검사

소스 S3 객체의 유효성을 검사하려면 다음 단계를 수행합니다.

1. 데이터 형식 및 압축 유형 유효성 검사

   - 지정된 접두사 아래의 일치하는 모든 Amazon S3 객체가 동일한 형식(DYNAMODB_JSON, DYNAMODB_ION, CSV)인지 확인합니다.
   - 지정된 접두사 아래의 일치하는 모든 Amazon S3 객체가 동일한 방식(GZIP, ZSTD, NONE)으로 압축되었는지 확인합니다.

   > **Note**
   >
   > ImportTable 호출에 지정된 입력 형식이 우선하므로 Amazon S3 객체는 해당 확장자(.csv/.json/.ion/.gz/.zstd 등)를 가질 필요가 없습니다.

2. 가져오기 데이터가 원하는 테이블 스키마를 준수하는지 검증합니다.

   - 소스 데이터의 각 항목에 프라이머리 키가 있는지 확인합니다. 가져오기에서 정렬 키는 선택 사항입니다.
   - 프라이머리 키 및 정렬 키와 연결된 속성 유형이 테이블 생성 파라미터에 지정된 대로 테이블 및 GSI 스키마의 속성 유형과 일치하는지 확인합니다.

문제 해결

CloudWatch 로그

실패한 가져오기 작업의 경우 자세한 오류 메시지가 CloudWatch 로그에 게시됩니다. 이러한 로그에 액세스하려면 먼저 다음 명령을 사용하여 출력 및 describe-import에서 ImportArn을 검색합니다.

```
aws dynamodb describe-import --import-arn arn:aws:dynamodb:us-east-1:ACCOUNT:table/
target-table/import/01658528578619-c4d4e311
}
```

출력 예제:

```
aws dynamodb describe-import --import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311"
{
    "ImportTableDescription": {
        "ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658528578619-c4d4e311",
        "ImportStatus": "FAILED",
        "TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
        "TableId": "7b7ecc22-302f-4039-8ea9-8e7c3eb2bcb8",
        "ClientToken": "30f8891c-e478-47f4-af4a-67a5c3b595e3",
        "S3BucketSource": {
            "S3BucketOwner": "ACCOUNT",
            "S3Bucket": "my-import-source",
            "S3KeyPrefix": "import-test"
        },
        "ErrorCount": 1,
        "CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-
dynamodb/imports:*",
        "InputFormat": "CSV",
        "InputCompressionType": "NONE",
        "TableCreationParameters": {
            "TableName": "target-table",
            "AttributeDefinitions": [
                {
                    "AttributeName": "pk",
                    "AttributeType": "S"
                }
            ],
            "KeySchema": [
                {
                    "AttributeName": "pk",
                    "KeyType": "HASH"
                }
            ],
            "BillingMode": "PAY_PER_REQUEST"
        },
        "StartTime": 1658528578.619,
        "EndTime": 1658528750.628,
        "ProcessedSizeBytes": 70,
        "ProcessedItemCount": 1,
        "ImportedItemCount": 0,
        "FailureCode": "ItemValidationError",
```

```
        "FailureMessage": "Some of the items failed validation checks and were not
 imported. Please check CloudWatch error logs for more details."
    }
}
```

위의 응답에서 로그 그룹과 가져오기 ID를 검색하고 이를 사용하여 오류 로그를 검색합니다. 가져오기 ID는 ImportArn 필드의 마지막 경로 요소입니다. 로그 그룹 이름은 /aws-dynamodb/imports입니다. 오류 로그 스트림 이름은 import-id/error입니다. 이 예제에서는 01658528578619-c4d4e311/error입니다.

항목에 pk 키가 누락됨

소스 S3 객체에 파라미터로 제공된 프라이머리 키가 포함되어 있지 않으면 가져오기가 실패합니다. 예를 들어 가져오기에 대한 프라이머리 키를 열 이름 'pk'로 정의할 수 있습니다.

```
aws dynamodb import-table –s3-bucket-source S3Bucket=my-import-
source,S3KeyPrefix=import-test.csv \
           –input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
           [{"AttributeName":"pk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"pk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}'
```

다음 내용이 있는 소스 객체 import-test.csv에서 'pk' 열이 누락되었습니다.

```
title,artist,year_of_release
The Dark Side of the Moon,Pink Floyd,1973
```

데이터 소스의 프라이머리 키가 누락되어 항목 유효성 검사 오류로 인해 가져오기가 실패합니다.

CloudWatch 오류 로그의 예:

```
aws logs get-log-events –log-group-name /aws-dynamodb/imports –log-stream-name
 01658528578619-c4d4e311/error
{
"events": [
{
"timestamp": 1658528745319,
"message": "{\"itemS3Pointer\":{\"bucket\":\"my-import-source\",\"key\":
\"import-test.csv\",\"itemIndex\":0},\"importArn\":\"arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311\",\"errorMessages
\":[\"One or more parameter values were invalid: Missing the key pk in the item\"]}",
"ingestionTime": 1658528745414
```

```
}
],
"nextForwardToken": "f/36986426953797707963335499204463414460239026137054642176/s",
"nextBackwardToken": "b/36986426953797707963335499204463414460239026137054642176/s"
}
```

이 오류 로그에는 "One or more parameter values were invalid: Missing the key pk in the item(하나 이상의 파라미터 값이 잘못되었습니다. 항목에 키 pk가 누락되었습니다)"라고 나와 있습니다. 이 가져오기 작업이 실패했기 때문에 'target-table' 테이블이 존재하지만 가져온 항목이 없으므로 비어 있습니다. 첫 번째 항목이 처리되었고 객체가 항목 유효성 검사에 실패했습니다.

문제를 해결하려면 'target-table'이 더 이상 필요하지 않은 경우 먼저 이를 삭제합니다. 그런 다음 소스 객체에 있는 프라이머리 키 열 이름을 사용하거나 소스 데이터를 다음과 같이 업데이트합니다.

```
pk,title,artist,year_of_release
Albums::Rock::Classic::1973::AlbumId::ALB25,The Dark Side of the Moon,Pink Floyd,1973
```

대상 테이블 존재

가져오기 작업을 시작하고 다음과 같은 응답을 받았습니다.

```
An error occurred (ResourceInUseException) when calling the ImportTable operation:
 Table already exists: target-table
```

이 오류를 해결하려면 아직 존재하지 않는 테이블 이름을 선택하고 가져오기를 다시 시도해야 합니다.

지정된 버킷이 존재하지 않습니다

소스 버킷이 없는 경우 가져오기가 실패하고 오류 메시지 세부 정보가 CloudWatch에 기록됩니다.

describe import 예제:

```
aws dynamodb –endpoint-url $ENDPOINT describe-import –import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658530687105-e6035287"
{
"ImportTableDescription": {
"ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658530687105-e6035287",
"ImportStatus": "FAILED",
"TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
"TableId": "e1215a82-b8d1-45a8-b2e2-14b9dd8eb99c",
"ClientToken": "3048e16a-069b-47a6-9dfb-9c259fd2fb6f",
```

```
"S3BucketSource": {
"S3BucketOwner": "531234567890",
"S3Bucket": "BUCKET_DOES_NOT_EXIST",
"S3KeyPrefix": "import-test"
},
"ErrorCount": 0,
"CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-dynamodb/
imports:*",
"InputFormat": "CSV",
"InputCompressionType": "NONE",
"TableCreationParameters": {
"TableName": "target-table",
"AttributeDefinitions": [
{
"AttributeName": "pk",
"AttributeType": "S"
}
],
"KeySchema": [
{
"AttributeName": "pk",
"KeyType": "HASH"
}
],
"BillingMode": "PAY_PER_REQUEST"
},
"StartTime": 1658530687.105,
"EndTime": 1658530701.873,
"ProcessedSizeBytes": 0,
"ProcessedItemCount": 0,
"ImportedItemCount": 0,
"FailureCode": "S3NoSuchBucket",
"FailureMessage": "The specified bucket does not exist (Service: Amazon S3; Status
 Code: 404; Error Code: NoSuchBucket; Request ID: Q4W6QYYFDWY6WAKH; S3 Extended Request
 ID: ObqSlLeIMJpQqHLRX2C5Sy7n+8g6iGPwy7ixg7eEeTuEkg/+chU/JF+RbliWytMlkUlUcuCLTrI=;
 Proxy: null)"
}
}
```


FailureCode는 S3NoSuchBucket이며, FailureMessag에는 요청 ID 및 오류가 발생한 서비스와 같은 세부 정보가 포함되어 있습니다. 데이터를 테이블로 가져오기 전에 오류가 발견되었으므로 새 DynamoDB 테이블이 생성되지 않습니다. 경우에 따라 데이터 가져오기가 시작된 후 이러한 오류가 발생하면 부분적으로 가져온 데이터가 있는 테이블이 유지됩니다.

이 오류를 해결하려면 소스 Amazon S3 버킷이 있는지 확인한 다음 가져오기 프로세스를 다시 시작합니다.

## Amazon S3에서 DynamoDB로 가져오기 모범 사례

다음은 Amazon S3에서 DynamoDB로 가져올 때의 모범 사례입니다.

S3 객체 5만 개 한도 이하로 유지

각 가져오기 작업은 최대 5만 개의 S3 객체를 지원합니다. 데이터세트에 5만 개 이상의 객체가 포함된 경우 이를 더 큰 객체로 통합하는 것을 고려해 보세요.

너무 큰 S3 객체 피하기

S3 객체는 병렬로 가져오게 됩니다. 중간 크기의 S3 객체가 여러 개 있으면 과도한 오버헤드 없이 병렬 실행이 가능합니다. 1KB 미만인 항목의 경우 각 S3 객체에 4백만 개의 항목을 배치하는 것을 고려해 보세요. 평균 항목 크기가 더 큰 경우 각 S3 객체에 비례적으로 더 적은 수의 항목을 배치하세요.

정렬된 데이터 무작위화

S3 객체가 데이터를 정렬된 순서로 보관하는 경우 롤링 핫 파티션을 생성할 수 있습니다. 이 상황은 한 파티션이 모든 활동을 수신한 후에 다음 파티션이 차례로 수신하는 식입니다. 순서가 정렬된 데이터는 가져오기 중에 동일한 대상 파티션에 기록되는 S3 객체의 시퀀스 항목으로 정의됩니다. 데이터 순서가 정렬되는 일반적인 상황 중 하나는 항목이 파티션 키를 기준으로 정렬되어 반복되는 항목이 동일한 파티션 키를 공유하는 CSV 파일입니다.

롤링 핫 파티션을 방지하려면 이러한 경우 순서를 무작위로 지정하는 것이 좋습니다. 이렇게 하면 쓰기 작업을 분산하여 성능을 향상시킬 수 있습니다. 자세한 내용은 DynamoDB에 데이터 업로드 중 효율적으로 쓰기 활동 배포 단원을 참조하십시오.

데이터를 압축하여 총 S3 객체 크기를 리전 한도 이하로 유지

S3에서 가져오기 프로세스에서는 가져올 S3 객체 데이터의 총 크기 합계에 한도가 있습니다. 한도는 us-east-1, us-west-2, eu-west-1 리전의 경우 15TB, 다른 리전의 경우 1TB입니다. 한도는 원시 S3 객체 크기를 기반으로 합니다.

압축을 통해 더 많은 원시 데이터를 한도 내에 맞출 수 있습니다. 압축만으로는 가져오기를 한도 내에 맞출 수 없는 경우 AWS Premium Support에 문의하여 할당량을 늘릴 수도 있습니다.

항목 크기가 성능에 미치는 영향 파악

평균 항목 크기가 매우 작은 경우(200바이트 미만), 가져오기 프로세스가 큰 항목 크기보다 약간 더 오래 걸릴 수 있습니다.

글로벌 보조 인덱스 없이 가져오기 고려

가져오기 작업의 지속 시간은 하나 이상의 글로벌 보조 인덱스(GSI)가 있는지 여부에 따라 달라질 수 있습니다. 카디널리티가 낮은 파티션 키로 인덱스를 설정하려는 경우 가져오기 작업에 포함하는 것보다 가져오기 작업이 완료될 때까지 인덱스 생성을 연기하면 가져오기 속도가 빨라질 수 있습니다.

> **Note**
>
> 가져오기 중에 GSI를 생성하면 쓰기 요금이 발생하지 않습니다(가져오기 후에 GSI를 생성하면 요금이 발생함).

## Amazon S3로 DynamoDB 데이터 내보내기: 작동 방식

S3로 DynamoDB 내보내기는 DynamoDB 데이터를 Amazon S3 버킷에 대규모로 내보내는 완전관리형 솔루션입니다. S3로 DynamoDB 내보내기를 사용하면 특정 시점 복구(PITR) 기간 내 언제든지 Amazon DynamoDB 테이블에서 Amazon S3 버킷으로 데이터를 내보낼 수 있습니다. 내보내기 기능을 사용하려면 테이블에서 PITR을 활성화해야 합니다. 이 기능을 사용하면 Athena, AWS Glue, Amazon SageMaker AI, Amazon EMR, AWS Lake Formation 등의 다른 AWS 서비스를 사용하여 데이터에 대한 분석과 복잡한 쿼리를 수행할 수 있습니다.

S3로 DynamoDB 내보내기를 사용하면 DynamoDB 테이블에서 전체 데이터와 증분 데이터를 모두 내보낼 수 있습니다. 내보내기는 비동기식이며 읽기 용량 단위(RCU)를 사용하지 않으므로 테이블 성능 및 가용성에 영향을 주지 않습니다. 지원되는 내보내기 파일 형식은 DynamoDB JSON 및 Amazon Ion 형식입니다. 다른 AWS 계정이 소유한 S3 버킷과 다른 AWS 리전으로 데이터를 내보낼 수도 있습니다. 데이터는 항상 종단 간 암호화됩니다.

DynamoDB 전체 내보내기는 내보내기가 완료된 시점의 DynamoDB 테이블(테이블 데이터 및 로컬 보조 인덱스) 크기를 기준으로 요금이 부과됩니다. DynamoDB 증분 내보내기는 내보내는 기간 동안 연속 백업에서 처리된 데이터 크기를 기준으로 요금이 부과됩니다. 증분 내보내기에는 최소 10MB의 요금이 부과됩니다. 내보낸 데이터를 Amazon S3에 저장하는 경우와 Amazon S3 버킷에 대해 PUT 요청을 하는 경우에는 추가 요금이 부과됩니다. 자세한 내용은 Amazon DynamoDB 요금 및 Amazon S3 요금을 참조하세요.

서비스 할당량에 대한 자세한 내용은 Amazon S3로 테이블 내보내기 섹션을 참조하세요.

주제

- DynamoDB에서 테이블 내보내기 요청
- DynamoDB 테이블 내보내기 출력 형식

# DynamoDB에서 테이블 내보내기 요청

DynamoDB 테이블 내보내기를 사용하면 테이블 데이터를 Amazon S3 버킷으로 내보낼 수 있으므로, Athena, AWS Glue, Amazon SageMaker AI, Amazon EMR, AWS Lake Formation 등의 다른 AWS 서비스를 사용하여 데이터에 대한 분석과 복잡한 쿼리를 수행할 수 있습니다. AWS Management Console, AWS CLI 또는 DynamoDB API를 사용하여 DynamoDB 테이블 내보내기를 요청할 수 있습니다.

> **Note**
>
> Amazon S3 요청자 지불 버킷은 지원되지 않습니다.

DynamoDB는 전체 내보내기와 증분 내보내기를 모두 지원합니다.

- 전체 내보내기를 사용하면 특정 시점 복구(PITR) 기간 내 어느 시점에서든 테이블의 전체 스냅샷을 Amazon S3 버킷으로 내보낼 수 있습니다.
- 증분 내보내기를 사용하면 PITR 기간 내에서 지정된 기간 사이에 변경, 업데이트 또는 삭제된 DynamoDB 테이블의 데이터를 Amazon S3 버킷으로 내보낼 수 있습니다.

주제


- 사전 조건
- AWS Management Console을 사용하여 내보내기 요청
- AWS Management Console에서 이전 내보내기에 대한 세부 정보 확인
- AWS CLI를 사용하여 내보내기 요청
- AWS CLI에서 이전 내보내기에 대한 세부 정보 확인
- AWS SDK를 사용하여 내보내기 요청
- AWS SDK를 사용하여 이전 내보내기에 대한 세부 정보 확인


## 사전 조건

### PITR 활성화

S3로 내보내기 기능을 사용하려면 테이블에서 PITR을 활성화해야 합니다. PITR을 활성화하는 방법에 대한 자세한 내용은 시점 복구를 참조하세요. PITR이 활성화되지 않은 테이블에 대해 내보내기를 요청하는 경우 다음 예외 메시지와 함께 요청이 실패합니다. "An error occurred

(PointInTimeRecoveryUnavailableException) when calling the ExportTableToPointInTime operation: Point in time recovery is not enabled for table 'my-dynamodb-table". 구성된 PITR RecoveryPeriodInDays 내에 있는 시점에서만 요청하고 내보낼 수 있습니다.

S3 권한 설정

쓸 수 있는 권한이 있는 Amazon S3 버킷으로 테이블 데이터를 내보낼 수 있습니다. 대상 버킷이 소스 테이블 소유자와 같은 AWS 리전에 있거나 소유자가 동일하지 않아도 됩니다. AWS Identity and Access Management(IAM) 정책에서 S3 작업(s3:AbortMultipartUpload, s3:PutObject, s3:PutObjectAcl) 및 DynamoDB 내보내기 작업(dynamodb:ExportTableToPointInTime)을 수행할 수 있도록 허용해야 합니다. 다음은 사용자에게 S3 버킷으로 내보내기를 수행할 수 있는 권한을 부여하는 샘플 정책의 예입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBExportAction",
            "Effect": "Allow",
            "Action": "dynamodb:ExportTableToPointInTime",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table"
        },
        {
            "Sid": "AllowS3BucketWrites",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

다른 계정에 있는 Amazon S3 버킷에 쓰기 작업을 수행해야 하거나 쓰기 권한이 없는 경우, Amazon S3 버킷 소유자는 DynamoDB에서 해당 버킷으로 내보낼 수 있도록 허용하는 버킷 정책을 추가해야 합니다. 다음은 대상 Amazon S3 버킷에 대한 정책 예시입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

내보내기가 진행 중일 때 이러한 권한을 취소하면 부분 파일이 생성됩니다.

> **Note**
>
> 내보내려는 테이블이나 버킷이 고객 관리형 키로 암호화된 경우 해당 KMS 키의 정책은 DynamoDB에 키를 사용할 수 있는 권한을 부여해야 합니다. 이 권한은 내보내기 작업을 트리거하는 IAM 사용자/역할을 통해 부여됩니다. 모범 사례를 포함한 암호화에 대한 자세한 내용은 DynamoDB가 AWS KMS를 사용하는 방법 및 사용자 지정 KMS 키 사용을 참조하세요.

AWS Management Console을 사용하여 내보내기 요청

다음 예제에서는 DynamoDB 콘솔을 사용하여 `MusicCollection`이라는 기존 테이블을 내보내는 방법을 보여줍니다.

> **Note**
>
> 이 절차에서는 특정 시점으로 복구를 활성화했다고 가정합니다. `MusicCollection` 테이블에 대해 이 기능을 활성화하려면 테이블의 개요 탭에 있는 테이블 세부 정보 섹션에서 특정 시점으로 복구에 대해 활성화를 선택합니다.

테이블 내보내기를 요청하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 S3로 내보내기(Exports to S3)를 선택합니다.
3. S3로 내보내기 버튼을 선택합니다.
4. 소스 테이블과 대상 S3 버킷을 선택합니다. 사용자 계정이 대상 버킷을 소유한 경우 Browse S3(S3 찾아보기) 버튼을 클릭하여 찾을 수 있습니다. 그렇지 않은 경우, 버킷의 URL을 `s3://bucketname/prefix format.`으로 입력합니다. **`prefix`**는 대상 버킷을 체계적으로 구성된 상태로 유지하는 데 도움이 되는 선택적 폴더입니다.
5. 전체 내보내기 또는 증분 내보내기를 선택합니다. 전체 내보내기는 지정한 시점의 테이블의 전체 테이블 스냅샷을 출력합니다. 증분 내보내기는 지정된 내보내기 기간 동안 테이블에 적용된 변경 사항을 출력합니다. 출력은 내보내기 기간의 최종 항목 상태만 포함하도록 압축됩니다. 동일한 내보내기 기간 내에 여러 번 업데이트된 경우에도 항목은 내보내기에 한 번만 표시됩니다.

   Full export

   1. 전체 테이블 스냅샷을 내보낼 시점을 선택합니다. PITR 기간 내의 어느 시점이나 가능합니다. 또는 현재 시간을 선택하여 최신 스냅샷을 내보낼 수도 있습니다.
   2. 내보낸 파일 형식에서 DynamoDB JSON과 Amazon Ion 중에 선택합니다. 기본적으로 테이블은 특정 시점으로 복구 기간 중 복원 가능한 마지막 시간부터 DynamoDB JSON 형식으로 내보내고 Amazon S3 키(SSE-S3)를 사용하여 암호화됩니다. 필요할 경우 이러한 내보내기 설정을 변경할 수 있습니다.

      > **Note**
      >
      > AWS Key Management Service(AWS KMS)에 의해 보호되는 키를 사용하여 내보내기를 암호화하도록 선택하는 경우 해당 키는 대상 S3 버킷과 동일한 리전에 있어야 합니다.

Incremental export

1. 증분 데이터를 내보내려는 내보내기 기간을 선택합니다. PITR 기간에서 시작 시간을 선택합니다. 내보내기 기간은 15분 이상, 24시간 이하여야 합니다. 내보내기 기간의 시작 시간은 포함되며 종료 시간은 제외됩니다.
2. 절대 모드와 상대 모드 중에서 선택합니다.
   a. 절대 모드는 지정된 기간 동안 증분 데이터를 내보냅니다.
   b. 상대 모드는 내보내기 작업 제출 시간을 기준으로 하는 내보내기 기간 동안 증분 데이터를 내보냅니다.
3. 내보낸 파일 형식에서 DynamoDB JSON과 Amazon Ion 중에 선택합니다. 기본적으로 테이블은 특정 시점으로 복구 기간 중 복원 가능한 마지막 시간부터 DynamoDB JSON 형식으로 내보내고 Amazon S3 키(SSE-S3)를 사용하여 암호화됩니다. 필요할 경우 이러한 내보내기 설정을 변경할 수 있습니다.

   > **Note**
   >
   > AWS Key Management Service(AWS KMS)에 의해 보호되는 키를 사용하여 내보내기를 암호화하도록 선택하는 경우 해당 키는 대상 S3 버킷과 동일한 리전에 있어야 합니다.

4. 보기 유형 내보내기에서 새 이미지와 이전 이미지 또는 새 이미지만을 선택합니다. 새 이미지는 항목의 최신 상태를 제공합니다. 이전 이미지는 지정된 시작 날짜 및 시간 바로 이전의 항목 상태를 제공합니다. 기본 설정은 새 이미지와 이전 이미지입니다. 새 이미지와 이전 이미지에 대한 자세한 내용은 증분 내보내기 출력 섹션을 참조하세요.

6. 시작하려면 내보내기를 선택합니다.

내보낸 데이터는 트랜잭션에서 일관성이 없습니다. 트랜잭션 작업이 두 개의 내보내기 출력 사이에서 혼선을 겪을 수 있습니다. 트랜잭션 작업에 의해 수정된 항목 중 일부는 내보내기에 반영되는 반면, 동일한 트랜잭션에서 수정된 다른 일부 항목은 동일한 내보내기 요청에 반영되지 않을 수 있습니다. 하지만 내보내기는 결국에 일관되게 이루어집니다. 내보내기 중에 트랜잭션이 중단되는 경우 다음 번 연속 내보내기에서 중복 없이 남은 트랜잭션이 수행됩니다. 내보내기에 사용되는 기간은 내부 시스템 시계를 기준으로 하며 애플리케이션의 로컬 시계와 1분씩 달라질 수 있습니다.

AWS Management Console에서 이전 내보내기에 대한 세부 정보 확인

이전에 실행한 내보내기 태스크에 대한 정보는 탐색 사이드바에서 S3로 내보내기 섹션을 선택하여 찾을 수 있습니다. 이 섹션에는 지난 90일간 생성한 모든 내보내기의 목록이 포함되어 있습니다. 내보내기 탭에 나열된 태스크의 ARN을 선택하면 선택한 고급 구성 설정을 포함하여 해당 내보내기에 대한 정보가 검색됩니다. 내보내기 태스크 메타데이터는 90일이 지나면 만료되고 해당 기간보다 오래된 작업은 이 목록에서 더 이상 찾을 수 없지만 S3 버킷의 객체는 버킷 정책에서 허용하는 기간까지 유지됩니다. DynamoDB는 내보내기 중 S3 버킷에서 생성하는 객체를 삭제하지 않습니다.

AWS CLI를 사용하여 내보내기 요청

다음 예제에서는 AWS CLI를 사용하여 `MusicCollection`이라는 기존 테이블을 `ddb-export-musiccollection`이라는 S3 버킷으로 내보내는 방법을 보여줍니다.

> **Note**
>
> 이 절차에서는 특정 시점으로 복구를 활성화했다고 가정합니다. 다음 명령을 실행하여 `MusicCollection` 테이블에 대해 이 기능을 활성화하세요.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name MusicCollection \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Full export

다음 명령은 `MusicCollection`을 접두사가 `2020-Nov`인 S3 버킷 `ddb-export-musiccollection-9012345678`으로 내보냅니다. 테이블 데이터는 특정 시점으로 복구 기간 중 특정 시간부터 DynamoDB JSON 형식으로 내보내고 Amazon S3 키(SSE-S3)를 사용하여 암호화됩니다.

> **Note**
>
> 교차 계정 테이블 내보내기를 요청하는 경우 반드시 `--s3-bucket-owner` 옵션을 포함해야 합니다.

```
aws dynamodb export-table-to-point-in-time \
```

```
  --table-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
  --s3-bucket ddb-export-musiccollection-9012345678 \
  --s3-prefix 2020-Nov \
  --export-format DYNAMODB_JSON \
  --export-time 1604632434 \
  --s3-bucket-owner 9012345678 \
  --s3-sse-algorithm AES256
```

Incremental export

다음 명령은 새 `--export-type` 및 `--incremental-export-specification`을 제공하여 증분 내보내기를 수행합니다. 기울임꼴로 표시된 값을 사용자 고유의 값으로 대체합니다. 시간은 epoch 이후 경과 시간(초)으로 지정됩니다.

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:REGION:ACCOUNT:table/TABLENAME \
  --s3-bucket BUCKET --s3-prefix PREFIX \
  --incremental-export-specification
 ExportFromTime=1693569600,ExportToTime=1693656000,ExportViewType=NEW_AND_OLD_IMAGES
 \
  --export-type INCREMENTAL_EXPORT
```

> **Note**
>
> AWS Key Management Service(AWS KMS)에 의해 보호되는 키를 사용하여 내보내기를 암호화하도록 선택하는 경우 해당 키는 대상 S3 버킷과 동일한 리전에 있어야 합니다.

AWS CLI에서 이전 내보내기에 대한 세부 정보 확인

이전에 실행한 내보내기 요청에 대한 정보는 `list-exports` 명령을 사용하여 확인할 수 있습니다. 이 명령은 지난 90일간 생성한 모든 내보내기의 목록을 반환합니다. 내보내기 태스크 메타데이터는 90일이 지나면 만료되고 해당 기간보다 오래된 작업은 `list-exports` 명령에서 더 이상 반환하지 않지만 S3 버킷의 객체는 버킷 정책에서 허용하는 기간까지 유지됩니다. DynamoDB는 내보내기 중 S3 버킷에서 생성하는 객체를 삭제하지 않습니다.

내보내기는 성공 또는 실패할 때까지의 `PENDING` 상태를 유지합니다. 성공하면 상태가 `COMPLETED`로 바뀝니다. 실패하면 상태가 `FAILED`로 바뀌고 `failure_message` 및 `failure_reason`이 표시됩니다.

다음 예제에서는 선택적 `table-arn` 파라미터를 사용하여 특정 테이블의 내보내기만 나열합니다.

```
aws dynamodb list-exports \
    --table-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog
```

특정 내보내기 태스크에 대해 고급 구성 설정을 포함하여 자세한 정보를 검색하려면 `describe-export` 명령을 사용합니다.

```
aws dynamodb describe-export \
    --export-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4
```

### AWS SDK를 사용하여 내보내기 요청

이 코드 조각을 통해 원하는 AWS SDK를 사용하여 테이블 내보내기를 요청할 수 있습니다.

#### Python

##### 전체 내보내기

```
import boto3
from datetime import datetime

# remove endpoint_url for real use
client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/export_table_to_point_in_time.html
client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
    ExportTime=datetime(2023, 9, 20, 12, 0, 0),
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

##### 증분 내보내기

```
import boto3
from datetime import datetime
```

```
client = boto3.client('dynamodb')

client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
    IncrementalExportSpecification={
      'ExportFromTime': datetime(2023, 9, 20, 12, 0, 0),
      'ExportToTime': datetime(2023, 9, 20, 13, 0, 0),
      'ExportViewType': 'NEW_AND_OLD_IMAGES'
    },
    ExportType='INCREMENTAL_EXPORT',
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

### AWS SDK를 사용하여 이전 내보내기에 대한 세부 정보 확인

이 코드 조각을 통해 원하는 AWS SDK를 사용하여 이전 테이블 내보내기에 대한 세부 정보를 얻을 수 있습니다.

#### Python

##### 목록

```
import boto3

client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/list_exports.html

print(
  client.list_exports(
     TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
  )
)
```

##### 설명

```
import boto3
```

```
client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/describe_export.html

print(
  client.describe_export(
     ExportArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE/
export/01695353076000-06e2188f',
  )['ExportDescription']
)
```

## DynamoDB 테이블 내보내기 출력 형식

DynamoDB 테이블 내보내기에는 테이블 데이터가 포함된 파일 외에 매니페스트 파일이 포함됩니다. 해당 파일은 모두 내보내기 요청에서 지정하는 Amazon S3 버킷에 저장됩니다. 다음 단원에서는 각 출력 객체의 형식과 내용을 설명합니다.

전체 내보내기 출력

매니페스트 파일

DynamoDB는 매니페스트 파일과 해당 체크섬 파일을 각 내보내기 요청에 지정된 S3 버킷에 생성합니다.

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

테이블 내보내기를 요청할 때 **export-prefix**를 선택합니다. 이렇게 하면 대상 S3 버킷의 파일을 정리하는 데 도움이 됩니다. **ExportId**는 동일한 S3 버킷으로 여러 내보내기를 수행할 때 export-prefix가 서로 덮어쓰지 않도록 서비스에서 생성되는 고유한 토큰입니다.

이 내보내기를 수행하면 파티션당 하나 이상의 파일이 생성됩니다. 비어 있는 파티션의 경우 내보내기 요청을 통해 빈 파일이 생성됩니다. 각 파일의 모든 항목은 해당 파티션의 해시된 키스페이스에서 가져온 것입니다.

> **Note**
>
> 또한 DynamoDB는 `_started`라는 빈 파일을 매니페스트 파일과 동일한 디렉터리에 생성합니다. 이 파일은 대상 버킷이 쓰기 가능하고 내보내기가 시작되었는지 확인합니다. 해당 파일은 삭제해도 됩니다.

요약 매니페스트

`manifest-summary.json` 파일에는 내보내기 작업에 대한 요약 정보가 포함되어 있습니다. 이를 통해 공유 데이터 폴더의 어떤 데이터 파일이 이 내보내기와 연관되어 있는지 알 수 있습니다. 형식은 다음과 같습니다.

```
{
   "version": "2020-06-30",
   "exportArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4",
   "startTime": "2020-11-04T07:28:34.028Z",
   "endTime": "2020-11-04T07:33:43.897Z",
   "tableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog",
   "tableId": "12345a12-abcd-123a-ab12-1234abc12345",
   "exportTime": "2020-11-04T07:28:34.028Z",
   "s3Bucket": "ddb-productcatalog-export",
   "s3Prefix": "2020-Nov",
   "s3SseAlgorithm": "AES256",
   "s3SseKmsKeyId": null,
   "manifestFilesS3Key": "AWSDynamoDB/01693685827463-2d8752fd/manifest-files.json",
   "billedSizeBytes": 0,
   "itemCount": 8,
   "outputFormat": "DYNAMODB_JSON",
   "exportType": "FULL_EXPORT"
}
```

파일 매니페스트

`manifest-files.json` 파일에는 내보낸 테이블 데이터를 포함하는 파일에 대한 정보가 포함되어 있습니다. 파일은 JSON 라인 형식이므로 줄 바꿈이 항목 구분 기호로 사용됩니다. 다음 예제에서는 파일 매니페스트에 있는 한 데이터 파일의 세부 정보가 가독성을 향상하기 위해 여러 줄 형식으로 되어 있습니다.

```
{
```

```
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/01693685827463-2d8752fd/data/asdl123dasas.json.gz"
}
```

데이터 파일

DynamoDB는 테이블 데이터를 DynamoDB JSON 및 Amazon Ion의 두 가지 형식으로 내보낼 수 있습니다. 선택하는 형식과 관계없이 데이터는 키로 명명되는 여러 압축 파일에 기록됩니다. 이러한 파일도 `manifest-files.json` 파일에 나열됩니다.

전체 내보내기 이후의 Amazon S3 버킷의 디렉터리 구조에는 내보내기 Id 폴더 아래의 모든 매니페스트 파일과 데이터 파일이 포함됩니다.

```
amzn-s3-demo-bucket/DestinationPrefix
.
### AWSDynamoDB
    ### 01693685827463-2d8752fd     // the single full export
    #   ### manifest-files.json     // manifest points to files under 'data' subfolder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### data                    // The data exported by full export
    #   #   ### asdl123dasas.json.gz
    #   #   ...
    #   ### _started                // empty file for permission check
```

DynamoDB JSON

DynamoDB JSON 형식의 테이블 내보내기는 여러 `Item` 객체로 구성됩니다. 개별 객체는 DynamoDB의 표준 마샬링된 JSON 형식입니다.

DynamoDB JSON 내보내기 데이터의 사용자 지정 구문 분석기를 생성하는 경우에는 형식이 JSON 라인입니다. 즉, 줄 바꿈이 항목 구분 기호로 사용된다는 의미입니다. Athena, AWS Glue 등의 여러 AWS 서비스에서 이 형식을 자동으로 구문 분석합니다.

다음 예제에서는 DynamoDB JSON 내보내기의 단일 항목이 가독성을 향상하기 위해 여러 줄 형식으로 되어 있습니다.

```
{
```

```
    "Item":{
        "Authors":{
            "SS":[
                "Author1",
                "Author2"
            ]
        },
        "Dimensions":{
            "S":"8.5 x 11.0 x 1.5"
        },
        "ISBN":{
            "S":"333-3333333333"
        },
        "Id":{
            "N":"103"
        },
        "InPublication":{
            "BOOL":false
        },
        "PageCount":{
            "N":"600"
        },
        "Price":{
            "N":"2000"
        },
        "ProductCategory":{
            "S":"Book"
        },
        "Title":{
            "S":"Book 103 Title"
        }
    }
}
```

### Amazon Ion

Amazon Ion은 대규모 서비스 중심 아키텍처를 엔지니어링할 때 매일 직면하는 신속한 개발, 분리, 효율성 문제를 해결하기 위해 빌드된 서식 있는 자기 기술형 계층적 데이터 직렬화 형식입니다. DynamoDB는 JSON의 상위 집합인 Ion 텍스트 형식으로 테이블 데이터를 내보낼 수 있습니다.

테이블을 Ion 형식으로 내보내면 테이블에서 사용되는 DynamoDB 데이터 유형이 Ion 데이터 유형에 매핑됩니다. DynamoDB 세트는 Ion 형식 주석을 사용하여 소스 테이블에서 사용되는 데이터 유형을 구분합니다.

다음 표에는 DynamoDB 데이터 형식과 Ion 데이터 형식의 매핑이 나와 있습니다.

| DynamoDB 데이터 형식 | Ion 표현 |
| --- | --- |
| 문자열(S) | 문자열 |
| Boolean(BOOL) | bool |
| 숫자(N) | decimal |
| 이진수(B) | blob |
| 세트(SS, NSS, BS) | list(유형 주석 $dynamodb_SS, $dynamodb_NS 또는 $dynamodb_BS 사용) |
| 목록 | list |
| 맵 | struct |

Ion 내보내기의 항목은 줄 바꿈으로 구분됩니다. 각 줄은 Ion 버전 마커로 시작되고 그 다음에 Ion 형식의 항목이 표시됩니다. 다음 예제에서는 Ion 내보내기의 항목이 가독성을 향상하기 위해 여러 줄 형식으로 되어 있습니다.

```
$ion_1_0 {
    Item:{
        Authors:$dynamodb_SS::["Author1","Author2"],
        Dimensions:"8.5 x 11.0 x 1.5",
        ISBN:"333-3333333333",
        Id:103.,
        InPublication:false,
        PageCount:6d2,
        Price:2d3,
        ProductCategory:"Book",
        Title:"Book 103 Title"
    }
}
```

증분 내보내기 출력

매니페스트 파일

DynamoDB는 매니페스트 파일과 해당 체크섬 파일을 각 내보내기 요청에 지정된 S3 버킷에 생성합니다.

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

테이블 내보내기를 요청할 때 **export-prefix**를 선택합니다. 이렇게 하면 대상 S3 버킷의 파일을 정리하는 데 도움이 됩니다. **ExportId**는 동일한 S3 버킷으로 여러 내보내기를 수행할 때 export-prefix가 서로 덮어쓰지 않도록 서비스에서 생성되는 고유한 토큰입니다.

이 내보내기를 수행하면 파티션당 하나 이상의 파일이 생성됩니다. 비어 있는 파티션의 경우 내보내기 요청을 통해 빈 파일이 생성됩니다. 각 파일의 모든 항목은 해당 파티션의 해시된 키스페이스에서 가져온 것입니다.

> **Note**
>
> 또한 DynamoDB는 _started라는 빈 파일을 매니페스트 파일과 동일한 디렉터리에 생성합니다. 이 파일은 대상 버킷이 쓰기 가능하고 내보내기가 시작되었는지 확인합니다. 해당 파일은 삭제해도 됩니다.

요약 매니페스트

manifest-summary.json 파일에는 내보내기 작업에 대한 요약 정보가 포함되어 있습니다. 이를 통해 공유 데이터 폴더의 어떤 데이터 파일이 이 내보내기와 연관되어 있는지 알 수 있습니다. 형식은 다음과 같습니다.

```
{
 "version": "2023-08-01",
 "exportArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test/
export/01695097218000-d6299cbd",
 "startTime": "2023-09-19T04:20:18.000Z",
 "endTime": "2023-09-19T04:40:24.780Z",
 "tableArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test",
 "tableId": "b116b490-6460-4d4a-9a6b-5d360abf4fb3",
```

```
 "exportFromTime": "2023-09-18T17:00:00.000Z",
 "exportToTime": "2023-09-19T04:00:00.000Z",
 "s3Bucket": "jason-exports",
 "s3Prefix": "20230919-prefix",
 "s3SseAlgorithm": "AES256",
 "s3SseKmsKeyId": null,
 "manifestFilesS3Key": "20230919-prefix/AWSDynamoDB/01693685934212-ac809da5/manifest-
files.json",
 "billedSizeBytes": 20901239349,
 "itemCount": 169928274,
 "outputFormat": "DYNAMODB_JSON",
 "outputView": "NEW_AND_OLD_IMAGES",
 "exportType": "INCREMENTAL_EXPORT"
}
```

파일 매니페스트

`manifest-files.json` 파일에는 내보낸 테이블 데이터를 포함하는 파일에 대한 정보가 포함되어 있습니다. 파일은 JSON 라인 형식이므로 줄 바꿈이 항목 구분 기호로 사용됩니다. 다음 예제에서는 파일 매니페스트에 있는 한 데이터 파일의 세부 정보가 가독성을 향상하기 위해 여러 줄 형식으로 되어 있습니다.

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/data/sgad6417s6vss4p7owp0471bcq.json.gz"
}
```

데이터 파일

DynamoDB는 테이블 데이터를 DynamoDB JSON 및 Amazon Ion의 두 가지 형식으로 내보낼 수 있습니다. 선택하는 형식과 관계없이 데이터는 키로 명명되는 여러 압축 파일에 기록됩니다. 이러한 파일도 `manifest-files.json` 파일에 나열됩니다.

증분 내보내기를 위한 데이터 파일은 모두 S3 버킷의 공통 데이터 폴더에 포함되어 있습니다. 매니페스트 파일은 내보내기 ID 폴더 아래에 있습니다.

```
amzn-s3-demo-bucket/DestinationPrefix
.
### AWSDynamoDB
    ### 01693685934212-ac809da5     // an incremental export ID
```

```
    #   ### manifest-files.json     // manifest points to files under 'data' folder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### _started                // empty file for permission check
    ### 01693686034521-ac809da5
    #   ### manifest-files.json
    #   ### manifest-files.checksum
    #   ### manifest-summary.json
    #   ### manifest-summary.md5
    #   ### _started
    ### data                        // stores all the data files for incremental
 exports
    #   ### sgad6417s6vss4p7owp0471bcq.json.gz
    #   ...
```

파일을 내보낼 때 각 항목의 출력에는 해당 항목이 테이블에서 업데이트된 시간을 나타내는 타임스탬프와 해당 항목이 `insert`, `update`, `delete` 중 어떤 작업인지 나타내는 데이터 구조가 포함됩니다. 타임스탬프는 내부 시스템 시계를 기반으로 하며 애플리케이션 시계에 따라 달라질 수 있습니다. 증분 내보내기의 경우 출력 구조에 두 가지 내보내기 보기 유형(새 이미지와 이전 이미지 또는 새 이미지만) 중에서 선택할 수 있습니다.

- 새 이미지는 항목의 최신 상태를 제공합니다.
- 이전 이미지는 지정된 시작 날짜 및 시간 바로 이전의 항목 상태를 제공합니다.

내보내기 기간 내에 항목이 어떻게 변경되었는지 확인하려는 경우 보기 유형이 유용할 수 있습니다. 이는 특히 다운스트림 시스템에 DynamoDB 파티션 키와 동일하지 않은 파티션 키가 있는 경우 다운스트림 시스템을 효율적으로 업데이트하는 데에도 유용할 수 있습니다.

출력 구조를 보면 증분 내보내기 출력의 항목이 `insert`, `update`, `delete` 중 어느 것인지 추론할 수 있습니다. 두 내보내기 보기 유형 모두의 증분 내보내기 구조와 해당 작업이 아래 표에 요약되어 있습니다.

| Operation | 새 이미지만 | 새 이미지와 이전 이미지 |
|---|---|---|
| Insert | 키 + 새 이미지 | 키 + 새 이미지 |
| 업데이트 | 키 + 새 이미지 | 키 +새 이미지 +이전 이미지 |
| 삭제 | 키 | 키 +이전 이미지 |

| Operation | 새 이미지만 | 새 이미지와 이전 이미지 |
| --- | --- | --- |
| Insert + delete | 출력 없음 | 출력 없음 |

### DynamoDB JSON

DynamoDB JSON 형식의 테이블 내보내기는 항목의 쓰기 시간을 나타내는 메타데이터 타임스탬프와 항목의 키 및 값으로 구성됩니다. 다음은 새 이미지와 이전 이미지의 내보내기 보기 유형을 사용하는 DynamoDB JSON 출력의 예를 보여줍니다.

```
// Ex 1: Insert
//   An insert means the item did not exist before the incremental export window
//   and was added during the incremental export window

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#100"
     }
   },
   "NewImage": {
     "PK": {
       "S": "CUST#100"
     },
     "FirstName": {
       "S": "John"
     },
     "LastName": {
       "S": "Don"
     }
   }
}

// Ex 2: Update
//   An update means the item existed before the incremental export window
//   and was updated during the incremental export window.
//   The OldImage would not be present if choosing "New images only".

{
```

```
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#200"
     }
   },
   "OldImage": {
     "PK": {
       "S": "CUST#200"
     },
     "FirstName": {
       "S": "Mary"
     },
     "LastName": {
       "S": "Grace"
     }
   },
   "NewImage": {
     "PK": {
       "S": "CUST#200"
     },
     "FirstName": {
       "S": "Mary"
     },
     "LastName": {
       "S": "Smith"
     }
   }
}

// Ex 3: Delete
//   A delete means the item existed before the incremental export window
//   and was deleted during the incremental export window
//   The OldImage would not be present if choosing "New images only".

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#300"
```

```
      }
    },
    "OldImage": {
      "PK": {
        "S": "CUST#300"
      },
      "FirstName": {
        "S": "Jose"
      },
      "LastName": {
        "S": "Hernandez"
      }
    }
}

// Ex 4: Insert + Delete
//   Nothing is exported if an item is inserted and deleted within the
//   incremental export window.
```

### Amazon Ion

Amazon Ion은 대규모 서비스 중심 아키텍처를 엔지니어링할 때 매일 직면하는 신속한 개발, 분리, 효율성 문제를 해결하기 위해 빌드된 서식 있는 자기 기술형 계층적 데이터 직렬화 형식입니다. DynamoDB는 JSON의 상위 집합인 Ion 텍스트 형식으로 테이블 데이터를 내보낼 수 있습니다.

테이블을 Ion 형식으로 내보내면 테이블에서 사용되는 DynamoDB 데이터 유형이 Ion 데이터 유형에 매핑됩니다. DynamoDB 세트는 Ion 형식 주석을 사용하여 소스 테이블에서 사용되는 데이터 유형을 구분합니다.

다음 표에는 DynamoDB 데이터 형식과 Ion 데이터 형식의 매핑이 나와 있습니다.

| DynamoDB 데이터 형식 | Ion 표현 |
| --- | --- |
| 문자열(S) | 문자열 |
| Boolean(BOOL) | bool |
| 숫자(N) | decimal |
| 이진수(B) | blob |

| DynamoDB 데이터 형식 | Ion 표현 |
| --- | --- |
| 세트(SS, NSS, BS) | list(유형 주석 $dynamodb_SS, $dynamodb_NS 또는 $dynamodb_BS 사용) |
| 목록 | list |
| 맵 | struct |

Ion 내보내기의 항목은 줄 바꿈으로 구분됩니다. 각 줄은 Ion 버전 마커로 시작되고 그 다음에 Ion 형식의 항목이 표시됩니다. 다음 예제에서는 Ion 내보내기의 항목이 가독성을 향상하기 위해 여러 줄 형식으로 되어 있습니다.

```
$ion_1_0 {
    Record:{
        Keys:{
             ISBN:"333-3333333333"
         },
        Metadata:{
            WriteTimestampMicros:1684374845117899.
        },
        OldImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:false,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        },
        NewImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            Dimensions:"8.5 x 11.0 x 1.5",
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:true,
            PageCount:6d2,
            Price:2d3,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        }
    }
```

```
}
```

# Amazon SageMaker Lakehouse와의 DynamoDB 제로 ETL 통합

Amazon SageMaker Lakehouse와의 DynamoDB 제로 ETL 통합을 사용하면 DynamoDB 데이터를 Amazon SageMaker Lakehouse에 자동으로 복제하여 사용자 지정 데이터 이동 파이프라인을 구축할 필요가 없습니다. 이 코드 없는 통합을 통해 고객은 DynamoDB 테이블 용량을 사용하지 않고 Amazon SageMaker Lakehouse를 사용하여 DynamoDB 데이터에 대해 분석 워크로드를 실행할 수 있습니다. 통합은 테이블에서 데이터를 자동으로 내보내고 일반적으로 15~30분 이내에 대상을 최신 상태로 유지합니다.

주제


- Amazon SageMaker Lakehouse와의 DynamoDB 제로 ETL 통합


## Amazon SageMaker Lakehouse와의 DynamoDB 제로 ETL 통합

DynamoDB 테이블과 Amazon SageMaker Lakehouse 간에 통합을 설정하려면 AWS Glue가 소스의 데이터에 액세스하고 대상에 쓰는 데 사용하는 IAM 역할 구성, 중간 또는 대상 위치의 데이터를 암호화하기 위한 KMS 키 사용 등과 같은 사전 조건이 필요합니다.

주제


- Amazon SageMaker Lakehouse와의 DynamoDB 제로 ETL 통합을 만들기 이전의 사전 조건
- Amazon SageMaker Lakehouse와의 DynamoDB 제로 ETL 통합 만들기
- 통합에 대한 CloudWatch 지표 보기


### Amazon SageMaker Lakehouse와의 DynamoDB 제로 ETL 통합을 만들기 이전의 사전 조건

DynamoDB 소스와 제로 ETL 통합을 구성하려면 AWS Glue가 DynamoDB 테이블에서 데이터에 액세스하고 내보낼 수 있도록 허용하는 리소스 기반 액세스(RBAC) 정책을 설정해야 합니다. 정책에는 특정 AWS 계정 및 리전에 대한 액세스를 제한하는 조건과 함께 `ExportTableToPointInTime`, `DescribeTable` 및 `DescribeExport`와 같은 특정 권한이 포함되어야 합니다. 자세한 내용은 Amazon DynamoDB 소스 구성을 참조하세요.

테이블에 대해 시점 복구(PITR)를 활성화해야 하며 AWS CLI 명령을 사용하여 정책을 적용할 수 있습니다. 보다 제한적인 액세스 제어를 위해 전체 통합 ARN을 지정하여 정책을 더욱 세부적으로 조정할 수 있습니다. 자세한 내용은 제로 ETL 통합을 설정하기 위한 사전 조건을 참조하세요.

## Amazon SageMaker Lakehouse와의 DynamoDB 제로 ETL 통합 만들기

통합 사전 조건을 완료한 후 아래 지침에 따라 제로 ETL 통합을 생성, 수정 또는 삭제할 수 있습니다.

통합 생성

통합 생성하기

1. AWS Management Console에 로그인하고 Amazon DynamoDB 콘솔(https://console.aws.amazon.com/dynamodbv2)을 엽니다.
2. 탐색 창에서 통합을 선택합니다.
3. Amazon SageMaker Lakehouse와의 제로 ETL 통합 생성을 선택한 후 다음을 선택합니다.
4. 통합을 만들려면 통합 생성을 참조하세요.
5. 통합을 수정하려면 통합 수정을 참조하세요.
6. 통합을 삭제하려면 통합 삭제를 참조하세요.
7. 교차 계정 통합을 설정하려면 교차 계정 통합 설정을 참조하세요.

대상 Amazon S3 테이블에서 압축 활성화

압축을 활성화하여 Amazon Athena의 쿼리 성능을 개선할 수 있습니다.

먼저 필요한 IAM 역할 구성을 포함하여 압축 리소스에 대한 사전 조건 설정을 완료합니다. 자세한 IAM 역할 구성 단계는 Lake Formation 설명서를 참조하세요. 압축을 위한 테이블 최적화를 참조하세요.

통합 중에 만들어진 AWS Glue 테이블에서 압축을 활성화하려면 Lake Formation 압축 활성화 프로세스를 따릅니다. 이렇게 하면 테이블의 성능과 쿼리 효율성을 최적화하는 데 도움이 됩니다.

## 통합에 대한 CloudWatch 지표 보기

통합이 완료되면 각 AWS Glue 작업에 대해 계정에 생성된 CloudWatch 지표와 EventBridge 알림을 볼 수 있습니다. 자세한 내용은 통합 모니터링을 참조하세요.

# Amazon OpenSearch Service와 Amazon DynamoDB의 제로 ETL 통합

Amazon DynamoDB는 OpenSearch Ingestion용 DynamoDB 플러그인을 통해 Amazon OpenSearch Service와의 제로 ETL 통합을 제공합니다. Amazon OpenSearch Ingestion은 코드를 작성하지 않고도 Amazon OpenSearch Service로 데이터를 수집할 수 있는 완전관리형 환경을 제공합니다.

OpenSearch Ingestion용 DynamoDB 플러그인을 사용하면 하나 이상의 DynamoDB 테이블을 하나 이상의 OpenSearch Service 인덱스에 대한 수집 소스로 사용할 수 있습니다. AWS Management Console에서 OpenSearch Ingestion 또는 DynamoDB 통합에서 DynamoDB를 소스로 사용하여 OpenSearch Ingestion 파이프라인을 찾아보고 구성할 수 있습니다.

- OpenSearch Ingestion 시작 안내서를 따라 OpenSearch Ingestion을 시작하세요.
- OpenSearch Ingestion용 DynamoDB 플러그인 설명서에서 DynamoDB 플러그인의 사전 요구 사항 및 모든 구성 옵션에 대해 알아보세요.

## 작동 방식

플러그인은 Amazon S3로 DynamoDB 내보내기를 사용하여 OpenSearch에 로드할 초기 스냅샷을 생성합니다. 스냅샷이 로드된 후 플러그인은 DynamoDB Streams를 사용하여 추가 변경 사항을 거의 실시간으로 복제합니다. 모든 항목은 OpenSearch Ingestion에서 이벤트로 처리되며 프로세서 플러그인을 사용하여 수정할 수 있습니다. 속성을 삭제하거나 복합 속성을 만든 다음 경로를 통해 다른 인덱스로 보낼 수 있습니다.

Amazon S3로 내보내기를 사용하려면 시점 복구(PITR)를 활성화해야 합니다. 또한 새 이미지 및 기존 이미지 옵션을 선택한 상태로 DynamoDB Streams를 활성화해야 사용할 수 있습니다. 내보내기 설정을 제외하면 스냅샷을 만들지 않고도 파이프라인을 생성할 수 있습니다.

스트림 설정을 제외하면 스냅샷만 있고 업데이트는 없는 파이프라인을 생성할 수도 있습니다. 플러그인은 테이블의 읽기 또는 쓰기 처리량을 사용하지 않으므로 프로덕션 트래픽에 영향을 주지 않고 안전하게 사용할 수 있습니다. 스트림의 병렬 소비자 수에는 제한이 있으며 이 통합 또는 다른 통합을 생성하기 전에 이 제한을 고려해야 합니다. 다른 고려 사항은 the section called “통합 모범 사례” 섹션을 참조하세요.

단순 파이프라인의 경우 단일 OpenSearch Compute Unit(OCU)은 초당 약 1MB의 쓰기를 처리할 수 있습니다. 이는 약 1,000개의 쓰기 요청 단위(WCU)에 해당합니다. 파이프라인의 복잡성 및 기타 요인에 따라 이보다 많거나 적을 수 있습니다.

OpenSearch Ingestion은 복구할 수 없는 오류를 유발하는 이벤트에 대해 DLQ(Dead Letter Queue)를 지원합니다. 또한 파이프라인은 DynamoDB, 파이프라인 또는 Amazon OpenSearch Service에서 서비스가 중단되더라도 사용자 개입 없이 중단된 지점부터 재개할 수 있습니다.

중단이 24시간 넘게 지속되면 업데이트가 손실될 수 있습니다. 하지만 파이프라인은 가용성이 복원된 후에도 여전히 사용 가능했던 업데이트를 계속 처리합니다. DLQ(Dead Letter Queue)에 있는 경우가 아니라면 삭제된 이벤트로 인한 불규칙성을 수정하기 위해 인덱스를 새로 빌드해야 합니다.

플러그인에 대한 모든 설정 및 세부 정보는 OpenSearch Ingestion DynamoDB 플러그인 설명서를 참조하세요.

## 콘솔을 통한 통합 생성 환경

DynamoDB와 OpenSearch Service는 AWS Management Console에서 통합된 경험을 제공하므로 시작 프로세스가 간소화됩니다. 이 단계를 거치면 서비스가 자동으로 DynamoDB 청사진을 선택하고 적절한 DynamoDB 정보를 추가합니다.

통합을 생성하려면 OpenSearch Ingestion 시작 안내서를 따르세요. Step 3: Create a pipeline에 도달하면 1단계와 2단계를 다음 단계로 대체하세요.

1. DynamoDB 콘솔로 이동합니다.
2. 왼쪽 탐색 창에서 통합을 선택합니다.
3. OpenSearch에 복제할 DynamoDB 테이블을 선택합니다.
4. 생성(Create)을 선택합니다.

여기부터 자습서의 나머지 부분을 계속 진행할 수 있습니다.

## 다음 단계

DynamoDB가 OpenSearch Service와 통합되는 방식을 더 잘 이해하려면 다음을 참조하세요.

- Getting started with Amazon OpenSearch Ingestion
- DynamoDB 플러그인 구성 및 요구 사항

# 인덱스의 영향을 미치는 변경 사항 처리

OpenSearch는 인덱스에 새 속성을 동적으로 추가할 수 있습니다. 하지만 지정된 키에 매핑 템플릿을 설정한 후에는 추가 조치를 취하여 변경해야 합니다. 또한 변경으로 인해 DynamoDB 테이블의 모든 데이터를 재처리해야 하는 경우 새 내보내기를 시작하기 위한 조치를 취해야 합니다.

> **Note**
>
> 이러한 모든 옵션에서 DynamoDB 테이블에 지정한 매핑 템플릿과의 형식 충돌이 있는 경우 여전히 문제가 발생할 수 있습니다. DLQ(Dead Letter Queue)가 활성화되어 있는지 확인하세요(개발 중인 경우에도). 이렇게 하면 OpenSearch의 인덱스로 인덱싱될 때 충돌을 일으키는 레코드에 어떤 문제가 있는지 더 쉽게 이해할 수 있습니다.

주제


- 작동 방식
- 인덱스 삭제 및 파이프라인 재설정(파이프라인 중심 옵션)
- 인덱스 재생성 및 파이프라인 재설정(인덱스 중심 옵션)
- 새 인덱스 생성 및 싱크(온라인 옵션)
- 유형 충돌 방지 및 디버깅을 위한 모범 사례


## 작동 방식

인덱스에 대한 영향을 미치는 변경 사항을 처리할 때 취해진 조치에 대한 간략한 개요는 다음과 같습니다. 이어지는 섹션의 단계별 절차를 참조하세요.

- 파이프라인 중지 및 시작: 이 옵션은 파이프라인 상태를 재설정하고 파이프라인은 새로운 전체 내보내기로 다시 시작됩니다. 비파괴적이므로 DynamoDB의 인덱스 또는 데이터를 삭제하지 않습니다. 이 작업을 수행하기 전에 새 인덱스를 생성하지 않으면 내보내기 시 현재 `_version`보다 오래된 문서를 인덱스에 삽입하려고 하기 때문에 버전 충돌로 인한 오류가 많이 발생할 수 있습니다. 이 오류는 무시할 수 있습니다. 중지된 상태의 파이프라인 요금은 청구되지 않습니다.
- 파이프라인 업데이트: 이 옵션은 상태를 잃지 않고 블루/그린 방식으로 파이프라인의 구성을 업데이트합니다. 파이프라인을 크게 변경하는 경우(예: 기존 인덱스에 새 경로, 인덱스 또는 키 추가) 파이프라인을 완전히 재설정하고 인덱스를 다시 생성해야 할 수 있습니다. 이 옵션은 전체 내보내기를 수행하지 않습니다.

- 인덱스 삭제 및 재생성: 이 옵션은 인덱스의 데이터 및 매핑 설정을 제거합니다. 영향을 미치는 변경을 매핑에 수행하기 전에 먼저 이 작업을 수행해야 합니다. 이는 인덱스를 다시 만들고 동기화할 때까지 인덱스를 사용하는 모든 애플리케이션에 영향을 미칩니다. 인덱스를 삭제해도 새로 내보내기는 시작되지 않습니다. 파이프라인을 업데이트한 후에만 인덱스를 삭제해야 합니다. 그렇지 않으면 설정을 업데이트하기 전에 인덱스가 다시 생성될 수 있습니다.

## 인덱스 삭제 및 파이프라인 재설정(파이프라인 중심 옵션)

아직 개발 중인 경우 이 방법이 가장 빠른 옵션인 경우가 많습니다. OpenSearch Service에서 인덱스를 삭제한 다음 파이프라인을 중지했다가 시작하여 모든 데이터를 새로 내보내세요. 이렇게 하면 매핑 템플릿이 기존 인덱스와 충돌하는 일이 없고, 불완전하게 처리된 테이블로 인한 데이터 손실도 없습니다.

1. AWS Management Console을 사용하거나 AWS CLI 또는 SDK에서 StopPipeline API 작업을 사용하여 파이프라인을 중지합니다.
2. 새 변경 사항으로 파이프라인 구성을 업데이트합니다.
3. REST API 직접 호출 또는 OpenSearch 대시보드를 통해 OpenSearch Service에서 인덱스를 삭제합니다.
4. 콘솔을 사용하거나 AWS CLI 또는 SDK에서 `StartPipeline` API 작업을 사용하여 파이프라인을 시작합니다.

   > **Note**
   >
   > 그러면 새 전체 내보내기가 시작되며 추가 비용이 발생합니다.

5. 새 인덱스를 생성하기 위해 새로 내보내기가 생성되므로 예상치 못한 문제가 있는지 모니터링합니다.
6. 인덱스가 OpenSearch Service의 예상과 일치하는지 확인합니다.

내보내기가 완료되고 스트림에서 읽기가 재개되면 이제 DynamoDB 테이블 데이터를 인덱스에서 사용할 수 있습니다.

## 인덱스 재생성 및 파이프라인 재설정(인덱스 중심 옵션)

이 방법은 DynamoDB에서 파이프라인을 재개하기 전에 OpenSearch Service의 인덱스 설계를 여러 번 반복해야 하는 경우에 적합합니다. 이는 개발 시 검색 패턴을 매우 빠르게 반복하고 각 반복 사이에 새 내보내기가 완료될 때까지 기다리지 않으려고 할 때 유용할 수 있습니다.

1. AWS Management Console을 사용하거나 AWS CLI 또는 SDK에서 StopPipeline API 작업을 직접 호출하여 파이프라인을 중지합니다.
2. OpenSearch에서 인덱스를 삭제하고 사용하려는 매핑 템플릿으로 다시 생성합니다. 일부 샘플 데이터를 수동으로 삽입하여 검색이 의도한 대로 작동하는지 확인할 수 있습니다. 샘플 데이터가 DynamoDB의 데이터와 충돌할 수 있는 경우 다음 단계로 넘어가기 전에 반드시 삭제해야 합니다.
3. 파이프라인에 인덱싱 템플릿이 있는 경우 이를 제거하거나 OpenSearch Service에서 이미 만든 템플릿으로 교체합니다. 인덱스 이름이 파이프라인의 이름과 일치하는지 확인하세요.
4. 콘솔을 사용하거나 AWS CLI 또는 SDK에서 `StartPipeline` API 작업을 직접 호출하여 파이프라인을 시작합니다.

   > **Note**
   >
   > 그러면 새 전체 내보내기가 시작되며 추가 비용이 발생합니다.

5. 새 인덱스를 생성하기 위해 새로 내보내기가 생성되므로 예상치 못한 문제가 있는지 모니터링합니다.

내보내기가 완료되고 스트림에서 읽기가 재개되면 이제 DynamoDB 테이블 데이터를 인덱스에서 사용할 수 있습니다.

### 새 인덱스 생성 및 싱크(온라인 옵션)

이 방법은 매핑 템플릿을 업데이트해야 하지만 현재 프로덕션에서 인덱스를 사용하고 있는 경우에 적합합니다. 이렇게 하면 완전히 새로운 인덱스가 만들어지는데, 동기화 및 검증을 마친 후 이 인덱스로 애플리케이션을 옮겨야 합니다.

> **Note**
>
> 그러면 스트림에 또 다른 소비자가 생깁니다. AWS Lambda 같은 다른 소비자 또는 글로벌 테이블이 있는 경우 문제가 될 수 있습니다. 새 인덱스를 로드할 용량을 확보하려면 기존 파이프라인의 업데이트를 일시 중지해야 할 수 있습니다.

1. 새 설정과 다른 인덱스 이름을 사용하여 새 파이프라인을 생성합니다.
2. 새 인덱스에 예상치 못한 문제가 있는지 모니터링합니다.
3. 애플리케이션을 새 인덱스로 전환합니다.
4. 모든 것이 제대로 작동하는지 확인한 후 기존 파이프라인을 중지하고 삭제합니다.

### 유형 충돌 방지 및 디버깅을 위한 모범 사례

- 형식 충돌이 있을 때 디버깅하기 쉽도록 항상 DLQ(Dead Letter Queue)를 사용하세요.
- 항상 매핑과 함께 인덱스 템플릿을 사용하고 `include_keys`를 설정합니다. OpenSearch Service는 새 키를 동적으로 매핑하지만 이로 인해 예기치 않은 동작(예: `GeoPoint`를 예상했지만 `string` 또는 `object`가 생성됨)이나 오류(예: `long`과 `float` 값이 혼합된 `number`)가 발생할 수 있습니다.
- 기존 인덱스를 프로덕션에서 계속 작동시켜야 하는 경우 이전의 인덱스 삭제 단계를 파이프라인 구성 파일에서 인덱스 이름을 바꾸는 것으로 대체할 수도 있습니다. 이렇게 하면 완전히 새로운 인덱스가 만들어집니다. 그런 다음 완료 후 새 인덱스를 가리키도록 애플리케이션을 업데이트해야 합니다.
- 프로세서로 해결하는 유형 전환 문제가 있는 경우 `UpdatePipeline`을 통해 테스트할 수 있습니다. 이렇게 하려면 DLQ(Dead Letter Queue)를 중지했다가 다시 시작하거나 처리하여 이전에 건너뛰었고 오류가 있는 문서를 수정해야 합니다.

## DynamoDB의 제로 ETL 통합 및 OpenSearch Service와 관련된 모범 사례

DynamoDB는 Amazon OpenSearch Service와 DynamoDB의 제로 ETL 통합을 제공합니다. 자세한 내용은 DynamoDB plugin for OpenSearch Ingestion과 Amazon OpenSearch Service의 구체적인 모범 사례를 참조하세요.

### 구성

- 검색을 수행해야 하는 데이터만 인덱싱하세요. 이를 구현하려면 항상 매핑 템플릿(`template_type: index_template` 및 `template_content`)과 `include_keys`를 사용하세요.
- 로그를 모니터링하여 유형 충돌과 관련된 오류가 있는지 확인하세요. OpenSearch Service에서는 지정된 키의 모든 값이 같은 유형이어야 합니다. 불일치가 있는 경우 예외가 발생합니다. 이러한 오류 중 하나가 발생하는 경우 프로세서를 추가하여 지정된 키가 항상 같은 값인지 확인할 수 있습니다.
- 일반적으로 `document_id` 값에는 `primary_key` 메타데이터 값을 사용하세요. OpenSearch Service에서 문서 ID는 DynamoDB의 프라이머리 키와 동일합니다. 프라이머리 키를 사용하면 문서를 쉽게 찾고 업데이트가 충돌 없이 일관되게 문서에 복제될 수 있습니다.

  도우미 함수 `getMetadata`를 사용하여 프라이머리 키(예: `document_id: "${getMetadata('primary_key')}"`)를 가져올 수 있습니다. 복합 기본 키를 사용하는 경우 도우미 함수가 두 키를 자동으로 연결합니다.
- 일반적으로 `action` 설정에는 `opensearch_action` 메타데이터 값을 사용하세요. 이렇게 하면 OpenSearch Service의 데이터가 DynamoDB의 최신 상태와 일치하도록 업데이트가 복제됩니다.

도우미 함수 `getMetadata`를 사용하여 프라이머리 키(예: `action: "${getMetadata('opensearch_action')}"`)를 가져올 수 있습니다. 필터링과 같은 사용 사례에 `dynamodb_event_name`을 통해 스트림 이벤트 유형을 가져올 수도 있습니다. 하지만 일반적으로 `action` 설정에는 사용하지 않는 것이 좋습니다.

## 관찰성

- 삭제된 이벤트를 처리하려면 OpenSearch 싱크에 항상 DLQ(Dead Letter Queue)를 사용하세요. DynamoDB는 일반적으로 OpenSearch Service보다 덜 구조화되어 있으며 예상치 못한 일이 발생할 가능성이 항상 있습니다. DLQ(Dead Letter Queue)를 사용하여 개별 이벤트를 복구하고 복구 프로세스를 자동화할 수 있습니다. 이렇게 하면 전체 인덱스를 다시 빌드할 필요가 없어집니다.
- 복제 지연이 예상한 시간을 초과하지 않도록 항상 알림을 설정하세요. 일반적으로 1분이 지나면 알림이 매우 시끄러워집니다. 이는 쓰기 트래픽이 얼마나 급증하는지와 파이프라인의 OpenSearch Compute Unit(OCU) 설정에 따라 달라질 수 있습니다.

  복제 지연이 24시간을 초과하면 스트림에서 이벤트가 삭제되기 시작하고 인덱스를 처음부터 완전히 다시 빌드하지 않는 한 정확성 문제가 발생합니다.

## 스케일링

- 파이프라인에 Auto Scaling을 사용하면 워크로드에 가장 적합하도록 OCU를 스케일 업 또는 스케일 다운할 수 있습니다.
- Auto Scaling을 사용하지 않는 프로비저닝된 처리량 테이블의 경우 쓰기 용량 단위(WCU)를 1,000으로 나눈 값을 기준으로 OCU를 설정하는 것이 좋습니다. 최소 OCU를 그 값보다 1개 적게 설정하고(최소 1개), 최대 OCU는 그 값보다 최소 1개 더 많게 설정합니다.
  - 공식:

    ```
    OCU_minimum = GREATEST((table_WCU / 1000) - 1, 1)
    OCU_maximum = (table_WCU / 1000) + 1
    ```

  - 예: 테이블에 2만 5,000개의 WCU가 프로비저닝되어 있습니다. 파이프라인의 OCU의 최솟값은 24(25,000/1,000 - 1), 최댓값은 최소 26(2,5000/1,000 + 1)으로 설정해야 합니다.
- Auto Scaling을 사용하는 프로비저닝된 처리량 테이블의 경우 최소 및 최대 WCU를 1,000으로 나눈 값을 기준으로 OCU를 설정하는 것이 좋습니다. 최소 OCU를 DynamoDB의 최솟값보다 1개 적게 설정하고, 최대 OCU를 DynamoDB의 최댓값보다 최소 1개 더 많게 설정합니다.

- 공식:

```
OCU_minimum = GREATEST((table_minimum_WCU / 1000) - 1, 1)
OCU_maximum = (table_maximum_WCU / 1000) + 1
```

- 예: 테이블에 최소 8,000에서 최대 1만 4,000이라는 Auto Scaling 정책이 있습니다. 파이프라인의 OCU의 최솟값은 7(8,000/1,000 - 1), 최댓값은 최소 15(14,000/1,000 + 1)로 설정해야 합니다.

- 온디맨드 처리량 테이블의 경우 초당 쓰기 요청 유닛의 일반적인 급증 및 급락을 기준으로 OCU를 설정하는 것이 좋습니다. 사용 가능한 집계에 따라 더 긴 기간의 평균을 구해야 할 수도 있습니다. 최소 OCU를 DynamoDB의 최솟값보다 1개 적게 설정하고, 최대 OCU를 DynamoDB의 최댓값보다 최소 1개 더 많게 설정합니다.
  - 공식:

```
# Assuming we have writes aggregated at the minute level
OCU_minimum = GREATEST((min(table_writes_1min) / (60 * 1000)) - 1, 1)
OCU_maximum = (max(table_writes_1min) / (60 * 1000)) + 1
```

  - 예: 테이블의 평균 급락은 초당 쓰기 요청 유닛 300개, 평균 급증은 4,300입니다. 파이프라인의 OCU의 최솟값은 1(300/1,000 - 1, 그러나 최소 1개), 최댓값은 5(4,300/1,000 + 1)로 설정해야 합니다.
- 대상 OpenSearch Service 인덱스를 확장하는 모범 사례를 따르세요. 인덱스의 규모가 필요한 것보다 작으면 DynamoDB에서의 수집 속도가 느려지고 지연이 발생할 수 있습니다.

> **Note**
>
> GREATEST는 인수 집합이 주어지면 값이 가장 큰 인수를 반환하는 SQL 함수입니다.

# DynamoDB와 Amazon EventBridge 통합

Amazon DynamoDB는 변경 데이터 캡처를 위한 DynamoDB Streams를 제공하므로, DynamoDB 테이블의 항목 수준 변경 사항을 캡처할 수 있습니다. DynamoDB Streams는 Lambda 함수를 간접 호출하여 변경 사항을 처리할 수 있어 다른 서비스 및 애플리케이션과의 이벤트 기반 통합이 가능합니다. DynamoDB Streams는 필터링도 지원하므로, 대상을 정한 효율적인 이벤트 처리가 가능합니다.

DynamoDB Streams는 샤드당 최대 2명의 동시 소비자를 지원하고 Lambda 이벤트 필터링을 통한 필터링을 지원하여 특정 기준과 일치하는 항목만 처리합니다. 2명 이상의 소비자를 지원해야 하는 요구

사항이 있는 고객도 있습니다. 변경 이벤트가 처리되기 전에 변경 이벤트를 보강하거나 고급 필터링 및 라우팅을 사용해야 하는 경우도 있습니다.

DynamoDB를 EventBridge와 통합하면 이러한 요구 사항을 지원할 수 있습니다.

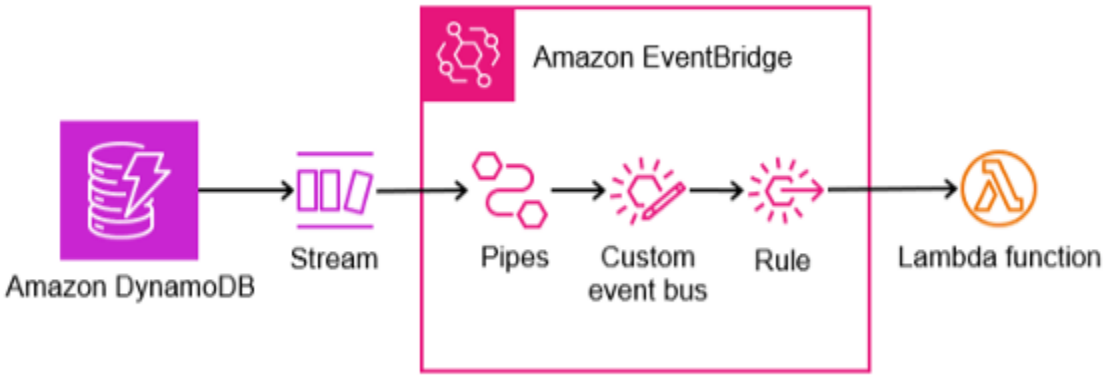
Amazon EventBridge는 이벤트를 사용하여 애플리케이션 구성 요소를 서로 연결하는 서버리스 서비스로, 확장 가능한 이벤트 기반 애플리케이션을 더 쉽게 구축할 수 있습니다. EventBridge는 EventBridge 파이프를 통해 Amazon DynamoDB와 기본적으로 통합되어 DynamoDB에서 EventBridge 버스로 데이터가 원활하게 흐릅니다. 그러면 해당 버스는 일련의 규칙과 대상을 통해 여러 애플리케이션 및 서비스로 팬아웃할 수 있습니다.

주제


- 작동 방식
- 콘솔을 통한 통합 생성
- 다음 단계


## 작동 방식

DynamoDB와 EventBridge 파이프를 통합하면 DynamoDB Streams를 사용하여 DynamoDB 테이블에서 시간 순서에 따라 항목 수준 변경 사항을 캡처합니다. 이러한 방식으로 캡처된 각 레코드에는 테이블에서 수정된 데이터가 포함됩니다.




EventBridge 파이프는 DynamoDB Streams의 이벤트를 소비하여 EventBridge 버스와 같은 대상으로 라우팅합니다. 이벤트 버스는 이벤트를 수신하여 표적(대상이라고도 함)에 전달하는 라우터입니다. 전송은 이벤트 내용과 일치하는 규칙을 기반으로 합니다. 파이프에는 필요에 따라 특정 이벤트를 필터링하고 이벤트 데이터가 대상으로 전송되기 전에 이벤트 데이터를 보강하는 기능도 포함되어 있습니다.

EventBridge는 여러 대상 유형을 지원하지만, 팬아웃 설계를 구현할 때 일반적으로 Lambda 함수를 대상으로 사용합니다. 다음 예제에서는 Lambda 함수 대상과의 통합을 보여줍니다.

## 콘솔을 통한 통합 생성

아래 단계에 따라 AWS Management Console을 통해 통합을 생성합니다.

1. DynamoDB 개발자 안내서의 스트림 활성화 섹션에 있는 단계를 따라 소스 테이블에서 DynamoDB Streams를 활성화합니다. 소스 테이블에서 DynamoDB Streams가 이미 활성화되어 있는 경우, 현재 소비자가 2명 미만인지 확인하세요. 소비 대상은 Lambda 함수, DynamoDB 전역 테이블, Amazon OpenSearch Service와의 Amazon DynamoDB 제로 ETL 통합 또는 DynamoDB Streams Kinesis 어댑터 등을 통해 스트림에서 직접 읽는 애플리케이션 등일 수 있습니다.
2. EventBridge 사용 설명서의 Amazon EventBridge 이벤트 버스 만들기 섹션에 있는 단계에 따라 EventBridge 이벤트 버스를 생성합니다.
   a. 이벤트 버스를 생성할 때 스키마 검색을 활성화합니다.
3. EventBridge 사용 설명서의 Amazon EventBridge 파이프 만들기 섹션에 있는 단계에 따라 EventBridge 파이프를 생성합니다.
   a. 소스를 구성할 때 소스 필드에서 DynamoDB를 선택하고 DynamoDB Streams 필드에서 소스 테이블 스트림의 이름을 선택합니다.
   b. 대상을 구성할 때 대상 서비스 필드에서 EventBridge 이벤트 버스를 선택하고 이벤트 버스를 대상으로 설정 필드에서 2단계에서 생성된 이벤트 버스를 선택합니다.
4. 소스 DynamoDB 테이블에 예제 항목을 작성하여 이벤트를 트리거합니다. 이렇게 하면 EventBridge가 예제 항목에서 스키마를 유추할 수 있습니다. 이 스키마를 사용하여 라우팅 이벤트에 대한 규칙을 생성할 수 있습니다. 예를 들어, 속성 오버로드가 포함된 설계 패턴을 구현하는 경우 정렬 키 값에 따라 다른 규칙을 트리거해야 할 수 있습니다. DynamoDB에 항목을 쓰는 방법에 대한 자세한 내용은 DynamoDB 개발자 안내서의 항목 및 속성 작업 섹션에서 확인할 수 있습니다.
5. Lambda 개발자 안내서의 Python을 사용하여 Lambda 함수 빌드 섹션에 있는 단계에 따라 대상으로 사용할 예제 Python Lambda 함수를 생성합니다. 함수를 생성할 때 아래 예제 코드를 사용하여 통합을 시연할 수 있습니다. 간접 호출되면 CloudWatch Logs에서 볼 수 있는 이벤트와 함께 수신된 `NewImage` 및 `OldImage`가 출력됩니다.

```
import json

def lambda_handler(event, context):
    dynamodb = event.get('detail', {}).get('dynamodb', {})
    new_image = dynamodb.get('NewImage')
    old_image = dynamodb.get('OldImage')
```

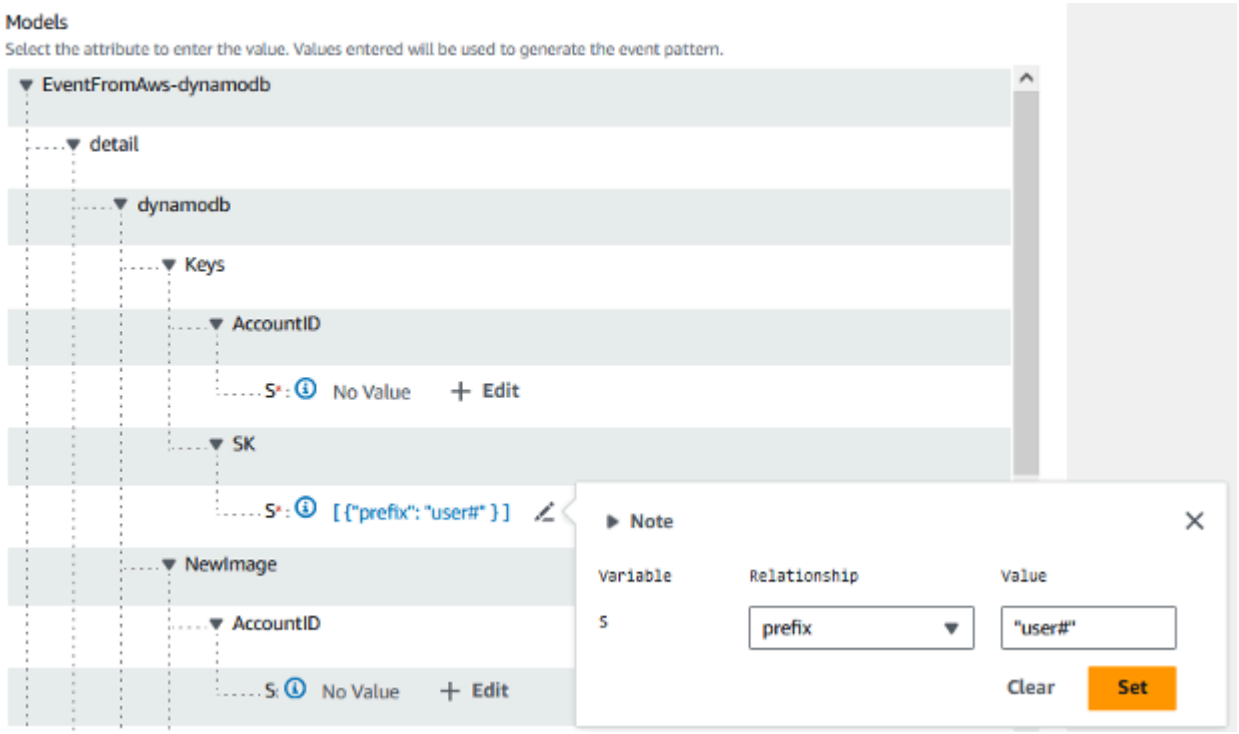
```
    if new_image:
        print("NewImage:", json.dumps(new_image, indent=2))
    if old_image:
        print("OldImage:", json.dumps(old_image, indent=2))

    return {'statusCode': 200, 'body': json.dumps(event)}
```

6. Create a rule 섹션의 단계에 따라 이벤트에 반응하는 새 Lambda 함수로 이벤트를 라우팅하는 EventBridge 규칙을 생성합니다.

   a. 규칙 세부 정보를 정의할 때는 2단계에서 생성한 이벤트 버스의 이름을 이벤트 버스로 선택하세요.

   b. 이벤트 패턴을 만들 때는 기존 스키마의 가이드를 따르세요. 여기에서 이벤트의 검색된 스키마 레지스트리와 검색된 스키마를 선택할 수 있습니다. 이를 통해 특정 속성과 일치하는 메시지만 라우팅하도록 사용 사례에 맞는 이벤트 패턴을 구성할 수 있습니다. 예를 들어, SK가 "user#"로 시작하는 DynamoDB 항목에서만 일치시키려면 다음과 같은 구성을 사용합니다.




   c. 스키마에 대한 패턴 설계를 완료한 후 JSON에서 이벤트 패턴 생성을 클릭합니다. 대신 DynamoDB Streams에 나타나는 모든 이벤트를 일치시키려면 이벤트 패턴에 다음 JSON을 사용하세요.

```
{
  "source": ["aws.dynamodb"]
```

```
}
```

d. 대상을 선택할 때는 AWS 서비스 가이드를 따르세요. 대상 선택 필드에서 'Lambda 함수'를 선택합니다. 함수 필드에서 5단계에서 생성한 Lambda 함수를 선택합니다.

7. 이제 EventBridge 사용 설명서의 이벤트 버스에서 스키마 검색 시작 또는 중지 섹션에 있는 단계에 따라 이벤트 버스에서 스키마 검색을 중지할 수 있습니다.
8. 두 번째 예제 항목을 소스 DynamoDB 테이블에 작성하여 이벤트를 트리거합니다. 각 단계에서 이벤트가 성공적으로 처리되었는지 검증하세요.
   a. EventBridge 사용 설명서의 Monitoring Amazon EventBridge 섹션에 따라 이벤트 버스의 CloudWatch 지표 PutEventsApproximateSuccessCount를 확인합니다.
   b. Lambda 개발자 안내서의 Lambda 함수 모니터링 및 문제 해결 섹션에 따라 Lambda 함수의 함수 로그를 확인합니다. Lambda 함수가 제공된 예제 코드를 사용하는 경우 CloudWatch Logs 로그 그룹에 출력된 DynamoDB Streams의 `NewImage` 및 `OldImage`를 확인해야 합니다.
   c. Lambda 개발자 안내서의 Lambda 함수 모니터링 및 문제 해결 섹션을 따라 Lambda 함수에 대한 오류 수 및 성공률(%) 지표를 확인합니다.

## 다음 단계

이 예제는 단일 Lambda 함수를 대상으로 하는 기본 통합을 제공합니다. 여러 규칙 생성, 여러 대상 생성, 다른 서비스와의 통합, 이벤트 보강과 같은 더 복잡한 구성을 효과적으로 이해하려면 전체 EventBridge 사용 안내서: Getting started with EventBridge를 참조하세요.

> **Note**
>
> 애플리케이션과 관련이 있을 수 있는 EventBridge 할당량을 숙지하세요. DynamoDB Streams의 용량은 테이블에 따라 규모가 조정되지만, EventBridge 할당량은 별개입니다. 대규모 애플리케이션에서 주의해야 할 일반적인 할당량은 초당 트랜잭션의 간접 호출 제한 한도와 초당 트랜잭션의 PutEvents 제한 한도입니다. 이러한 할당량은 대상으로 전송할 수 있는 간접 호출 수와 버스에 보낼 수 있는 초당 이벤트 수를 지정합니다.

# DynamoDB를 Amazon Managed Streaming for Apache Kafka와 통합

Amazon Managed Streaming for Apache Kafka(Amazon MSK)를 사용하면 완전관리형의 고가용성 Apache Kafka 서비스를 통해 스트리밍 데이터를 실시간으로 쉽게 수집하고 처리할 수 있습니다.

Apache Kafka는 스트리밍 데이터의 실시간 수집 및 처리에 최적화된 분산 데이터 스토어입니다. Kafka는 레코드 스트림을 처리하고, 레코드가 생성된 순서대로 레코드 스트림을 효과적으로 저장하고, 레코드 스트림을 게시 및 구독할 수 있습니다.

이러한 기능으로 인해 Apache Kafka는 실시간 스트리밍 데이터 파이프라인을 구축하는 데 자주 사용됩니다. 데이터 파이프라인은 한 시스템에서 다른 시스템으로 데이터를 안정적으로 처리하고 이동하며, 각기 다른 사용 사례를 지원하는 여러 데이터베이스를 쉽게 사용할 수 있도록 하여 목적별 데이터베이스 전략을 채택하는 데 중요한 역할을 할 수 있습니다.

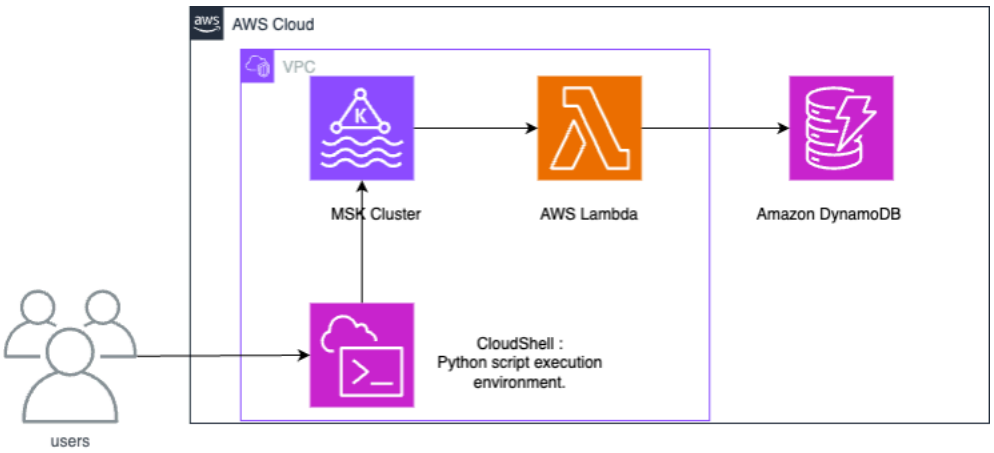
Amazon DynamoDB는 키-값 또는 문서 데이터 모델을 사용하고 일관된 한 자릿수 밀리초 성능과 함께 무한한 확장성이 필요한 애플리케이션을 지원하기 위해 이러한 데이터 파이프라인에서 일반적인 대상으로 사용됩니다.

주제


- 작동 방법
- Amazon MSK와 DynamoDB 간의 통합 설정
- 다음 단계


## 작동 방법

Amazon MSK와 DynamoDB 간의 통합은 Lambda 함수를 사용하여 Amazon MSK의 레코드를 사용하고 DynamoDB에 씁니다.




Lambda는 Amazon MSK의 새 메시지를 내부적으로 폴링한 다음 대상 Lambda 함수를 동기적으로 간접 호출합니다. Lambda 함수의 이벤트 페이로드에는 Amazon MSK의 메시지 배치가 포함되어 있습니다. Amazon MSK와 DynamoDB 간의 통합을 위해 Lambda 함수는 이러한 메시지를 DynamoDB에 씁니다.

# Amazon MSK와 DynamoDB 간의 통합 설정

> **Note**
>
> 다음 GitHub 리포지토리에서 이 예제에 사용된 리소스를 다운로드할 수 있습니다.

아래 단계는 Amazon MSK와 Amazon DynamoDB 간의 샘플 통합을 설정하는 방법을 보여줍니다. 이 예제는 사물 인터넷(IoT) 디바이스에서 생성되어 Amazon MSK로 수집된 데이터를 나타냅니다. 데이터가 Amazon MSK에 수집되면 Apache Kafka와 호환되는 분석 서비스 또는 타사 도구와 통합할 수 있으므로 다양한 분석 사용 사례가 가능합니다. DynamoDB도 통합하면 개별 디바이스 레코드의 키 값 조회가 가능합니다.

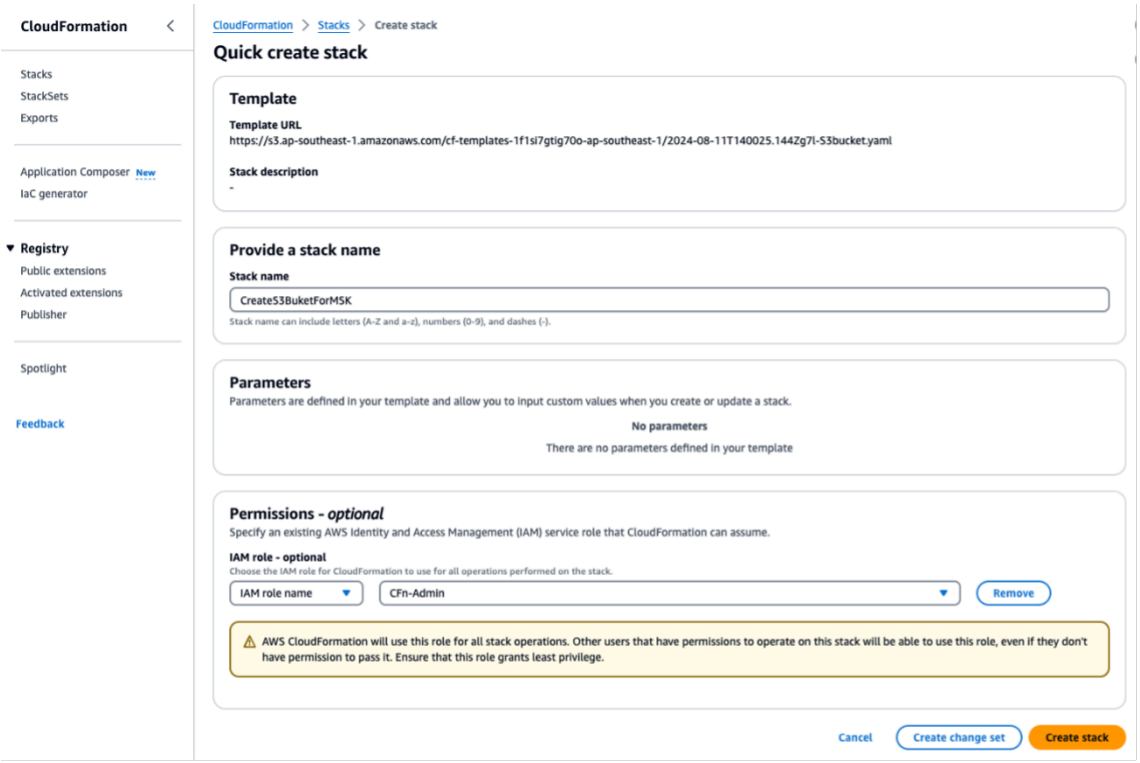
이 예제에서는 Python 스크립트가 Amazon MSK에 IoT 센서 데이터를 쓰는 방법을 보여줍니다. 그런 다음 Lambda 함수는 파티션 키 '`deviceid`'를 사용하여 DynamoDB에 항목을 씁니다.

제공된 CloudFormation 템플릿은 Amazon S3 버킷, Amazon VPC, Amazon MSK 클러스터 및 데이터 작업 테스트를 위한 AWS CloudShell의 리소스를 만듭니다.

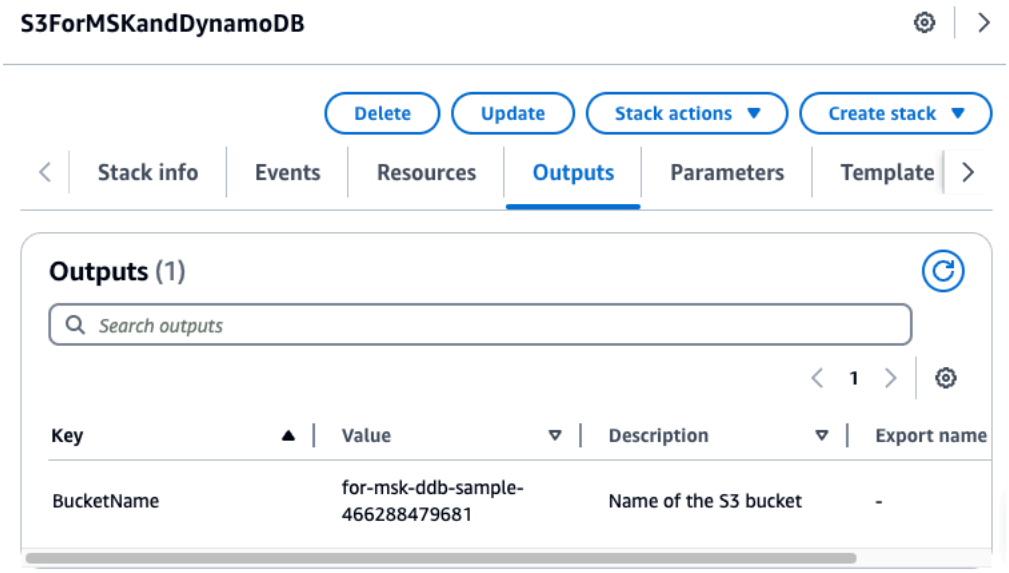
테스트 데이터를 만들려면 Amazon MSK 주제를 만든 다음 DynamoDB 테이블을 만듭니다. 관리 콘솔에서 Session Manager를 사용하여 CloudShell의 운영 체제에 로그인하고 Python 스크립트를 실행할 수 있습니다.

CloudFormation 템플릿을 실행한 후 다음 작업을 수행하여 이 아키텍처 구축을 완료할 수 있습니다.

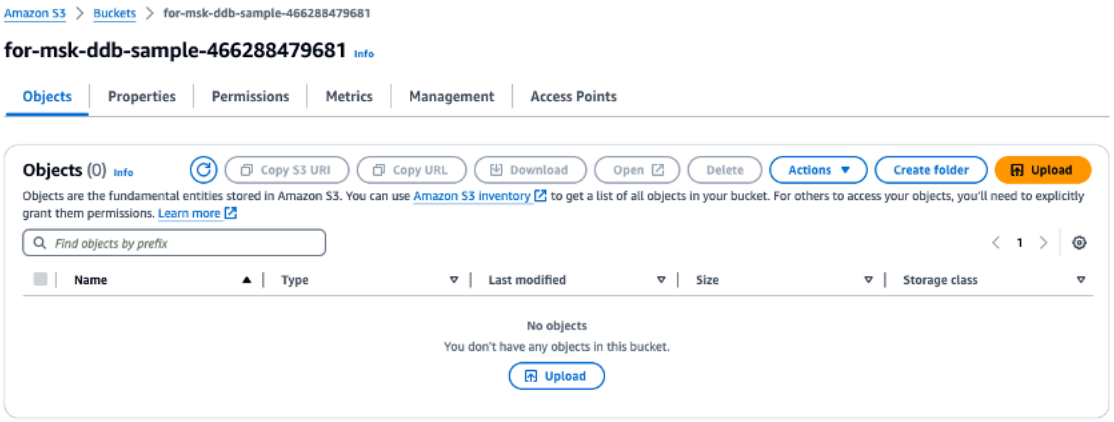
1. CloudFormation 템플릿 `S3bucket.yaml`을 실행하여 S3 버킷을 만듭니다. 후속 스크립트 또는 작업의 경우 동일한 리전에서 실행합니다. CloudFormation 스택의 이름으로 `ForMSKTestS3`를 입력합니다.



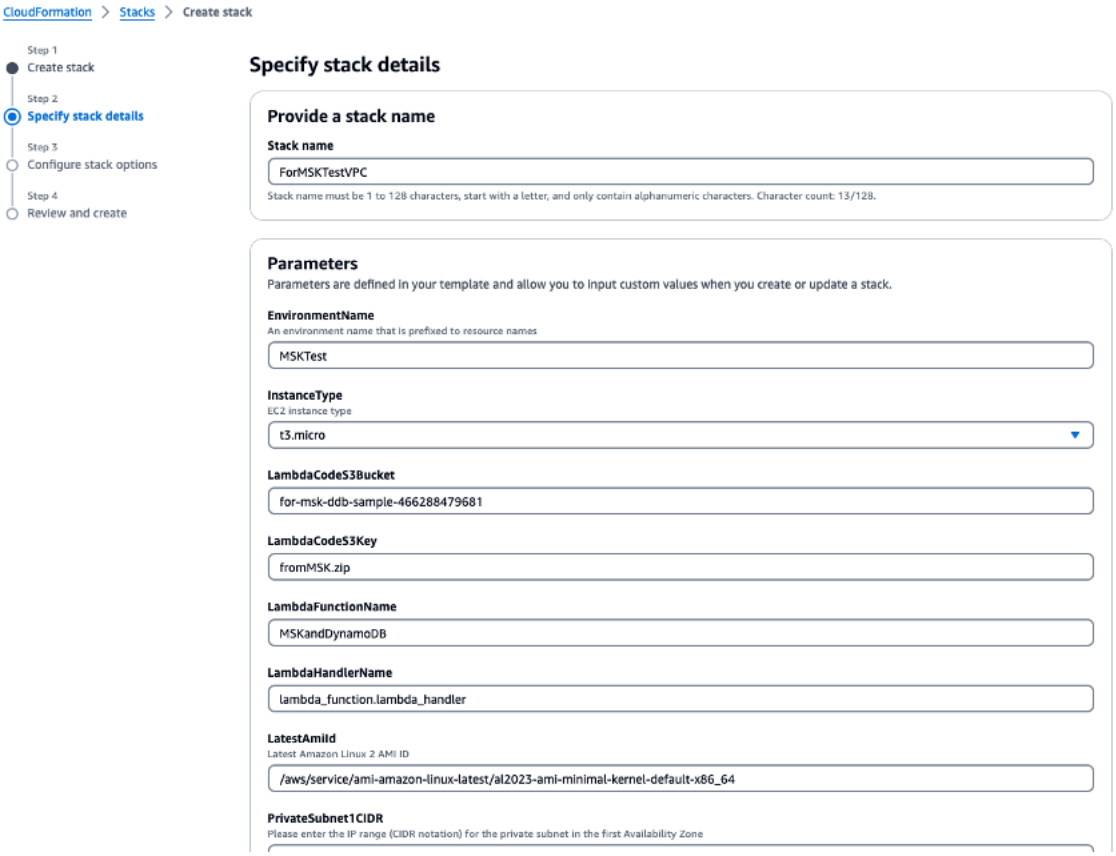

이 작업이 완료되면 출력 아래에 있는 S3 버킷 이름 출력을 기록해 둡니다. 3단계에서 이 이름이 필요합니다.




2. 다운로드한 ZIP 파일 `fromMSK.zip`을 방금 만든 S3 버킷으로 업로드합니다.



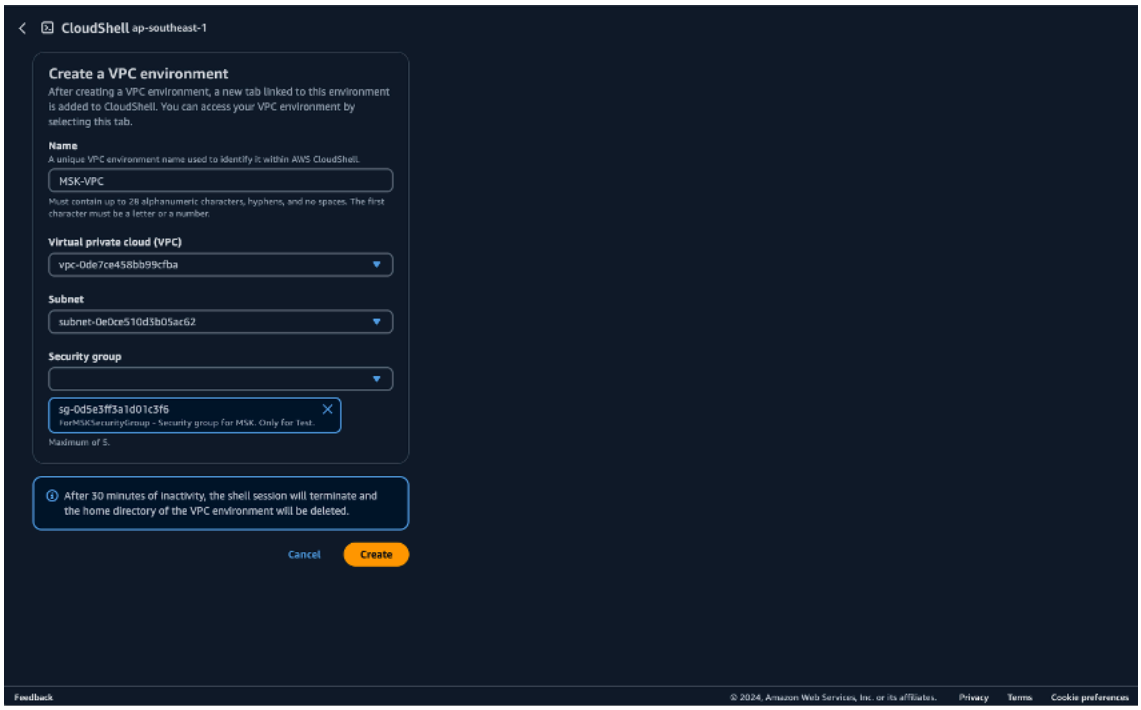

3. CloudFormation 템플릿 `VPC.yaml`을 실행하여 VPC, Amazon MSK 클러스터 및 Lambda 함수를 만듭니다. S3 버킷을 요청하는 파라미터 입력 화면이 표시되면 1단계에서 생성한 S3 버킷 이름을 입력합니다. CloudFormation 스택 이름을 `ForMSKTestVPC`로 설정합니다.




4. CloudShell에서 Python 스크립트를 실행하기 위한 환경을 준비합니다. AWS Management Console에서 CloudShell을 사용할 수 있습니다. CloudShell 사용에 대한 자세한 내용은 AWS CloudShell 시작하기를 참조하세요. CloudShell을 시작한 후 Amazon MSK 클러스터에 연결하기 위해 방금 만든 VPC에 속하는 CloudShell을 만듭니다. 프라이빗 서브넷에서 CloudShell을 만듭니다. 다음 필드를 입력합니다.

1. 이름 - 원하는 이름으로 설정할 수 있습니다. 예를 들어, MSK-VPC가 될 수 있습니다.
2. VPC - MSKTest를 선택합니다.
3. 서브넷 - MSKTest 프라이빗 서브넷(AZ1)을 선택합니다.
4. SecurityGroup - ForMSKSecurityGroup을 선택합니다.



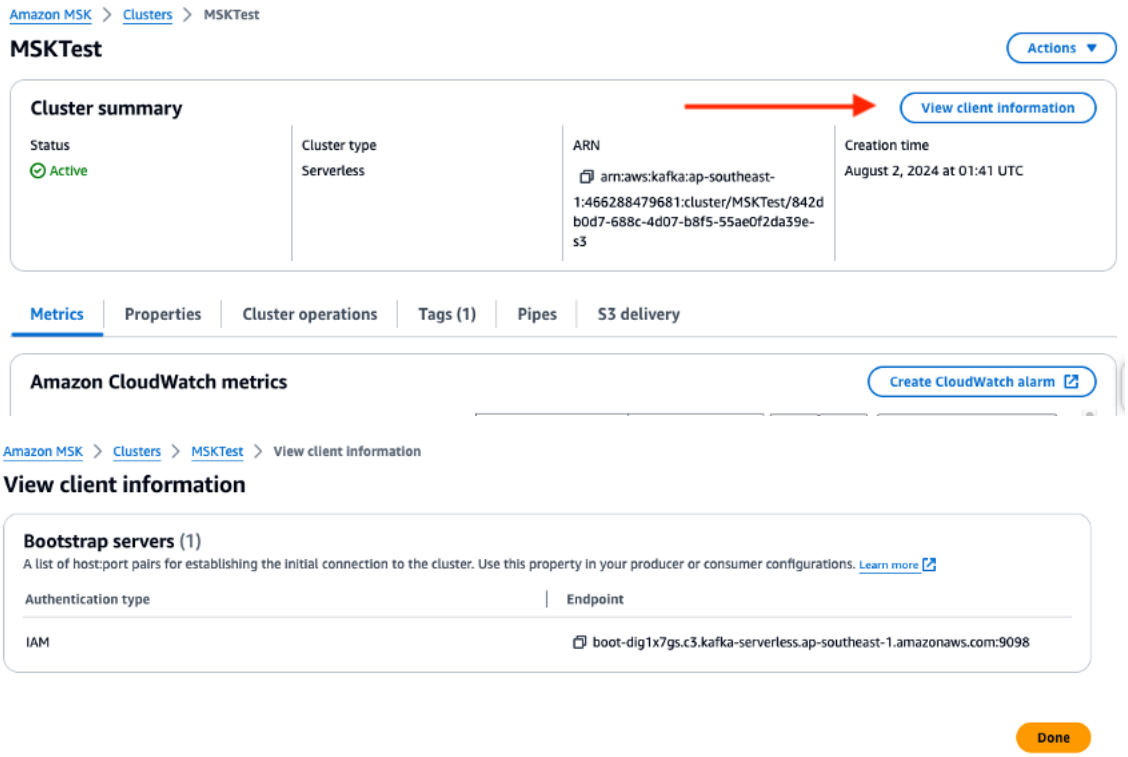
프라이빗 서브넷에 속하는 CloudShell이 시작되면 다음 명령을 실행합니다.

```
pip install boto3 kafka-python aws-msk-iam-sasl-signer-python
```

5. S3 버킷에서 Python 스크립트를 다운로드합니다.

```
aws s3 cp s3://[YOUR-BUCKET-NAME]/pythonScripts.zip ./
unzip pythonScripts.zip
```

6. 관리 콘솔을 확인하고 Python 스크립트에서 브로커 URL 및 리전 값에 대한 환경 변수를 설정합니다. 관리 콘솔에서 Amazon MSK 클러스터 브로커 엔드포인트를 확인합니다.




7. CloudShell에서 환경 변수를 설정합니다. 미국 서부(오리건)를 사용하는 경우:

```
export AWS_REGION="us-west-2"
export MSK_BROKER="boot-YOURMSKCLUSTER.c3.kafka-serverless.ap-
southeast-1.amazonaws.com:9098"
```

8. 다음 Python 스크립트를 실행합니다.

Amazon MSK 주제 만들기:

```
python ./createTopic.py
```

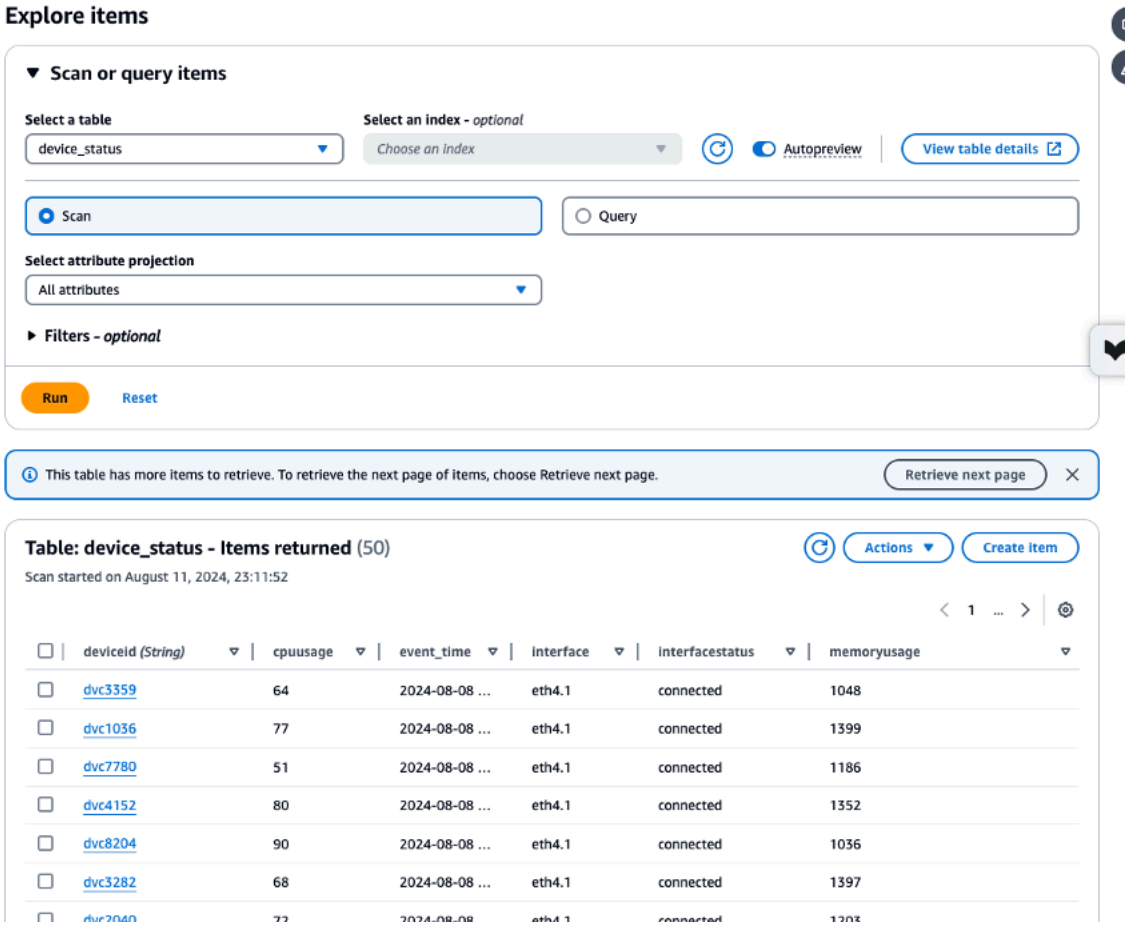
DynamoDB 테이블 만들기:

```
python ./createTable.py
```

Amazon MSK 주제에 테스트 데이터 쓰기:

```
python ./kafkaDataGen.py
```

9. 생성된 Amazon MSK, Lambda 및 DynamoDB 리소스에 대한 CloudWatch 지표를 확인하고 DynamoDB Data Explorer로 `device_status` 테이블에 저장된 데이터를 확인하여 모든 프로세스가 올바르게 실행되었는지 확인합니다. 각 프로세스가 오류 없이 실행되는 경우 CloudShell에서 Amazon MSK로 쓰인 테스트 데이터가 DynamoDB에도 쓰였는지 확인할 수 있습니다.



10. 이 예제를 마치면 이 자습서에서 만든 리소스를 삭제합니다. `ForMSKTestS3` 및 `ForMSKTestVPC`의 두 CloudFormation 스택을 삭제합니다. 스택 삭제가 성공적으로 완료되면 모든 리소스가 삭제됩니다.

## 다음 단계

> **Note**
>
> 이 예제를 따르는 동안 리소스를 만든 경우 예상치 못한 요금이 발생하지 않도록 리소스를 삭제해야 합니다.

통합은 Amazon MSK와 DynamoDB를 연결하여 스트림 데이터가 OLTP 워크로드를 지원할 수 있도록 하는 아키텍처를 식별했습니다. 여기서 DynamoDB를 OpenSearch Service와 연결하여 더 복잡한 검색을 실현할 수 있습니다. 더욱 복잡한 이벤트 중심 요구 사항이 있다면 EventBridge와 통합하고, 더 높은 처리량과 짧은 지연 시간이 필요하다면 Amazon Managed Service for Apache Flink와 같은 확장을 고려하세요.

# DynamoDB와의 통합 모범 사례

DynamoDB를 다른 서비스와 통합할 때는 항상 각 서비스 사용에 대한 모범 사례를 따라야 합니다. 하지만 통합과 관련하여 고려해야 할 몇 가지 모범 사례가 있습니다.

주제

- DynamoDB에서 스냅샷 생성
- DynamoDB에서의 변경 데이터 캡처

## DynamoDB에서 스냅샷 생성

- 일반적으로 초기 복제를 위한 스냅샷을 생성할 때는 Amazon S3로 내보내기를 사용하는 것이 좋습니다. 둘 다 비용 효율적이며 처리량 측면에서 애플리케이션 트래픽과 경쟁하지 않습니다. 새 테이블에 백업 및 복원한 다음 스캔 작업을 수행하는 방법도 고려해 볼 수 있습니다. 이렇게 하면 애플리케이션과의 처리량 경쟁을 피할 수 있지만 일반적으로 내보내기보다 훨씬 덜 비용 효과적입니다.
- 내보내기를 수행할 때는 항상 `StartTime`을 설정하세요. 이렇게 하면 변경 데이터 캡처(CDC)를 어디서 시작할지 쉽게 결정할 수 있습니다.

- S3로 내보내기를 사용하는 경우 S3 버킷에 수명 주기 작업을 설정하세요. 일반적으로 만료 작업을 7일로 설정하는 것이 안전하지만 회사에서 지침을 제시한다면 따라야 합니다. 수집 후 명시적으로 항목을 삭제하더라도 이렇게 하면 문제를 포착하여 불필요한 비용을 줄이고 정책 위반을 예방할 수 있습니다.

## DynamoDB에서의 변경 데이터 캡처

- 실시간에 가까운 CDC가 필요한 경우 DynamoDB Streams 또는 Amazon Kinesis Data Streams(KDS)를 사용하세요. 어느 것을 사용할지 결정할 때는 일반적으로 다운스트림 서비스와 함께 사용하기 가장 쉬운 방법을 고려합니다. 파티션 키 수준에서 순서대로 이벤트를 처리해야 하거나 항목이 매우 큰 경우에는 DynamoDB Streams를 사용하세요.
- 실시간에 가까운 CDC가 필요하지 않은 경우 증분 내보내기와 함께 Amazon S3로 내보내기를 사용하여 두 시점 사이에 발생한 변경 사항만 내보낼 수 있습니다.

  스냅샷을 생성할 때 S3로 내보내기를 사용한 경우 유사한 코드를 사용하여 증분 내보내기를 처리할 수 있으므로 이 방법이 특히 유용할 수 있습니다. 일반적으로 S3로 내보내기가 이전 스트리밍 옵션보다 약간 저렴하지만 보통 비용은 옵션 선택의 주요 요인이 아닙니다.
- 일반적으로 DynamoDB 스트림의 동시 소비자는 두 명만 있을 수 있습니다. 통합 전략을 계획할 때 이 점을 고려하세요.
- 스캔을 사용하여 변경 사항을 감지하지 마세요. 규모가 작을 때는 효과가 있을 수 있지만 곧 실용성이 떨어집니다.

# DynamoDB에서 생성형 AI 사용

Amazon DynamoDB는 모든 규모에서 10밀리초 미만의 성능을 제공하는 서버리스, NoSQL, 완전관리형 데이터베이스입니다. DynamoDB는 처리량이 많은 워크로드에 최적화되어 있으며 생성형 AI 모델과 통합하여 기능을 확장할 수 있습니다. 생성형 AI 모델을 사용하면 DynamoDB 테이블에 저장된 데이터를 실시간으로 작업하고 상황에 맞게 인식되고 고도로 맞춤화된 애플리케이션을 구축할 수 있습니다. 또한 비즈니스, 사용자 및 애플리케이션 데이터를 완전하게 활용하여 생성형 AI 솔루션을 사용자 지정함으로써 최종 사용자 경험을 향상할 수 있습니다.

생성형 AI 및 AWS가 생성형 AI 애플리케이션 구축을 지원하는 솔루션에 대해 자세히 알아보려면 생성형 AI를 사용하여 비즈니스 전환을 참조하시기 바랍니다.

주제

- DynamoDB용 생성형 AI 사용 사례
- DynamoDB용 생성형 AI 블로그
- OpenSearch Service와 DynamoDB의 제로 ETL 통합 활용

## DynamoDB용 생성형 AI 사용 사례

DynamoDB는 파운데이션 모델(FM)로 구축된 챗봇 및 고객 상담 센터와 같은 AI 기반 대화형 애플리케이션에 널리 사용됩니다. Amazon Bedrock, Amazon SageMaker AI 또는 기타 모델 공급업체를 통해 FM에 액세스할 수 있습니다. 이러한 애플리케이션은 일반적으로 DynamoDB를 사용하여 3가지 데이터 패턴(애플리케이션 데이터, 비즈니스 데이터 및 사용자 데이터) 전반에서 맞춤화를 개선하고 사용자 경험을 향상시킵니다. 이러한 데이터 패턴의 몇 가지 예는 다음과 같습니다.

- LangChain, LlamaIndex 또는 사용자 지정 코드와의 통합을 통해 채팅 메시지 기록 등의 애플리케이션 데이터를 저장합니다. 이 컨텍스트를 통해 모델이 사용자와 전후로 대화할 수 있도록 하여 사용자 경험을 향상시킵니다.
- 인벤토리, 요금 및 설명서와 같은 비즈니스 데이터를 활용하여 사용자 지정 사용자 환경을 조성합니다.
- 웹 기록, 과거 주문 및 사용자 기본 설정 등의 사용자 데이터를 적용하여 맞춤화된 답변을 제공합니다.

예를 들어 보험 회사는 DynamoDB로 챗봇을 빌드하여 거의 실시간 데이터에 대한 검색 증강 생성(RAG) 기반 생성형 AI 모델에 대한 액세스를 제공할 수 있습니다. 이러한 데이터의 예로는 실시간

모기지 요금, 제품 요금, 규정 준수/표준 계약 사본, 사용자 웹 기록 및 사용자 기본 설정이 있습니다. DynamoDB를 RAG와 결합하면 보험 상품 및 사용자 데이터에 관한 심층적이고 업데이트된 정보가 추가됩니다. 이렇게 하면 프롬프트와 답변이 강화되어 최종 사용자에게 정확하고 맞춤화된 경험을 거의 실시간으로 제공할 수 있습니다.

마찬가지로 금융 서비스 산업 고객은 DynamoDB, Amazon Bedrock 지식 기반 및 Amazon Bedrock 에이전트를 사용하여 RAG 기반 생성형 AI 애플리케이션을 구축합니다. 이러한 애플리케이션에서는 오픈 소스 수익 보고서 및 통화 스크립트를 사용할 수 있습니다. 또한 사용자별 포트폴리오와 트랜잭션 기록을 사용하여 향후 전망을 포함한 온디맨드 포트폴리오 요약을 만들 수 있습니다.

## DynamoDB용 생성형 AI 블로그

다음 문서에서는 DynamoDB의 기능을 활용하여 고급 AI 기반 애플리케이션을 구축하는 데 도움이 되는 자세한 사용 사례, 모범 사례 및 단계별 가이드를 제공합니다.

- 생성형 AI 챗봇용 Amazon DynamoDB 데이터 모델
- Amazon DynamoDB, Amazon Bedrock 및 LangChain을 사용하여 확장 가능한 컨텍스트 인식 챗봇 구축

## OpenSearch Service와 DynamoDB의 제로 ETL 통합 활용

Amazon Bedrock을 DynamoDB와 함께 사용하여 Amazon Titan 및 기타 서드 파티 모델 등 파운데이션 모델(FM)에 대한 서버리스 액세스를 제공할 수 있습니다. 생성형 AI 애플리케이션을 구축할 때 Amazon OpenSearch Service와의 제로 ETL 통합을 활용하여 벡터 검색 기능을 사용하는 것이 가능합니다. DynamoDB 제로 ETL에서 OpenSearch로의 생성형 AI 통합 및 Amazon Bedrock 워크숍에서는 OpenSearch와의 DynamoDB 제로 ETL 통합을 설정하는 핸즈온 경험을 제공합니다. 이 워크숍에서는 다음 작업을 수행합니다.

- DynamoDB 테이블에서 OpenSearch로 파이프라인을 만듭니다.
- OpenSearch에서 Amazon Bedrock 커넥터를 만듭니다.
- OpenSearch를 벡터 스토어로 활용하여 Amazon Bedrock을 쿼리합니다.
- Amazon Bedrock의 Claude FM을 사용하여 OpenSearch에서 반환한 검색 결과를 영어로 설명하는 플레인 응답을 만듭니다.

이 워크숍을 통해 DynamoDB를 OpenSearch와 통합하여 생성형 AI 애플리케이션을 구축할 수 있습니다. 또한 기존 사용 사례에 DynamoDB와 OpenSearch를 통합하는 데 도움이 되는 데이터베이스 엔진

전반에서 유연한 쿼리 기능도 시연합니다. 이 워크숍은 Amazon DynamoDB Immersion Day의 7개 모듈 중 하나입니다. 모든 AWS 계정에서 이 워크숍을 실행할 수 있습니다.

또한 DynamoDB와 OpenSearch Service 간의 제로 ETL 통합을 설정하는 방법에 대한 다음 블로그 게시물을 참조하시기 바랍니다. 이 블로그 게시물에서는 수신 데이터에 Amazon Bedrock을 사용하여 임베딩을 자동으로 생성하도록 OpenSearch Service에서 모델 커넥터를 설정하는 방법도 설명해 줍니다. Amazon DynamoDB의 벡터 검색과 Amazon OpenSearch Service의 제로 ETL 통합

# Amazon DynamoDB에 대한 할당량 및 제약 조건

이 주제에서는 Amazon DynamoDB 내의 현재 할당량(이전에는 제한이라고 함)을 설명합니다. 이 주제에서는 현재 할당량을 보고 할당량 증가를 요청하는 등 할당량 관리 작업을 수행하는 방법도 설명합니다.

주제


- DynamoDB에서 할당량 관리 작업 수행
- DynamoDB에서 할당량 증가 요청
- Amazon DynamoDB의 할당량
- Amazon DynamoDB의 제약 조건


# DynamoDB에서 할당량 관리 작업 수행

Amazon DynamoDB에는 테이블, 스트림, 인덱스 등과 같은 여러 서비스 구성 요소가 있습니다. AWS 계정을 생성할 때 이러한 구성 요소에 기본 할당량(이전에는 제한이라고 함)이 설정됩니다. 다르게 표시되지 않는 한 리전별로 각 할당량이 적용됩니다. 일부 할당량에 대한 증가를 요청할 수 있습니다. 리소스 할당량에 도달하면 해당 리소스 생성을 위한 추가 요청이 예외와 함께 실패합니다.

## DynamoDB 할당량 액세스

다음과 같은 방법으로 DynamoDB Service Quotas에 대한 작업을 수행할 수 있습니다.

- AWS Management Console

  Service Quotas 콘솔은 Service Quotas를 보고 관리하는 데 사용할 수 있는 브라우저 기반 인터페이스입니다. 상단 탐색 모음에서 Service Quotas를 선택하거나 AWS Management Console에서 Service Quotas를 검색하여 모든 AWS Management Console 페이지에서 Service Quotas에 액세스할 수 있습니다.

- AWS Command Line Interface 도구

  AWS Command Line Interface 도구를 사용할 때는 시스템의 명령줄에서 명령을 실행하여 Service Quotas 작업을 수행할 수 있습니다. AWS 작업을 수행하는 스크립트를 작성하는 경우 명령줄 도구가 유용합니다.

- AWS SDK

다양한 프로그래밍 언어 및 플랫폼(예: Java, Python, Ruby, .NET, iOS, Android 등)에 AWS SDK를 사용하여 Service Quotas 작업을 수행할 수 있습니다.

Service Quotas 콘솔에서 조정 가능한 할당량을 사용할 수 없는 경우 AWS Support Center Console을 사용하여 서비스 할당량 증가 사례를 생성합니다.

## 콘솔에서 현재 할당량 보기

Service Quotas 콘솔을 사용하여 현재 DynamoDB 할당량을 보려면

1. https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/에서 Service Quotas 콘솔을 엽니다.
2. 화면 상단의 탐색 모음에서 리전을 선택합니다.
3. 콘솔에는 DynamoDB 할당량 이름, 적용된 계정 수준 할당량 값, AWS 기본 할당량 값, 사용률 및 계정 수준 또는 리소스 수준 할당량의 조정 가능성에 대한 세부 정보가 표시됩니다.

   적용된 할당량 값 또는 사용률을 사용할 수 없는 경우 콘솔에 사용할 수 없음이 표시됩니다. 지원 센터 콘솔을 통해 적용된 할당량 값을 요청할 수 있습니다.
4. 특정 할당량 이름을 선택하여 세부 정보 페이지를 봅니다. 세부 정보 페이지에는 해당 할당량의 설명, 할당량 코드, 할당량 ARN, 사용률, 적용된 계정 수준 할당량 값, 조정 가능성 및 AWS 기본 할당량 값이 표시됩니다.

   해당하는 경우 세부 정보 페이지에는 모니터링  옵션, 경보, 요청 기록 및 할당량의 태그도 표시됩니다.

## AWS CLI를 사용하여 현재 할당량 보기

DynamoDB 할당량의 기본값을 보려면:

- DynamoDB 서비스 코드(dynamodb)를 사용해 `ListDefaultServiceQuotas` 작업을 직접적으로 호출하여 Amazon DynamoDB 서비스 할당량의 기본값을 검색합니다.

```
$ aws service-quotas list-aws-default-service-quotas \
        --service-code dynamodb

 {
    "Quotas": [
```

```
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1::dynamodb/L-F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false
        },
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1::dynamodb/L-AB614373",
            "QuotaCode": "L-AB614373",
            "QuotaName": "Table-level write throughput limit",
            "Value": 40000.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false
        }......
    ]
}
```

적용된 할당량 값을 보려면:

- DynamoDB 서비스 코드(dynamodb)를 사용하고 ACCOUNT, RESOURCE 또는 ALL을 파라미터 QuotaAppliedAtLevel의 값으로 각각 전달해 ListServiceQuotas 작업을 직접적으로 호출하여 계정 수준, 리소스 수준 또는 모든 수준에서 적용된 모든 할당량 값을 검색합니다. 다음 CLI 예제에서는 계정 수준에서 적용된 할당량 값을 검색합니다.

```
$ aws service-quotas list-service-quotas \
        --service-code dynamodb \
        --quota-applied-at-level ACCOUNT


{
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
```

```
            "QuotaArn": "arn:aws:servicequotas:us-east-1:303935678045:dynamodb/L-
F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
{
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1:303935678045:dynamodb/L
-F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false,
            "QuotaAppliedAtLevel": "ACCOUNT"
        }.....

        }
    ]
}
```

# DynamoDB에서 할당량 증가 요청

Service Quotas 콘솔, AWS CLI 또는 지원 사례를 사용하여 IAM 할당량 증가를 요청할 수 있습니다. Service Quotas 콘솔에서 조정 가능한 할당량을 사용할 수 없는 경우 AWS Support Center Console을 사용하여 서비스 할당량 증가 사례를 생성합니다.

지원는 할당량 증가 요청을 승인 또는 거부하거나 부분적으로 승인할 수 있습니다. 증가분은 즉시 부여되지 않으며 적용되는 데 며칠이 걸릴 수 있습니다.

Service Quotas 콘솔을 사용하여 증가 요청

1. Service Quotas 콘솔 열기(https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/)
2. 화면 상단의 탐색 모음에서 리전을 선택합니다.

3. 리소스 이름을 기준으로 목록을 필터링합니다. 예를 들어, 온디맨드 인스턴스의 할당량을 찾으려면 온디맨드를 입력합니다.
4. 할당량을 조정할 수 있는 경우 할당량을 선택한 다음 할당량 증가 요청을 선택합니다.
5. 할당량 값 변경에 새 할당량 값을 입력합니다.
6. 요청을 선택합니다.
7. 콘솔에서 보류 중이거나 최근에 해결된 요청을 보려면 탐색 창에서 대시보드를 선택합니다. 보류 중인 요청의 경우 요청 상태를 선택하여 요청 접수증을 엽니다. 요청의 초기 상태는 Pending(보류 중)입니다. 상태가 할당량 요청됨으로 변경되면 지원의 케이스 번호가 표시됩니다. 이 케이스 번호를 선택하여 요청의 티켓을 엽니다.

AWS CLI 또는 SDK를 사용하여 할당량 증가를 요청하는 방법을 비롯한 자세한 내용은 Service Quotas 사용 설명서의 할당량 증가 요청을 참조하세요.

# Amazon DynamoDB의 할당량

이 섹션에서는 Amazon DynamoDB 내의 현재 할당량(이전에는 제한이라고 함)에 대해 설명합니다. 각 할당량은 다르게 지정되지 않는 한 리전별로 적용됩니다.

주제

- 읽기/쓰기 처리량
- 예약 용량
- 표
- 글로벌 테이블
- 보조 인덱스
- 프로젝션된 보조 인덱스 속성
- DynamoDB Streams
- Amazon S3에서 가져오기
- Amazon S3로 테이블 내보내기
- 백업 및 복원
- Contributor Insights

# 읽기/쓰기 처리량

## 처리량 기본 할당량

AWS는 계정이 한 리전 내에서 프로비저닝하고 소비할 수 있는 처리량에 몇 가지 기본 할당량을 둡니다.

계정 수준 읽기 처리량 및 계정 수준 쓰기 처리량 할당량은 계정 수준에서 적용됩니다. 이러한 계정 수준 할당량은 해당 리전의 모든 계정 테이블 및 글로벌 보조 인덱스에 대해 프로비저닝된 처리량 용량의 합계에 적용됩니다. 모든 계정의 사용 가능 처리량은 한 테이블 또는 여러 테이블에 프로비저닝할 수 있습니다. 이러한 할당량은 프로비저닝된 용량 모드를 사용하는 테이블에만 적용됩니다.

테이블 수준 읽기 처리량 할당량 및 테이블 수준 쓰기 처리량 할당량은 프로비저닝된 용량 모드를 사용하는 테이블과 온디맨드 용량 모드를 사용하는 테이블에 다르게 적용됩니다.

프로비저닝된 용량 모드 테이블 및 GSI의 경우 할당량은 리전의 모든 테이블 또는 해당 GSI에 프로비저닝할 수 있는 최대 읽기 및 쓰기 용량 단위입니다. 개별 테이블과 모든 GSI의 합계도 계정 수준의 읽기 및 쓰기 처리량 할당량 이하로 유지되어야 합니다. 이는 프로비저닝된 모든 테이블과 해당 GSI의 합계가 계정 수준의 읽기 및 쓰기 처리량 할당량 이하로 유지되어야 한다는 요구 사항에 추가됩니다.

온디맨드 용량 모드 테이블 및 GSI의 경우 테이블 수준 할당량은 모든 테이블 또는 해당 테이블 내의 개별 GSI에 사용할 수 있는 최대 읽기 및 쓰기 용량 단위입니다. 온디맨드 모드의 테이블에는 계정 수준의 읽기 및 쓰기 처리량 할당량이 적용되지 않습니다.

다음은 기본적으로 계정에 적용되는 처리량 할당량입니다.

> **Note**
>
> 서비스 할당량 증가를 통해 DynamoDB 테이블에 대한 읽기 용량 단위(RCU) 또는 쓰기 용량 단위(WCU)를 원하는 수만큼 요청할 수 있습니다. 다음 표에 나열된 값은 초기 기본 할당량을 나타냅니다. 이는 테이블의 최대 한도가 아닙니다.

| 처리량 할당량 이름 | 온디맨드 | 프로비저닝됨 | 조정 가능 |
| --- | --- | --- | --- |
| 테이블당 | 40,000 읽기 요청 단위 및 40,000 쓰기 요청 단위 | 40,000 읽기 용량 단위 및 40,000 쓰기 용량 단위 | 예 |

| 처리량 할당량 이름 | 온디맨드 | 프로비저닝됨 | 조정 가능 |
| --- | --- | --- | --- |
| 계정당 | 해당 사항 없음 | 80,000 읽기 용량 단위 및 80,000 쓰기 용량 단위 | 예 |
| 모든 테이블 또는 글로벌 보조 인덱스의 최소 처리량 | 해당 사항 없음 | 1 읽기 용량 단위 및 1 쓰기 용량 단위 | 예 |

## 처리량 늘리기 또는 줄이기(프로비저닝된 테이블의 경우)

### 프로비저닝된 처리량 늘리기

AWS Management Console 또는 `UpdateTable` 작업을 사용하여 필요한 만큼 자주 `ReadCapacityUnits` 또는 `WriteCapacityUnits`를 늘릴 수 있습니다. 단일 호출에서 테이블, 해당 테이블의 모든 글로벌 보조 인덱스 또는 테이블과 인덱스 조합의 할당된 처리량을 늘릴 수 있습니다. 새 설정은 `UpdateTable` 작업이 완료된 후에 적용됩니다.

프로비저닝된 용량을 추가할 때는 계정당 할당량을 초과할 수 없으며, DynamoDB가 프로비저닝된 용량이 너무 빠르게 늘지 않도록 제한합니다. 이러한 제한 사항을 만족하는 경우에는 원하는 만큼 테이블의 할당된 용량을 늘릴 수 있습니다. 계정당 할당량에 대한 자세한 내용은 앞의 처리량 기본 할당량 섹션을 참조하세요.

### 프로비저닝된 처리량 줄이기

`UpdateTable` 작업의 모든 테이블 및 글로벌 보조 인덱스에 대해 `ReadCapacityUnits` 또는 `WriteCapacityUnits`(또는 둘 다)를 줄일 수 있습니다. 새 설정은 `UpdateTable` 작업이 완료된 후에 적용됩니다.

DynamoDB 테이블에서 하루에 수행할 수 있는 프로비저닝된 용량 감소 횟수에는 기본 할당량이 있습니다. 하루는 협정 세계시(UTC)에 따라 정의됩니다. 특정 일에 아직 용량 감소를 수행하지 않은 경우 한 시간 내에 최대 4회 용량 감소를 수행할 수 있습니다. 그 후에는 시간당 1회 추가 감소를 수행할 수 있습니다(60분에 1회). 이렇게 하면 하루 최대 감소 횟수가 27회까지 줄어듭니다.

> **Important**
>
> 테이블 감소 제한과 글로벌 보조 인덱스 감소 제한은 별개이므로 특정 테이블에 대한 모든 글로벌 보조 인덱스는 자체 감소 제한을 갖습니다. 그러나 단일 요청이 테이블 및 글로벌 보조 인

덱스에 대한 처리량을 감소시키는 경우에는 둘 중 하나가 현재 제한을 초과하게 되면 요청이 거부됩니다. 요청은 부분적으로 처리되지 않습니다.

Example

하루의 첫 4시간 동안에는 글로벌 보조 인덱스가 있는 테이블을 다음과 같이 수정할 수 있습니다.

- 테이블의 `WriteCapacityUnits` 또는 `ReadCapacityUnits`(또는 둘 다)를 네 번 감소합니다.
- 글로벌 보조 인덱스의 `WriteCapacityUnits` 또는 `ReadCapacityUnits`(또는 둘 다)를 네 번 감소합니다.

같은 날의 종료 시간에 테이블 및 글로벌 보조 인덱스 처리량은 잠재적으로 각각 총 27회 감소될 수 있습니다.

## 예약 용량

AWS는 계정에서 구매할 수 있는 활성 예약 용량에 기본 할당량을 지정합니다. 할당량 한도는 쓰기 용량 단위(WCU)와 읽기 용량 단위(RCU)의 예약 용량 조합입니다.

| 예약 용량 할당량 | 활성 예약 용량 | 조정 가능 |
| --- | --- | --- |
| 계정당 | 1,000,000개의 프로비저닝 용량 단위(WCUs _ RCUs) | 예 |

한 번의 구매로 1,000,000개 이상의 프로비저닝 용량 단위를 구매하려고 하면 이 서비스 할당량 한도에 대한 오류 메시지가 표시됩니다. 활성 예약 용량이 있는 상태에서 활성 프로비저닝 용량 단위가 1,000,000개를 초과하도록 추가 예약 용량을 구매하려고 할 경우 이 서비스 할당량 한도에 대한 오류 메시지가 표시됩니다.

## 표

### 테이블 크기

테이블 크기는 실제로 제한이 없습니다. 테이블의 항목 수 또는 바이트 수에는 제약이 없습니다.

### 계정당 최대 테이블 수(리전당)

임의의 AWS 계정에 대해 AWS 리전당 2,500개 테이블의 초기 할당량이 있습니다.

단일 계정에 2,500개 이상의 테이블이 필요한 경우 AWS 계정 팀에 문의하여 최대 1만 개의 테이블까지 늘릴 수 있는 방안을 모색하세요. 1만 개 이상의 경우 권장되는 모범 사례는 여러 계정을 설정하는 것입니다. 각 계정은 최대 1만 개의 테이블을 제공할 수 있습니다.

## 글로벌 테이블

글로벌 테이블을 사용할 때 다음과 같은 기본 할당량이 적용됩니다.

| 기본 글로벌 테이블 할당량 | 온디맨드 | 프로비저닝됨 |
| --- | --- | --- |
| MRSC 글로벌 테이블 수(the section called “일관성 모델” 참조) | 모든 용량 모드에서 400개의 총 MRSC 글로벌 테이블 | 모든 용량 모드에서 400개의 총 MRSC 글로벌 테이블 |
| 다중 리전 최종 일관성(MREC)을 위해 구성된 테이블당 처리량 | 40,000 읽기 요청 단위 및 40,000 쓰기 요청 단위 | 40,000 읽기 용량 단위 및 40,000 쓰기 용량 단위 |
| 다중 리전 강력한 일관성(MRSC)을 위해 구성된 테이블당 처리량 | 40,000 읽기 요청 단위 및 40,000 쓰기 요청 단위 | 40,000 읽기 용량 단위 및 40,000 쓰기 용량 단위 |
| 계정별, 리전별, 일별 새 복제본에 대해 백필된 데이터 | 10TB | 10TB |

> **Note**
>
> AWS Support를 통해 할당량 한도 증가를 요청해야 하는 경우가 있을 수 있습니다. 다음 중 하나라도 해당하는 경우 https://aws.amazon.com/support를 참조하세요.
>
> - 복제본 생성에 성공하려면 글로벌 테이블 처리량 할당량이 테이블당 처리량 할당량보다 크거나 같아야 합니다. MREC 및 MRSC 글로벌 테이블에는 별도의 글로벌 테이블 처리량 할당량이 있습니다.
> - 24시간 내에 하나의 대상 리전에 총 합계가 10TB를 초과하는 복제본을 추가하는 경우 추가 복제본 데이터 채우기 할당량에 대한 서비스 할당량 증가를 요청해야 합니다.
> - 다음과 비슷한 오류가 발생하는 경우

- 'example_region_B' 리전의 현재 계정 한도를 초과하므로 'example_region_A' 리전에 'example_table' 테이블의 복제본을 생성할 수 없습니다.

## 보조 인덱스

테이블당 최대 5개의 로컬 보조 인덱스를 정의할 수 있습니다.

기본적으로 테이블당 20개의 글로벌 보조 인덱스 할당량이 있습니다.

## 프로젝션된 보조 인덱스 속성

테이블의 모든 로컬 및 글로벌 보조 인덱스에 속성을 최대 100개까지 프로젝션할 수 있습니다. 사용자 지정 프로젝션 속성에만 이 할당량이 적용됩니다.

`CreateTable` 작업에서 `INCLUDE`의 `ProjectionType`을 지정할 경우 모든 보조 인덱스에서 합계한 `NonKeyAttributes`에 지정되는 속성의 총 개수는 100을 초과하지 않아야 합니다. 동일한 속성을 2개의 다른 인덱스에 프로젝션할 경우 할당량을 확인할 때 2개의 서로 다른 속성으로 계산됩니다.

이 할당량은 `ProjectionType`이 `KEYS_ONLY` 또는 `ALL`인 보조 인덱스에는 적용되지 않습니다.

## DynamoDB Streams

### DynamoDB Streams 내 샤드의 동시 리더

글로벌 테이블이 아닌 단일 리전 테이블의 경우 최대 2개의 동시 프로세스가 한 번에 동일 DynamoDB Streams 샤드에서 읽기 작업을 수행하도록 설계할 수 있습니다. 이 제한을 초과하면 요청 병목이 발생할 수 있습니다. 글로벌 테이블의 경우 요청 제한을 피하기 위해 동시 리더 수를 1로 제한하는 것이 좋습니다.

### DynamoDB Streams가 활성화된 테이블에 대한 최대 쓰기 용량

AWS는 DynamoDB Streams이 활성화된 DynamoDB 테이블에 대한 쓰기 용량에 몇 가지 기본 할당량을 둡니다. 이러한 기본 할당량은 프로비저닝된 읽기/쓰기 용량 모드의 테이블에만 적용됩니다.

- 테이블당 - 쓰기 용량 단위 40,000

## Amazon S3에서 가져오기

Amazon S3에서 DynamoDB 가져오기는 us-east-1, us-west-2 및 eu-west-1 리전에서 한 번에 총 가져오기 소스 객체 크기가 15TB인 최대 50개의 동시 가져오기 작업을 지원할 수 있습니다. 다른 모든 리전에서는 총 크기가 1TB인 최대 50개의 동시 가져오기 작업이 지원됩니다. 각 가져오기 작업은 모든 리전에서 최대 5만 개의 Amazon S3 객체를 가져올 수 있습니다. 가져오기 및 검증에 대한 자세한 내용은 가져오기 형식 할당량 및 검증을 참조하세요.

## Amazon S3로 테이블 내보내기

전체 내보내기: 최대 300개의 동시 내보내기 태스크 또는 모든 진행 중인 테이블 내보내기에서 최대 총 100TB를 내보낼 수 있습니다. 이러한 두 제한 모두 내보내기가 대기열에 추가되기 전에 확인됩니다.

증분 내보내기: Amazon S3로 DynamoDB 증분 내보내기는 최대 300개의 동시 내보내기 작업 또는 모든 진행 중인 테이블 내보내기에서 최대 총 100TB를 지원할 수 있습니다. 내보내기 기간 제한은 최소 15분, 최대 24시간입니다.

## 백업 및 복원

DynamoDB는 DynamoDB 온디맨드 또는 연속 백업을 통해 총 50TB의 동시 복원을 최대 50개까지 지원합니다. AWS Backup은 총 25TB의 동시 복원을 최대 50개까지 지원합니다.

## Contributor Insights

DynamoDB 테이블에서 고객 인사이트를 활성화해도 Contributor Insights 규칙 제한이 계속 적용됩니다. 자세한 내용은 CloudWatch 서비스 할당량을 참조하세요.

# Amazon DynamoDB의 제약 조건

이 섹션에서는 Amazon DynamoDB 내의 현재 제약 조건(이전에는 제한이라고 함)에 대해 설명합니다.

주제

- 읽기/쓰기 용량 모드
- 보조 인덱스
- 파티션 키와 정렬 키
- 이름 지정 규칙
- 데이터 타입

- Items
- 속성
- 표현식 파라미터
- DynamoDB Transactions
- DynamoDB Streams
- DynamoDB Accelerator(DAX)
- API별 제약 조건
- DynamoDB 저장 데이터 암호화

## 읽기/쓰기 용량 모드

테이블은 언제든지 온디맨드 모드에서 프로비저닝된 용량 모드로 전환할 수 있습니다. 용량 모드 간에 여러 번 전환하는 경우 다음 조건이 적용됩니다.

- 온디맨드 모드에서 새로 생성된 테이블은 언제든지 프로비저닝된 용량 모드로 전환할 수 있습니다. 하지만 테이블 생성 타임스탬프 이후 24시간이 지난 뒤에야 온디맨드 모드로 다시 전환할 수 있습니다.
- 온디맨드 모드의 기존 테이블은 언제든지 프로비저닝된 용량 모드로 전환할 수 있습니다. 하지만 온디맨드로의 전환을 나타내는 마지막 타임스탬프가 발생한 지 24시간이 지난 후에야 다시 온디맨드 모드로 전환할 수 있습니다.

읽기 및 쓰기 용량 모드 간 전환에 대한 자세한 내용은 DynamoDB에서 용량 모드 전환 시 고려 사항 섹션을 참조하세요.

### 용량 단위 크기(프로비저닝된 테이블의 경우)

읽기 용량 단위 1 = 초당 강력히 일관된 읽기 1 또는 초당 최종적으로 일관된 읽기 2(최대 4KB 크기 항목의 경우).

쓰기 용량 단위 1 = 초당 쓰기 1(최대 1KB 크기 항목의 경우)

트랜잭션 읽기 요청은 최대 4KB 크기 항목의 초당 1회 읽기를 수행하는 데 2개의 읽기 용량 단위가 필요합니다.

트랜잭션 쓰기 요청은 최대 1KB 크기 항목의 초당 1회 쓰기를 수행하는 데 2개의 쓰기 용량 단위가 필요합니다.

### 요청 단위 크기(온디맨드 테이블의 경우)

읽기 요청 단위 1 = 초당 강력히 일관된 읽기 1 또는 초당 최종적으로 일관된 읽기 2(최대 4KB 크기 항목의 경우).

쓰기 요청 단위 1 = 초당 쓰기 1(최대 1KB 크기 항목의 경우)

트랜잭션 읽기 요청은 최대 4KB 크기 항목의 초당 1회 읽기를 수행하는 데 2개의 읽기 요청 단위가 필요합니다.

트랜잭션 쓰기 요청은 최대 1KB 크기 항목의 초당 1회 쓰기를 수행하는 데 2개의 쓰기 요청 단위가 필요합니다.

## 보조 인덱스

### 테이블당 프로젝션된 보조 인덱스 속성

테이블의 모든 로컬 및 글로벌 보조 인덱스에 속성을 총 100개까지 프로젝션할 수 있습니다. 사용자 지정 프로젝션 속성에만 이 제한이 적용됩니다.

`CreateTable` 작업에서 `INCLUDE`의 `ProjectionType`을 지정할 경우 모든 보조 인덱스에서 합계한 `NonKeyAttributes`에 지정되는 속성의 총 개수는 100을 초과하지 않아야 합니다. 동일한 속성을 2개의 다른 인덱스에 프로젝션할 경우 합계를 확인할 때 2개의 서로 다른 속성으로 계산됩니다.

이 제한은 `ProjectionType`이 `KEYS_ONLY` 또는 `ALL`인 보조 인덱스에 적용되지 않습니다.

## 파티션 키와 정렬 키

### 파티션 키 길이

파티션 키 값의 최소 길이는 1바이트입니다. 최대 길이는 2,048바이트입니다.

### 파티션 키 값

고유 파티션 키 값의 수는 테이블 또는 보조 인덱스에 대해 실제로 제한이 없습니다.

### 정렬 키 길이

정렬 키 값의 최소 길이는 1바이트입니다. 최대 길이는 1,024바이트입니다.

### 정렬 키 값

일반적으로 파티션 키 값당 고유 정렬 키 값의 수는 실제로 제한이 없습니다.

보조 인덱스를 포함하는 테이블은 예외입니다. 항목 컬렉션은 동일한 파티션 키 속성 값을 가진 항목 세트입니다. 글로벌 보조 인덱스에서 항목 컬렉션은 기본 테이블과 독립적이며 다른 파티션 키 속성을 가질 수 있지만, 로컬 보조 인덱스에서는 인덱싱된 뷰가 테이블의 항목과 동일한 파티션에 콜로케이션되며 동일한 파티션 키 속성을 공유합니다. 이 로컬리티 때문에 테이블에 하나 이상의 LSI가 있는 경우 항목 컬렉션을 여러 파티션에 배포할 수 없습니다.

LSI가 하나 이상 있는 테이블의 경우 항목 컬렉션 크기는 10GB를 초과할 수 없습니다. 여기에는 모든 기본 테이블 항목과 파티션 키 속성의 값이 동일한 모든 프로젝션된 LSI 뷰가 포함됩니다. 파티션의 최대 크기는 10GB입니다. 더 자세한 내용은 항목 컬렉션 크기 제한섹션을 참조하세요.

## 이름 지정 규칙

### 테이블 이름 및 보조 인덱스 이름

테이블 및 보조 인덱스의 이름은 3자 이상이어야 하며 255자를 초과할 수 없습니다. 다음은 허용된 문자입니다.

- `A-Z`
- `a-z`
- `0-9`
- `_` (밑줄)
- `-` (하이픈)
- `.`(점)

### 속성 이름

일반적으로 속성 이름은 1자 이상이어야 하며 64KB를 초과할 수 없습니다.

다음과 같은 예외가 있습니다. 이러한 속성 이름은 255자 이내여야 합니다.

- 보조 인덱스 파티션 키 이름
- 보조 인덱스 정렬 키 이름

- 사용자 지정 프로젝션 속성 이름(로컬 보조 인덱스에만 적용 가능). `CreateTable` 작업에서 `INCLUDE`의 `ProjectionType`을 지정할 경우 `NonKeyAttributes` 파라미터에 있는 속성 이름의 길이가 제한됩니다. `KEYS_ONLY` 및 `ALL` 프로젝션 유형에는 적용되지 않습니다.

이러한 속성 이름은 UTF-8로 인코딩해야 하며, 인코딩 후 각 이름의 총 크기가 255바이트를 초과할 수 없습니다.

## 데이터 타입

### String

문자열 길이는 최대 항목 크기 400KB의 제한을 받습니다.

문자열은 UTF-8 이진수 인코딩을 사용하는 유니코드입니다. UTF-8은 가변 폭 인코딩이므로, DynamoDB는 문자열의 길이를 UTF-8 바이트 기준으로 결정합니다.

### 숫자

숫자는 최대 38자리 정밀도의 양수, 음수 또는 0이 될 수 있습니다.

- 양의 범위: `1E-130` ~ `9.99999999999999999999999999999999999999E+125`
- 음의 범위: `-9.99999999999999999999999999999999999999E+125` ~ `-1E-130`

DynamoDB는 요청 및 응답에서 JSON 문자열을 사용하여 숫자 데이터를 표시합니다. 자세한 내용은 DynamoDB 하위 수준 API 섹션을 참조하세요.

숫자 정밀도가 중요한 경우에는 숫자 형식에서 변환된 문자열을 사용하여 숫자를 DynamoDB로 전달해야 합니다.

### 바이너리

이진수 길이는 최대 항목 크기 400KB의 제한을 받습니다.

이진 속성을 처리하는 애플리케이션에서는 데이터를 DynamoDB로 보내기 전에 Base64 형식으로 인코딩해야 합니다. DynamoDB가 데이터를 받아 부호가 없는 바이트 배열로 디코딩하고 속성 길이로 사용합니다.

# Items

## 항목 크기

DynamoDB에서 최대 항목 크기가 400KB를 초과할 수 없습니다. 여기에는 속성 이름 이진 길이(UTF-8 길이)와 속성 값 길이(이진 길이)가 모두 포함됩니다. 속성 이름은 크기 제한에 포함됩니다.

예를 들어 속성이 2개인 항목이 있습니다. 한 속성은 이름이 "shirt-color"이고 값이 "R"이며 다른 속성은 이름이 "shirt-size"이고 값이 "M"입니다. 해당 항목의 총 크기는 23바이트입니다.

## 로컬 보조 인덱스가 있는 테이블의 항목 크기

테이블에 있는 각 로컬 보조 인덱스의 경우 다음의 전체 크기에 400KB의 제한이 적용됩니다.

- 테이블에 있는 항목의 데이터 크기
- 모든 로컬 보조 인덱스에서 해당 항목(키 값 및 프로젝션 속성 포함)의 크기입니다.

# 속성

## 항목별 속성 이름-값 페어

항목당 속성 누적 크기는 최대 DynamoDB 항목 크기(400KB) 이내여야 합니다.

## 목록, 맵 또는 세트의 값 수

목록, 맵 또는 집합 내 값의 수에는 제한이 없습니다. 단, 값을 포함하는 항목이 400KB 항목 크기 제한을 초과하지 않아야 합니다.

## 속성 값

속성이 테이블 또는 인덱스의 키 속성으로 사용되지 않는 경우 빈 문자열 및 이진 속성 값을 사용할 수 있습니다. 빈 문자열 및 이진 값은 집합, 목록 및 맵 형식 내에서 사용할 수 있습니다. 속성 값은 빈 집합(문자열 집합, 숫자 집합 또는 이진 집합)일 수 없습니다. 하지만 빈 목록 및 맵은 허용됩니다.

## 속성 중첩 깊이

DynamoDB는 최대 32개 수준 깊이까지 중첩된 속성을 지원합니다.

# 표현식 파라미터

표현식 파라미터는 ProjectionExpression, ConditionExpression, UpdateExpression 및 FilterExpression을 포함합니다.

## 길이

표현식 문자열의 최대 길이는 4KB입니다. 예를 들어, ConditionExpression a=b의 크기는 3바이트입니다.

표현식 속성 이름 또는 표현식 속성 값 하나의 최대 길이는 255바이트입니다. 예를 들어, #name은 5바이트이고, :val은 4바이트입니다.

한 표현식에서 모든 치환 변수의 최대 길이는 2MB입니다. 이 길이는 모든 ExpressionAttributeNames 및 ExpressionAttributeValues의 길이의 합계입니다.

## 연산자 및 피연산자

UpdateExpression에 허용된 최대 연산자 또는 함수의 수는 300개입니다. 예를 들어 UpdateExpression SET a = :val1 + :val2 + :val3에 2개의 "+" 연산자가 포함됩니다.

IN 비교기의 최대 피연산자 수는 100개입니다.

## 예약어

DynamoDB는 예약어와 충돌하는 이름의 사용을 금지하지 않습니다. (전체 목록은 DynamoDB의 예약어 섹션을 참조하세요.)

하지만 표현식 파라미터에 예약어를 사용하는 경우에는 ExpressionAttributeNames도 지정해야 합니다. 자세한 내용은 DynamoDB의 표현식 속성 이름(별칭) 섹션을 참조하세요.

# DynamoDB Transactions

DynamoDB 트랜잭션 API 작업의 제약 조건은 다음과 같습니다.

- 트랜잭션은 고유 항목을 100개 이상 포함할 수 없습니다.
- 트랜잭션은 4MB를 초과하는 데이터를 포함할 수 없습니다.
- 트랜잭션에서 동일한 테이블의 동일한 항목에 대해 두 작업을 동시에 수행할 수 없습니다. 예를 들어 동일한 트랜잭션에서 동일한 항목에 대해 ConditionCheck 작업과 Update 작업을 동시에 수행할 수 없습니다.

- 하나의 트랜잭션을 여러 AWS 계정 또는 리전의 테이블에서 작동할 수 없습니다.
- 트랜잭션 작업은 원래 쓰기 작업이 실행된 AWS 리전에서만 ACID(원자성, 일관성, 격리 및 내구성) 보장을 제공합니다. 전역 테이블에서는 트랜잭션이 리전 간에 지원되지 않습니다. 예를 들어 미국 동부(오하이오) 및 미국 서부(오레곤) 리전에 복제본이 있는 전역 테이블이 있고 미국 동부(버지니아 북부) 리전에서 `TransactWriteItems` 작업을 수행한다고 가정합니다. 이 경우 변경 사항이 복제될 때 미국 서부(오레곤) 리전에서 부분적으로 완료된 트랜잭션을 관찰할 수 있습니다. 변경 사항은 소스 리전에서 커밋된 이후에만 다른 리전에 복제됩니다.

# DynamoDB Streams

## DynamoDB Streams 내 샤드의 동시 리더

글로벌 테이블이 아닌 단일 리전 테이블의 경우 최대 2개의 프로세스가 동시에 동일 DynamoDB Streams 샤드에서 읽기 작업을 수행하도록 설계할 수 있습니다. 이 제한을 초과하면 요청 병목이 발생할 수 있습니다. 글로벌 테이블의 경우 요청 제한을 피하기 위해 동시 리더 수를 1로 제한하는 것이 좋습니다.

# DynamoDB Accelerator(DAX)

## AWS 리전 가용성

DAX를 사용할 수 있는 AWS 리전 목록은 AWS 일반 참조의 DynamoDB Accelerator(DAX)를 참조하세요.

## 노드

DAX 클러스터는 정확히 1개의 프라이머리 노드와 0~10개 사이의 읽기 전용 복제본 노드로 구성됩니다.

하나의 AWS 리전에서 AWS 계정당 총 노드 수는 50개를 초과할 수 없습니다.

## 파라미터 그룹

리전마다 최대 20개의 DAX 파라미터 그룹을 생성할 수 있습니다.

## 서브넷 그룹

리전마다 최대 50개의 DAX 서브넷 그룹을 생성할 수 있습니다.

하나의 서브넷 그룹 내에서는 최대 20개까지 서브넷을 정의할 수 있습니다.

> **Important**
>
> DAX 클러스터는 최대 500개의 DynamoDB 테이블을 지원합니다. DynamoDB 테이블 500개를 초과하면 클러스터의 가용성과 성능이 저하될 수 있습니다.

## API별 제약 조건

### `CreateTable`/`UpdateTable`/`DeleteTable`/`PutResourcePolicy`/`DeleteResourcePolicy`

일반적으로 CreateTable, UpdateTable, DeleteTable, PutResourcePolicy, DeleteResourcePolicy 요청을 어떤 조합으로든 최대 500개까지 동시에 실행할 수 있습니다. 따라서 `CREATING`, `UPDATING` 또는 `DELETING` 상태의 총 테이블 수가 500을 초과할 수 없습니다.

테이블 그룹 간에 변경 가능한(`CreateTable`, `DeleteTable`, `UpdateTable`, `PutResourcePolicy`, `DeleteResourcePolicy`) 컨트롤 플레인 API 요청을 초당 최대 2,500개까지 제출할 수 있습니다. 하지만 `PutResourcePolicy` 및 `DeleteResourcePolicy` 요청의 개별 한도는 더 낮습니다. 자세한 내용은 `PutResourcePolicy` 및 `DeleteResourcePolicy`에 대한 다음 할당량 세부 정보를 참조하세요.

리소스 기반 정책을 포함하는 `CreateTable` 및 `PutResourcePolicy` 요청은 정책의 KB마다 2개의 추가 요청으로 계산됩니다. 예를 들어 정책 크기가 5KB인 `CreateTable` 또는 `PutResourcePolicy` 요청은 요청 11개로 계산됩니다. `CreateTable` 요청이 1개, 리소스 기반 정책이 10개(2 x 5KB)입니다. 마찬가지로 크기가 20KB인 정책은 41개의 요청으로 계산됩니다. `CreateTable` 요청이 1개, 리소스 기반 정책이 40개(2 x 20KB)입니다.

#### `PutResourcePolicy`

테이블 그룹에서 초당 최대 25개의 `PutResourcePolicy` API 요청을 제출할 수 있습니다. 개별 테이블에 대한 요청이 성공하면 이후 15초 동안 새 `PutResourcePolicy` 요청이 지원되지 않습니다.

리소스 기반 정책 문서에 지원되는 최대 크기는 20KB입니다. DynamoDB는 이 한도를 기준으로 정책의 크기를 계산할 때 공백을 계산합니다.

#### `DeleteResourcePolicy`

테이블 그룹에서 초당 최대 50개의 `DeleteResourcePolicy` API 요청을 제출할 수 있습니다. 개별 테이블에 대한 `PutResourcePolicy` 요청이 성공하면 이후 15초 동안 `DeleteResourcePolicy` 요청이 지원되지 않습니다.

**BatchGetItem**

단일 BatchGetItem 작업은 최대 100개의 항목까지 가져올 수 있습니다. 가져온 모든 항목의 전체 크기는 16MB를 초과할 수 없습니다.

**BatchWriteItem**

단일 BatchWriteItem 작업은 최대 25개의 PutItem 또는 DeleteItem 요청을 포함할 수 있습니다. 쓰여진 모든 항목의 전체 크기는 16MB를 초과할 수 없습니다.

**DescribeStream**

DescribeStream을 초당 최대 10회 직접 호출할 수 있습니다.

**DescribeTableReplicaAutoScaling**

DescribeTableReplicaAutoScaling 방법은 초당 10개의 요청만 지원합니다.

**DescribeLimits**

DescribeLimits는 주기적으로만 호출해야 합니다. 1분에 두 번 이상 호출하는 경우 병목 오류가 발생할 수 있습니다.

**DescribeContributorInsights/ListContributorInsights/UpdateContributorInsights**

DescribeContributorInsights, ListContributorInsights, UpdateContributorInsights는 주기적으로만 호출해야 합니다. 각 API에 대해 DynamoDB 는 초당 5개의 요청만 지원합니다.

**DescribeTable/ListTables/GetResourcePolicy**

어떤 조합으로든 읽기 전용(DescribeTable, ListTables, GetResourcePolicy) 컨트롤 플레인 API 요청을 초당 최대 2,500개까지 제출할 수 있습니다. GetResourcePolicy API의 개별 하한은 초당 요청 100개입니다.

**DescribeTimeToLive**

DescribeTimeToLive 작업은 초당 10개의 읽기 요청 단위로 스로틀링됩니다. 이 제한을 초과하면 DynamoDB가 ThrottlingException 오류를 반환합니다.

**Query**

Query의 결과 집합은 호출당 1MB로 제한됩니다. 쿼리 응답의 LastEvaluatedKey를 사용하여 더 많은 결과를 가져올 수 있습니다.

**Scan**

Scan의 결과 집합은 호출당 1MB로 제한됩니다. 스캔 응답의 LastEvaluatedKey를 사용하여 더 많은 결과를 가져올 수 있습니다.

**UpdateKinesisStreamingDestination**

UpdateKinesisStreamingDestination 작업을 수행할 때 24시간 동안 최대 3회까지 ApproximateCreationDateTimePrecision에 새 값을 설정할 수 있습니다.

**UpdateTableReplicaAutoScaling**

UpdateTableReplicaAutoScaling 메서드는 초당 10개의 요청만 지원합니다.

**UpdateTableTimeToLive**

UpdateTableTimeToLive 메서드는 시간당 지정된 테이블당 Time to Live (TTL) 활성화 또는 비활성화 요청을 하나만 지원합니다. 이 변경을 완전히 처리하는 데 최대 1시간까지 걸릴 수 있습니다. 이 1시간 동안 동일한 테이블에 대한 추가 UpdateTimeToLive 호출을 수행하면 ValidationException이 발생합니다.

## DynamoDB 저장 데이터 암호화

AWS 소유 키, AWS 관리형 키 및 고객 관리형 키 간의 전환은 테이블 생성 시점을 기점으로 테이블당 24시간마다 최대 4회까지 언제든 허용됩니다. 지난 6시간 내 변경 사항이 없는 경우 추가 변경이 허용됩니다. 이에 따라 하루에 변경할 수 있는 최대 횟수를 8회로 설정할 수 있습니다(처음 6시간 동안 4회 변경, 이후 6시간마다 1회 변경(당일 기준)).

위의 할당량을 모두 사용한 경우에도 필요한 횟수만큼 AWS 소유 키를 사용하도록 암호화 키를 전환할 수 있습니다.

이러한 할당량은 용량 증가를 요청하지 않은 경우에 적용됩니다. 서비스 할당량 증가를 요청하려면 https://aws.amazon.com/support를 참조하세요.

# DynamoDB API 참조

Amazon DynamoDB API 참조에는 다음이 지원하는 전체 작업 목록이 포함되어 있습니다.

- DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator(DAX)

# Amazon DynamoDB 문제 해결

다음 주제에서는 Amazon DynamoDB를 사용할 때 발생할 수 있는 오류 및 문제에 대한 문제 해결 조언을 제공합니다. 여기에 나열되지 않은 문제를 발견하는 경우 이 페이지의 피드백 버튼을 사용하여 해당 문제를 보고할 수 있습니다.

문제 해결 조언과 일반적인 지원 질문에 대한 답변은 AWS 지식 센터를 참조하세요.

주제


- Amazon DynamoDB의 내부 서버 오류 문제 해결
- Amazon DynamoDB의 지연 시간 문제 해결
- DynamoDB의 제한 문제


## Amazon DynamoDB의 내부 서버 오류 문제 해결

DynamoDB에서 내부 서버 오류(500 오류)는 서비스가 요청을 처리할 수 없음을 나타냅니다. 이러한 오류는 플릿의 일시적인 네트워크 문제, 인프라 문제, 스토리지 노드 관련 문제 등 다양한 이유로 발생할 수 있습니다.

DynamoDB 테이블의 수명 주기 동안 일부 내부 서버 오류가 발생할 수 있습니다. 이는 서비스의 분산 특성 때문에 발생할 수 있으며 일반적으로 걱정할 필요는 없습니다. DynamoDB는 사용자 개입 없이 서비스와 관련된 모든 일시적인 문제를 실시간으로 자동 복구하고 해결합니다. 하지만 테이블에 대한 요청에서 내부 서버 오류가 지속적으로 많은 것으로 관찰되면(the section called “SystemErrors” 지표에서 확인할 수 있음) 더 자세히 조사해야 합니다.

주제


- 내부 서버 오류 조사
- 내부 서버 오류로 인한 영향 최소화
- 운영 상황 파악 개선


### 내부 서버 오류 조사

DynamoDB 테이블에서 내부 서버 오류가 발생하는 경우 다음 옵션을 고려해 보세요.

1. AWS Health Dashboard 확인.

문제를 식별하기 위한 첫 번째 단계는 AWS Service Health Dashboard와 AWS Account Health Dashboard를 확인하는 것입니다. 이 대시보드는 모든 서비스 전반의 문제, 영향을 받는 테이블, 지속되는 문제 및 문제가 해결된 후의 근본 원인에 대한 중요한 정보를 제공합니다.

이 대시보드의 세부 정보를 검토하면 사용 중인 AWS 서비스의 현재 상태와 계정에 영향을 줄 수 있는 잠재적 문제를 더 잘 이해할 수 있습니다. 이 정보는 문제를 해결하고 운영 중단을 최소화하기 위한 다음 단계를 결정하는 데 도움이 될 수 있습니다.

2. 지원에 문의.

   요청에서 장기간 지속적으로 오류가 발생하는 경우 서비스에 문제가 있는 것일 수 있습니다. 일반적으로 지난 15분 동안 전체 실패율이 1% 이상이면 AWS Support 팀에 문제를 에스컬레이션하는 것이 좋습니다. 자세한 내용은 DynamoDB 서비스 수준 계약을 참조하세요.

   AWS Support 팀에 사례를 개설할 때 문제 해결 프로세스를 신속하게 진행할 수 있도록 다음 세부 정보를 제공하세요.

   - 영향을 받는 DDB, 테이블 또는 보조 인덱스
   - 오류가 관찰된 기간
   - DynamoDB 요청 ID(예: `4KBNVRGD25RG1KE09UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG`, 애플리케이션 로그에서 찾을 수 있음)

   지원 사례에 이러한 세부 정보를 포함하면 AWS 팀에서 문제를 이해하고 더 빠르게 해결하는 데 도움이 됩니다. 요청 ID가 없는 경우에도 사용 가능한 다른 세부 정보를 사용하여 사례를 기록해야 합니다.

## 내부 서버 오류로 인한 영향 최소화

DynamoDB 서비스에서 내부 서버 오류가 발생하는 경우 애플리케이션에 미치는 영향을 최소화하려면 다음 모범 사례를 고려하세요.

- 백오프 및 재시도 사용 – DynamoDB의 기본 SDK 동작은 백오프 및 재시도 전략 측면에서 대부분의 애플리케이션에 적합한 균형을 찾도록 설계되었습니다. 하지만 애플리케이션의 가동 중지 시간 허용 한도 및 성능 요구 사항에 따라 이러한 설정을 조정할 수 있습니다. 이러한 재시도 설정을 미세 조정할 수 있는 방법을 파악하려면 백오프 및 재시도에 대해 자세히 알아보세요.
- 최종적으로 일관된 읽기 사용: 애플리케이션에서 강력히 일관된 읽기를 요구하지 않는 경우 최종적으로 일관된 읽기를 사용하는 것이 좋습니다. 이러한 읽기는 사용 가능한 모든 스토리지 노드에서 처

리되므로 비용이 적게 들고 내부 서버 오류로 인한 일시적인 문제가 발생할 가능성이 적습니다. 자세한 내용은 DynamoDB 읽기 일관성 단원을 참조하십시오.

## 운영 상황 파악 개선

오늘날의 디지털 환경에서는 애플리케이션의 고가용성과 신뢰성을 유지하는 것이 매우 중요합니다. 한 가지 중요한 측면은 DynamoDB 테이블 및 글로벌 보조 인덱스(GSI)의 내부 서버 오류(ISE)를 사전에 모니터링하는 것입니다. CloudWatch 경보를 생성하여 이러한 오류를 모니터링하면 운영 상황을 더 잘 파악하고 최종 사용자에게 영향을 미치기 전에 잠재적 문제에 대해 알림을 받을 수 있습니다. 이 접근 방식은 AWS Well-Architected 프레임워크의 운영 우수성 요소와 연계되어 DynamoDB 워크로드가 성능, 보안 및 신뢰성을 위해 최적화되도록 합니다.

CloudWatch 경보 생성

지표를 수동으로 관찰하는 대신 지속적으로 많은 수의 내부 서버 오류에 대한 알림을 수신하도록 DynamoDB 테이블에 CloudWatch 경보를 설정해야 합니다. 이는 AWS의 모든 워크로드에 대한 Well-Architected 프레임워크의 운영 우수성 요소와 연계됩니다. DynamoDB 테이블을 잘 설계하는 방법에 대한 자세한 내용은 DynamoDB Well-Architected 렌즈를 사용하여 DynamoDB 워크로드 최적화 섹션을 참조하세요.

이러한 경보는 사용자 지정 지표 수학을 사용하여 5분 동안 요청 실패 비율을 계산합니다. 권장되는 모범 사례는 연속된 3개의 데이터 포인트가 1% 임계값을 위반할 때(즉, 15분 내에 전체 요청의 1%가 실패) ALARM 상태에 들어가도록 경보를 구성하는 것입니다.

아래 샘플은 테이블에 CloudWatch 경보를 생성하고 테이블에 GSI를 생성하는 데 도움이 되는 AWS CloudFormation 템플릿입니다.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Sample template for monitoring DynamoDB
Parameters:
 DynamoDBProvisionedTableName:
    Description: Name of DynamoDB Provisioned Table to create
    Type: String
    MinLength: 3
    MaxLength: 255
    ConstraintDescription : https://docs.aws.amazon.com/amazondynamodb/latest/
developerguide/Limits.html#limits-naming-rules
  DynamoDBSNSEmail:
    Description : Email Address subscribed to newly created SNS Topic
    Type: String
    AllowedPattern: "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$"
```

```
    MinLength: 1
    MaxLength: 255

Resources:
  DynamoDBMonitoringSNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      DisplayName: DynamoDB Monitoring SNS Topic
      Subscription:
        - Endpoint: !Ref DynamoDBSNSEmail
          Protocol: email
      TopicName: dynamodb-monitoring

  DynamoDBTableSystemErrorAlarm:
    Type: 'AWS::CloudWatch::Alarm'
    Properties:
      AlarmName: 'DynamoDBTableSystemErrorAlarm'
      AlarmDescription: 'Alarm when system errors exceed 1% of total number of requests
 for 15 minutes'
      AlarmActions:
        - !Ref DynamoDBMonitoringSNSTopic
      Metrics:
        - Id: 'e1'
          Expression: 'm1/(m1+m2+m3)'
          Label: SystemErrorsOverTotalRequests
        - Id: 'm1'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'SystemErrors'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm2'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedReadCapacityUnits'
              Dimensions:
                - Name: 'TableName'
```

```
                Value: !Ref DynamoDBProvisionedTableName
          Period: 300
          Stat: 'SampleCount'
          Unit: 'Count'
        ReturnData: False
      - Id: 'm3'
        MetricStat:
          Metric:
            Namespace: 'AWS/DynamoDB'
            MetricName: 'ConsumedWriteCapacityUnits'
            Dimensions:
              - Name: 'TableName'
                Value: !Ref DynamoDBProvisionedTableName
          Period: 300
          Stat: 'SampleCount'
          Unit: 'Count'
        ReturnData: False
    EvaluationPeriods: 3
    Threshold: 1.0
    ComparisonOperator: 'GreaterThanThreshold'
 DynamoDBGSISystemErrorAlarm:
   Type: 'AWS::CloudWatch::Alarm'
   Properties:
     AlarmName: 'DynamoDBGSISystemErrorAlarm'
     AlarmDescription: 'Alarm when GSI system errors exceed 2% of total number of
requests for 15 minutes'
     AlarmActions:
       - !Ref DynamoDBMonitoringSNSTopic
     Metrics:
       - Id: 'e1'
         Expression: 'm1/(m1+m2+m3)'
         Label: GSISystemErrorsOverTotalRequests
       - Id: 'm1'
         MetricStat:
           Metric:
             Namespace: 'AWS/DynamoDB'
             MetricName: 'SystemErrors'
             Dimensions:
               - Name: 'TableName'
                 Value: !Ref DynamoDBProvisionedTableName
               - Name: 'GlobalSecondaryIndexName'
                 Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
           Period: 300
           Stat: 'SampleCount'
```

```
            Unit: 'Count'
          ReturnData: False
        - Id: 'm2'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedReadCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
                - Name: 'GlobalSecondaryIndexName'
                  Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm3'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedWriteCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
                - Name: 'GlobalSecondaryIndexName'
                  Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
      EvaluationPeriods: 3
      Threshold: 1.0
      ComparisonOperator: 'GreaterThanThreshold'
```

# Amazon DynamoDB의 지연 시간 문제 해결

워크로드의 지연 시간이 길면 CloudWatch `SuccessfulRequestLatency` 지표를 분석하고 백분위수 지표(p50)를 통해 평균 지연 시간 및 중앙값 지연 시간을 확인하여 DynamoDB와 관련이 있는지 알아볼 수 있습니다. 보고된 `SuccessfulRequestLatency`의 일부 변동성은 정상이며, 가끔 발생하는 급증(특히 `Maximum` 통계 및 높은 백분위수)은 걱정할 필요가 없습니다. 그러나 `Average` 통계 또는 p50(중앙값)이 급격히 증가했는데도 계속 유지되는 경우에는 AWS 서비스 상태 대시보드와 개인 상

태 대시보드에서 자세한 내용을 확인해야 합니다. 가능한 원인으로는 테이블의 항목 크기(1KB 항목과 400KB 항목은 지연 시간이 다름) 또는 쿼리 크기(항목 10개 대 100개)가 있습니다.

백분위수 지표(p99, p90 등)는 지연 시간 분포를 더 잘 이해하는 데 도움이 될 수 있습니다. 예시:

- p50(중앙값)은 워크로드의 일반적인 지연 시간을 보여줍니다.
- p90은 요청의 90%가 이 값보다 빠르다는 것을 보여줍니다.
- p99는 요청의 1%에 영향을 미치는 최악의 지연 시간을 식별하는 데 도움이 됩니다.

정상인 p50 값과 높은 p99 값은 요청의 작은 부분에 영향을 미치는 산발적인 문제를 나타낼 수 있지만, 지속적으로 높은 p50 값은 성능 저하를 의미할 수 있습니다.

> **Note**
>
> 사용자 지정 백분위수 값(예: p99.9)을 분석하려면 CloudWatch 지표 통계 필드에 원하는 백분위수(예: p99.9)를 수동으로 입력합니다. 이를 통해 드롭다운에 나열된 기본 백분위수를 초과하는 지연 시간 분포를 평가할 수 있습니다.

지연 시간 지표의 일부 변동, 특히 높은 백분위수는 정상입니다. 이는 DynamoDB 테이블에 저장된 데이터의 고가용성 및 내구성을 유지하는 데 도움이 되는 DynamoDB 기반 백그라운드 작업 또는 일시적인 인프라 문제의 결과로 볼 수 있습니다.

필요한 경우 AWS Support를 통해 지원 사례를 개설하고 런북에 따라 애플리케이션에 사용할 수 있는 대체 옵션(예: 다중 리전 아키텍처를 사용하는 경우 리전 철수)을 계속 평가해 보세요. 지원 사례를 열 때 이러한 ID를 제공하는 느린 요청에 대해 AWS Support에 요청 ID를 로깅해야 합니다.

`SuccessfulRequestLatency` 지표는 DynamoDB 서비스 내부의 지연 시간만 측정하며 클라이언트 측 활동 및 네트워크 이동 시간은 포함되지 않습니다. 클라이언트에서 DynamoDB 서비스로의 호출에 대한 전체 지연 시간에 대해 자세히 알아보려면 AWS SDK에서 지연 시간 지표 로깅을 활성화하면 됩니다.

> **Note**
>
> 대부분의 싱글톤 작업(프라이머리 키의 값을 완전히 지정하여 단일 항목에 적용되는 작업)의 경우 DynamoDB는 한 자릿수 밀리초의 `Average SuccessfulRequestLatency`를 제공합니다. 이 값에는 DynamoDB 엔드포인트에 액세스하는 호출자 코드에 대한 전송 오버헤드가 포함되지 않습니다. 다중 항목 데이터 작업의 경우 지연 시간은 결과 세트의 크기, 반환된 데이터 구조의 복잡성, 적용된 조건 표현식 및 필터 표현식과 같은 요인에 따라 달라집니다. 동일한 파

라미터로 동일한 데이터 세트에 대한 다중 항목 작업을 반복하는 경우 DynamoDB의 `Average SuccessfulRequestLatency`는 일관성이 매우 뛰어납니다.

지연 시간을 줄이려면 다음 중 하나 이상의 전략을 고려하세요.

- 연결 재사용: DynamoDB 요청은 기본적으로 HTTPS를 사용하여 인증된 세션을 통해 이루어집니다. 연결을 시작하려면 여러 번의 왕복이 필요하고 시간도 오래 걸리므로 첫 번째 요청의 지연 시간이 연결을 재사용하는 후속 요청의 지연 시간보다 더 깁니다. 이미 초기화된 연결을 통한 요청은 DynamoDB의 일관되고 짧은 지연 시간을 제공합니다. 새 연결을 설정하는 데 따른 지연 시간 오버헤드를 피하기 위해 다른 요청이 없을 경우 30초마다 `GetItem` 요청을 전송하여 '연결 유지' 메커니즘을 구현하는 것이 좋습니다.
- 최종 읽기 일관성 사용: 애플리케이션에서 강력히 일관된 읽기를 요구하지 않는 경우 기본값인 최종 읽기 일관성을 사용하는 것이 좋습니다. 최종적으로 일관된 읽기는 비용이 더 낮고 여러 가용 영역에서 발생할 수 있으므로 요청자에게 공동 배치된 가용 영역을 선택하여 지연 시간을 줄일 수 있습니다. 자세한 내용은 DynamoDB 읽기 일관성 섹션을 참조하세요.
- 요청 해징 구현: 매우 짧은 p99 지연 시간 요구 사항의 경우 요청 해징 구현을 고려하세요. 요청 헤징을 사용하면 초기 요청이 응답을 충분히 빠르게 수신하지 못하는 경우 동등한 요청을 한 번 더 보내고 두 요청이 경합하도록 합니다. 먼저 응답을 받은 요청이 처리됩니다. 이렇게 하면 일부 추가 요청의 비용으로 최악의 지연 시간이 개선됩니다. 두 번째 요청을 보내기 전에 대기할 시간을 결정할 수 있습니다. 헤징은 읽기의 경우 더 쉽습니다. 쓰기의 경우 타임스탬프 기반 순서를 사용하여 헤징된 요청을 첫 번째 시도 시 발생하는 것으로 처리하여 순서가 잘못된 업데이트를 방지합니다. 이 접근 방식은 Timestamp writes for write hedging in Amazon DynamoDB에서 설명되었습니다.
- 요청 제한 시간 및 재시도 동작 조정: 클라이언트에서 DynamoDB로 가는 경로는 여러 구성 요소를 통과하며, 각 구성 요소는 중복성을 염두에 두고 설계되었습니다. 다음 측면을 고려합니다.
  - 네트워크 복원력
  - TCP 패킷 제한 시간
  - DynamoDB의 분산 아키텍처

  기본 SDK 동작은 대부분의 애플리케이션에 최적화되어 있습니다. 하지만 빠른 실패 전략을 구현하고 제한 시간 설정을 조정할 수 있습니다. 평소보다 훨씬 오래 걸리는 요청은 궁극적으로 성공할 가능성이 낮습니다. 빠른 실패 후 재시도하면 다른 경로를 통해 빠르게 성공할 수 있습니다. 이는 요청 헤징과 유사하지만 첫 번째 요청을 진행하는 대신 종료합니다.

  제한 시간 값을 너무 낮게 설정하면 안 됩니다. 제한 시간이 너무 짧으면 클라이언트로 인한 가용성 문제가 발생할 수 있습니다. 예를 들어 소켓 제한 시간이 50밀리초이면 네트워크 지연 시간이 급증할

때(예: 단일 흐름 트래픽에 대한 Amazon EC2 인스턴스 대역폭 제한에 도달하는 경우) 연결 오류가 발생할 수 있습니다. 제한 시간을 줄이는 것의 이점과 애플리케이션 가용성에 대한 잠재적 위험을 신중하게 비교하고 짧은 제한 시간보다는 헤징을 선택하세요.

이 주제에 대한 유용한 논의는 Tuning AWS Java SDK HTTP request settings for latency-aware Amazon DynamoDB applications를 참조하세요.

- 클라이언트와 DynamoDB 엔드포인트 간 거리 줄이기: 사용자가 전 세계에 분산되어 있는 경우 글로벌 테이블 - 다중 활성, 다중 리전 복제 사용을 고려해 보세요. 글로벌 테이블에서는 테이블을 사용할 수 있는 지정된 AWS 리전으로 테이블을 복제할 수 있습니다. 데이터 사본을 최종 사용자 가까이 두어 읽기 및 쓰기 작업 중 네트워크 지연 시간을 줄일 수 있습니다. DynamoDB 글로벌 테이블을 효과적으로 사용하는 방법에 대한 자세한 내용은 AWS 권장 가이드의 Amazon DynamoDB 글로벌 테이블 사용을 참조하세요.
- 캐싱 사용: 트래픽에 읽기가 많은 경우 다음과 같은 캐싱 서비스 중 하나를 사용해 보세요.
  - DynamoDB Accelerator(DAX): DynamoDB를 위한 완전관리형, 고가용성, 인 메모리 캐시로, 초당 요청 수가 몇 백만 개인 경우에도 몇 밀리초에서 몇 마이크로초까지 최대 10배의 성능 개선을 제공합니다. DAX에 대한 자세한 내용은 DynamoDB Accelerator(DAX)를 통한 인 메모리 가속화 섹션을 참조하세요.
  - Amazon ElastiCache: DynamoDB와 통합하여 캐시 어사이드 패턴을 사용해 읽기 성능을 개선할 수 있는 완전관리형 인 메모리 캐싱 서비스입니다. 자세한 내용은 AWS 권장 가이드의 읽기-스루 캐싱을 사용하여 Amazon DynamoDB와 Amazon ElastiCache 통합을 참조하세요.

# DynamoDB의 제한 문제

조절 기능은 애플리케이션에서 용량 단위를 너무 많이 사용하지 않도록 방지합니다. 이 주제에서는 프로비저닝된 용량 모드와 온디맨드 용량 모드의 일반적인 제한 문제를 해결하는 방법을 설명합니다. 또한, CloudWatch를 사용하여 문제의 원인을 조사하는 방법도 설명합니다.

주제


- 프로비저닝된 모드의 제한 문제 해결
- 온디맨드 모드의 제한 문제 해결
- CloudWatch 지표를 사용하여 제한 문제 조사

# 프로비저닝된 모드의 제한 문제 해결

테이블 또는 인덱스에 대해 프로비저닝된 처리량을 애플리케이션에서 초과할 경우 요청 조정이 적용됩니다. 조절 기능은 애플리케이션에서 용량 단위를 너무 많이 사용하지 않도록 방지합니다. DynamoDB는 읽기 또는 쓰기 작업을 제한할 때 호출자에게 `ProvisionedThroughputExceededException`을 반환합니다. 그러면 애플리케이션에서는 잠깐 동안 기다렸다가 요청을 재시도하는 등 적절한 조치를 취할 수 있게 됩니다.

제한과 관련된 것으로 보이는 문제를 해결하려면 먼저 제한이 DynamoDB과 애플리케이션 중 무엇에서 발생하는지 확인하는 것이 중요합니다.

이 주제에서는 프로비저닝된 용량 모드의 일반적인 제한 문제를 해결하는 방법을 설명합니다. 다음은 몇 가지 일반적인 시나리오와 이를 해결하는 데 도움이 될 수 있는 단계입니다.

DynamoDB 테이블에 프로비저닝된 용량이 충분한 것으로 보이지만 요청이 제한되고 있음

이는 처리량이 분당 평균보다 낮지만 초당 사용 가능한 양을 초과할 때 발생할 수 있습니다. DynamoDB는 분 단위 수준의 지표만 CloudWatch에 보고하며, 이 지표는 1분 동안의 합계로 계산되어 평균화됩니다. 하지만 DynamoDB 자체는 초당 속도 제한을 적용합니다. 따라서 이 1분 중 작은 부분(예: 몇 초 이하)에서 해당 처리량이 너무 많이 발생하는 경우 1분의 나머지 시간에 대한 요청이 제한될 수 있습니다.

예를 들어 테이블에 60WCU를 프로비저닝한 경우 1분 안에 3,600회의 쓰기 작업을 수행할 수 있습니다. 그러나 3,600개의 WCU 요청이 모두 같은 초에 발생하면 1분의 나머지 시간은 제한됩니다.

이 시나리오를 해결하는 한 가지 방법은 API 호출에 지터와 지수 백오프를 추가하는 것입니다. 자세한 내용은 백오프 및 지터에 대한 게시물을 참조하세요.

AutoScaling이 활성화되었지만 테이블에 여전히 제한이 발생함

이는 트래픽이 급증할 때 발생할 수 있습니다. 2개의 데이터 포인트가 1분 범위 이내에 구성된 목표 사용률 값을 위반하면 Auto Scaling이 트리거될 수 있습니다. 따라서 일관된 2분 동안 소비된 용량이 목표 사용률을 초과하므로 Auto Scaling이 발생할 수 있습니다. 하지만 급증 간격이 1분보다 크면 Auto Scaling이 트리거되지 않을 수 있습니다.

마찬가지로 15개의 연속 데이터 포인트가 목표 사용률보다 낮을 때 스케일 다운 이벤트가 트리거될 수 있습니다. 두 경우 모두 Auto Scaling이 트리거된 후 `UpdateTable` API 작업이 간접적으로 호출됩니다. 그런 다음 테이블 또는 인덱스의 프로비저닝된 용량을 업데이트하는 데 몇 분 정도 걸릴 수 있습니다. 이 기간 동안 테이블의 이전 프로비저닝된 용량을 초과하는 모든 요청은 제한됩니다.

요약하면 Auto Scaling이 DynamoDB 테이블을 스케일 업하기 위해서는 목표 사용률 값이 위반된 연속적인 데이터 포인트가 필요합니다. 이러한 이유로 Auto Scaling은 급증한 워크로드를 처리하기 위한 솔루션으로 권장되지 않습니다. 자세한 내용은 Auto Scaling 비용 최적화 설명서를 참조하세요.

핫키로 인해 제한 문제가 발생할 수 있음

DynamoDB에서 카디널리티가 높지 않은 파티션 키는 몇 개의 파티션만을 대상으로 하는 요청이 많을 수 있습니다. 이 결과 생성된 핫 파티션이 파티션 제한인 초당 3,000RCU 또는 1,000WCU를 초과하면 제한이 발생할 수 있습니다. 진단 도구인 CloudWatch Contributor Insights(CCI)는 각 테이블의 항목 액세스 패턴에 대한 CCI 그래프를 제공하여 이를 디버깅하는 데 도움이 될 수 있습니다. DynamoDB 테이블의 가장 자주 액세스하는 키와 기타 트래픽 추세를 지속적으로 모니터링할 수 있습니다. CloudWatch Contributor Insights에 대한 자세한 내용은 CloudWatch Contributor Insights for DynamoDB를 참조하세요. 자세한 내용은 DynamoDB에 워크로드가 배포되도록 파티션 키 설계 및 올바른 DynamoDB 파티션 키 선택을 참조하세요.

테이블로 향하는 트래픽이 테이블 수준의 처리량 할당량을 초과하고 있음

테이블 수준 읽기 처리량 할당량 및 테이블 수준 쓰기 처리량 할당량은 어느 리전에서나 테이블 수준에서 적용됩니다. 이러한 할당량은 프로비저닝된 용량 모드 테이블 및 온디맨드 용량 모드 테이블에 모두 적용됩니다. 기본적으로 테이블에 할당되는 처리량 할당량은 4만 개의 읽기 요청 단위와 4만 개의 쓰기 요청 단위입니다. 테이블에 대한 트래픽이 이 할당량을 초과하면 테이블에 제한이 발생할 수 있습니다. 이를 방지하는 방법에 대한 자세한 내용은 운영 인식을 위한 DynamoDB 모니터링을 참조하세요.

이 문제를 해결하려면 Service Quotas 콘솔을 사용하여 계정의 테이블 수준 읽기 또는 쓰기 처리량 할당량을 늘리세요.

## 온디맨드 모드의 제한 문제 해결

온디맨드 용량 모드를 사용하는 DynamoDB 테이블은 애플리케이션의 트래픽 볼륨에 따라 자동으로 조정됩니다. 하지만 온디맨드 모드를 사용하는 테이블은 여전히 제한받을 수 있습니다. 이 주제에서는 온디맨드 테이블의 일반적인 제한 문제를 해결하는 방법을 설명합니다.

트래픽이 이전 최고치의 2배 이상입니다.

30분 이내에 이전 트래픽 최고치의 2배를 초과하면 제한 문제가 발생할 수 있습니다. 이전 트래픽 최고치를 초과하기 전에 최소 30분에 걸쳐 트래픽 증가를 분산시키는 것이 좋습니다. 테이블로 향하는 트래픽을 모니터링하려면 Amazon CloudWatch의 `ConsumedReadCapacityUnits` 지표를 사용하세요. 자세한 내용은 DynamoDB 지표 및 차원 단원을 참조하십시오.

새 온디맨드 테이블의 경우 최대 4,000회 쓰기 요청 단위 또는 12,000회 읽기 요청 단위를 즉시 처리할 수 있습니다.

온디맨드 용량 모드로 전환한 기존 테이블의 경우 이전 최고치는 다음 값 중 하나입니다.

- 테이블에 대한 이전 프로비저닝 처리량의 절반
- 온디맨드 용량 모드를 사용하여 새로 생성된 테이블의 설정

자세한 내용은 온디맨드 용량 모드의 초기 처리량을 참조하세요.

트래픽이 파티션당 최대값 초과

테이블 또는 GSI의 각 파티션은 최대 3,000회 읽기 요청 단위나 1,000회 쓰기 요청 단위를 처리할 수 있습니다. 파티션에 대한 트래픽이 이 제한을 초과하면 파티션에 스로틀링이 발생할 수 있습니다. 이 문제를 해결하려면 다음 작업을 수행합니다.

1. DynamoDB용 CloudWatch Contributor Insights를 사용하여 테이블에서 가장 자주 액세스되고 제한되는 키를 식별합니다.
2. 핫 파티션 키에 대한 요청이 시간 경과에 따라 분산되도록 테이블에 대한 요청을 무작위로 지정합니다. 자세한 내용은 쓰기 샤딩을 사용해 DynamoDB 테이블에 워크로드를 고르게 배포 섹션을 참조하세요.

핫키로 인해 제한 문제가 발생할 수 있음

DynamoDB에서 카디널리티가 높지 않은 파티션 키는 몇 개의 파티션만을 대상으로 하는 요청이 많을 수 있습니다. 이 결과 생성된 핫 파티션이 파티션 제한인 초당 3,000RCU 또는 1,000WCU를 초과하면 스로틀링이 발생할 수 있습니다.

진단 도구인 CloudWatch Contributor Insights(CCI)는 각 테이블의 항목 액세스 패턴에 대한 CCI 그래프를 제공하여 이를 디버깅하는 데 도움이 될 수 있습니다. DynamoDB 테이블의 가장 자주 액세스하는 키와 기타 트래픽 추세를 지속적으로 모니터링할 수 있습니다. CloudWatch Contributor Insights에 대한 자세한 내용은 CloudWatch Contributor Insights for DynamoDB를 참조하세요. 자세한 내용은 DynamoDB에 워크로드가 배포되도록 파티션 키 설계 및 올바른 DynamoDB 파티션 키 선택을 참조하세요.

트래픽이 테이블당 계정 할당량 초과

온디맨드 테이블의 경우 테이블 수준 읽기 처리량 할당량 및 테이블 수준 쓰기 처리량 할당량은 계정 수준에서 적용됩니다. 기본적으로 테이블 처리량은 최대 4만 개의 읽기 요청 단위와 4만 개의 쓰기 요청 단위입니다. 테이블로 향하는 트래픽이 테이블당 계정 할당량의 처리량을 초과할 경우 테이블에 제한이 발생할 수 있습니다. 이 문제를 해결하려면 Service Quotas 콘솔을 사용하여 계정의 테이블 수준 읽기 처리량 할당량 또는 쓰기 처리량 할당량을 늘리세요.

테이블의 글로벌 보조 인덱스가 제한됨

DynamoDB 테이블에 제한받는 중인 보조 글로벌 인덱스가 있는 경우, 제한으로 인해 기본 테이블에 역압 제한이 발생할 수 있습니다. 자세한 내용은 글로벌 보조 인덱스에 제한이 발생하면 Amazon DynamoDB 테이블에 어떤 영향을 미치나요? 및 DynamoDB에서 글로벌 보조 인덱스 사용 섹션을 참조하세요.

트래픽이 구성된 최대 처리량을 초과합니다.

온디맨드 테이블의 읽기 또는 쓰기 작업이 사전 정의된 처리량 제한을 초과하면 테이블이 일시적으로 제한되고 ThrottlingException 오류 메시지가 표시됩니다.

사용 사례에 따라 다음 작업을 완료합니다.

- 최대 테이블 처리량 설정을 늘리거나 끄려면 UpdateTable API를 사용합니다.
- 기다렸다가 요청을 다시 시도합니다. the section called “오류 재시도 횟수 및 지수 백오프” 섹션을 참조하세요.
- 테이블 또는 글로벌 보조 인덱스에 구성된 최대 처리량을 모니터링하려면 CloudWatch 콘솔에서 the section called “OnDemandMaxReadRequestUnits” 및 the section called “OnDemandMaxWriteRequestUnits” 지표를 사용합니다.

## CloudWatch 지표를 사용하여 제한 문제 조사

아래는 제한 이벤트 중에 모니터링해야 하는 몇 가지 DynamoDB 지표입니다. 이를 사용하면 제한된 요청을 생성하는 작업을 파악하고 근본 문제를 식별하는 데 도움이 됩니다.

- `ThrottledRequests`
  - 하나의 제한된 요청에는 여러 개의 제한된 이벤트가 포함될 수 있으므로 요청보다 이벤트를 조사하는 것이 더 적절할 수 있습니다. 예를 들어 테이블의 항목을 GSI로 업데이트하는 경우 테이블에 대한 쓰기 작업, 각 인덱스에 대한 쓰기 작업 등과 같은 여러 이벤트가 발생합니다. 이러한 이벤트 중 하나 이상이 제한되더라도 `ThrottledRequest`는 하나뿐입니다.
- `ReadThrottleEvents`
  - 테이블 또는 GSI에 프로비저닝된 RCU를 초과하는 요청이 있는지 확인합니다.
- `WriteThrottleEvents`
  - 테이블 또는 GSI에 프로비저닝된 WCU를 초과하는 요청이 있는지 확인합니다.
- `OnlineIndexConsumedWriteCapacity`

  - 테이블에 새 GSI를 추가할 때 소비되는 WCU 수에 주의합니다. GSI의 `ConsumedWriteCapacityUnits`에는 인덱스 생성 중에 소비되는 WCU는 포함되지 않는다는 점에 유의합니다.
  - GSI의 WCU를 너무 낮게 설정하면 채우기 단계에서 수신되는 쓰기 작업이 제한될 수 있습니다.
- `Provisioned Read/Write`
  - 지정된 기간 동안 테이블 또는 지정된 글로벌 보조 인덱스에 대해 소비된 프로비저닝된 읽기 또는 쓰기 용량 단위 수를 확인합니다.
  - `TableName` 차원은 기본적으로 테이블에 대한 `ProvisionedReadCapacityUnits`만 반환한다는 점에 유의하세요. 글로벌 보조 인덱스에 대해 프로비저닝된 읽기 또는 쓰기 용량 단위 수를 보려면 `TableName` 및 `GlobalSecondaryIndexName`을 모두 지정해야 합니다.
- `Consumed Read/Write`
  - 지정된 기간 동안 사용된 읽기 또는 쓰기 용량 단위 수를 볼 수 있습니다.

DynamoDB CloudWatch 지표에 대한 자세한 내용은 DynamoDB 지표 및 차원 섹션을 참조하세요.

# DynamoDB 부록

주제


- DynamoDB와의 SSL/TLS 연결 설정 문제 해결
- DynamoDB에서 사용할 테이블 및 데이터 예제
- DynamoDB에 예시 테이블 생성 및 데이터 업로드
- AWS SDK for Python (Boto)을 사용하는 DynamoDB 예시 애플리케이션: Tic-Tac-Toe
- DynamoDB의 예약어
- AWS SDK for Java 1.x 예제
- AWS SDK for Go 1.x 예제
- AWS SDK for Node.js 2.x 예제


# DynamoDB와의 SSL/TLS 연결 설정 문제 해결

Amazon DynamoDB는 타사 인증 기관 대신 ATS(Amazon Trust Services) 인증 기관에서 서명한 보안 인증서로 엔드포인트를 옮기고 있습니다. 2017년 12월, Amazon Trust Services에서 발급한 보안 인증서로 EU-WEST-3(파리) 리전을 시작했습니다. 2017년 12월 이후 시작된 모든 새 리전에는 Amazon Trust Services에서 발급한 보안 인증서가 포함된 엔드포인트가 있습니다. 이 안내서는 SSL/TLS 연결 문제를 확인하고 해결하는 방법을 보여줍니다.

## 애플리케이션 또는 서비스 테스트

대부분의 AWS SDK 및 Command Line Interface(CLI)는 Amazon Trust Services 인증 기관을 지원합니다. 2013년 10월 29일 이전에 릴리스된 AWS SDK for Python 또는 CLI 버전을 사용하는 경우에는 업그레이드해야 합니다. .NET, Java, PHP, Go, JavaScript 및 C++ SDK와 CLI는 인증서를 번들하지 않으며, 해당 인증서는 기본 운영 체제에서 제공됩니다. Ruby SDK에는 2015년 6월 10일 이후의 필수 CA가 하나 이상 포함되어 있습니다. 이 날짜 이전의 Ruby V2 SDK는 인증서를 번들하지 않습니다. 미지원, 사용자 지정 또는 수정된 AWS SDK 버전을 사용하거나 사용자 지정 트러스트 스토어를 사용하는 경우 Amazon Trust Services 인증 기관에 필요한 지원이 없을 수도 있습니다.

DynamoDB 엔드포인트에 대한 액세스를 확인하려면 EU-WEST-3 리전에서 DynamoDB API 또는 DynamoDB Streams API에 액세스하는 테스트를 개발하고 TLS 핸드셰이크가 성공했는지 확인해야 합니다. 이러한 테스트에서 액세스해야 하는 특정 엔드포인트는 다음과 같습니다.

- DynamoDB: https://dynamodb.eu-west-3.amazonaws.com
- DynamoDB Streams: https://streams.dynamodb.eu-west-3.amazonaws.com

애플리케이션이 Amazon Trust Services 인증 기관을 지원하지 않는 경우 다음과 같은 실패 중 하나가 표시됩니다.

- SSL/TLS 협상 오류
- 소프트웨어에서 SSL/TLS 협상 실패를 나타내는 오류가 수신되기 전까지 오래 지연되었습니다. 지연 시간은 클라이언트의 재시도 전략과 제한 시간 구성에 따라 달라집니다.

## 클라이언트 브라우저 테스트

브라우저가 Amazon DynamoDB에 연결할 수 있는지 확인하려면 https://dynamodb.eu-west-3.amazonaws.com URL을 여세요. 테스트에 성공하면 다음과 같은 메시지가 표시됩니다.

```
healthy: dynamodb.eu-west-3.amazonaws.com
```

테스트가 실패하면 https://untrusted-root.badssl.com/과 유사한 오류가 표시됩니다.

## 소프트웨어 애플리케이션 클라이언트 업데이트

DynamoDB 또는 DynamoDB Streams API 엔드포인트(브라우저를 통해서든 프로그래밍 방식으로든 관계없음)에 액세스하는 애플리케이션은 다음 CA 중 하나를 아직 지원하지 않는 경우 클라이언트 시스템에서 신뢰하는 CA 목록을 업데이트해야 합니다.

- Amazon Root CA 1
- Starfield Services Root Certificate Authority - G2
- Starfield Class 2 Certification Authority

클라이언트가 이미 위의 3개 CA 중 하나라도 신뢰하는 경우 해당 클라이언트는 DynamoDB에서 사용하는 인증서를 신뢰하며 다른 작업이 필요하지 않습니다. 그러나 클라이언트가 위의 CA 중 어느 것도 아직 신뢰하지 않는 경우 DynamoDB 또는 DynamoDB Streams API에 대한 HTTPS 연결이 실패합니다. 자세한 내용은 https://aws.amazon.com/blogs/security/how-to-prepare-for-aws-move-to-its-own-certificate-authority/의 블로그 게시물을 참조하세요.

## 클라이언트 브라우저 업데이트

브라우저를 업데이트하여 브라우저의 인증서 번들을 업데이트할 수 있습니다. 가장 일반적인 브라우저에 대한 지침은 브라우저의 웹 사이트에서 찾을 수 있습니다.

- Chrome: https://support.google.com/chrome/answer/95414?hl=en
- Firefox: https://support.mozilla.org/en-US/kb/update-firefox-latest-version
- Safari: https://support.apple.com/en-us/HT204416
- Internet Explorer: https://support.microsoft.com/en-us/help/17295/windows-internet-explorer-which-version#ie=other

## 수동으로 인증서 번들 업데이트

DynamoDB API 또는 DynamoDB Streams API에 액세스할 수 없는 경우 인증서 번들을 업데이트해야 합니다. 업데이트하려면 필수 CA 중 하나 이상을 가져와야 합니다. https://www.amazontrust.com/repository/에서 해당 CA를 찾을 수 있습니다.

다음 운영 체제 및 프로그래밍 언어는 Amazon Trust Services 인증서를 지원합니다.

- 2005년 1월 이후 업데이트가 설치된 Microsoft Windows 버전, Windows Vista, Windows 7, Windows Server 2008 및 최신 버전
- MacOS X 10.4 릴리스 5용 Java가 설치된 Mac OS X 10.4, Mac OS X 10.5 이상 버전
- Red Hat Enterprise Linux 5(2007년 3월), Linux 6 및 Linux 7과 CentOS 5, CentOS 6 및 CentOS 7
- Ubuntu 8.10
- Debian 5.0
- Amazon Linux(모든 버전)
- Java 1.4.2_12, Java 5 업데이트 2 및 Java 6, Java 7 및 Java 8을 비롯한 모든 최신 버전

계속 연결할 수 없는 경우 추가 지원을 위해 소프트웨어 설명서, OS 공급업체에 문의하거나 AWS Support(https://aws.amazon.com/support)에 문의하세요.

# DynamoDB에서 사용할 테이블 및 데이터 예제

이Amazon DynamoDB 개발자 안내서에서는 샘플 테이블을 사용하여 DynamoDB의 다양한 측면을 설명합니다.

| 테이블 이름 | 프라이머리 키 |
| --- | --- |
| ProductCatalog | 단순 기본 키:<br>• `Id` (번호) |
| 포럼 | 단순 기본 키:<br>• `Name` (문자열) |
| Thread | 복합 기본 키:<br>• `ForumName` (문자열)<br>• `Subject` (문자열) |
| Reply | 복합 기본 키:<br>• `Id` (문자열)<br>• `ReplyDateTime` (문자열) |

Reply 테이블에는 PostedBy-Message-Index라는 글로벌 보조 인덱스가 있습니다. 이 인덱스는 Reply 테이블의 키가 아닌 속성 2개에 대한 쿼리를 지원합니다.

| 인덱스 이름 | 프라이머리 키 |
| --- | --- |
| PostedBy-Message-Index | 복합 기본 키:<br>• `PostedBy` (문자열)<br>• `Message` (문자열) |

이러한 테이블에 대한 자세한 내용은 다음(1단계: DynamoDB에 테이블 생성 및 2단계: DynamoDB 테이블에 데이터 쓰기) 단원을 참조하세요.

## 샘플 데이터 파일

주제

- ProductCatalog 샘플 데이터

- Forum 샘플 데이터
- Thread 샘플 데이터
- Reply 샘플 데이터

다음 단원에서는 ProductCatalog, Forum, Thread 및 Reply 테이블을 로드하는 데 사용되는 샘플 데이터 파일을 예시합니다.

각 데이터 파일은 여러 `PutRequest` 요소를 포함하는데, 각 요소에는 하나의 항목이 들어갑니다. 이러한 `PutRequest` 요소는 AWS CLI(AWS Command Line Interface)를 사용하여 `BatchWriteItem` 작업에 대한 입력으로 사용됩니다.

## ProductCatalog 샘플 데이터

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "101"
                    },
                    "Title": {
                        "S": "Book 101 Title"
                    },
                    "ISBN": {
                        "S": "111-1111111111"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            }
                        ]
                    },
                    "Price": {
                        "N": "2"
                    },
                    "Dimensions": {
                        "S": "8.5 x 11.0 x 0.5"
                    },
                    "PageCount": {
```

```
                        "N": "500"
                    },
                    "InPublication": {
                        "BOOL": true
                    },
                    "ProductCategory": {
                        "S": "Book"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "102"
                    },
                    "Title": {
                        "S": "Book 102 Title"
                    },
                    "ISBN": {
                        "S": "222-2222222222"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            },
                            {
                                "S": "Author2"
                            }
                        ]
                    },
                    "Price": {
                        "N": "20"
                    },
                    "Dimensions": {
                        "S": "8.5 x 11.0 x 0.8"
                    },
                    "PageCount": {
                        "N": "600"
                    },
                    "InPublication": {
                        "BOOL": true
```

```
                    },
                    "ProductCategory": {
                        "S": "Book"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "103"
                    },
                    "Title": {
                        "S": "Book 103 Title"
                    },
                    "ISBN": {
                        "S": "333-3333333333"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            },
                            {
                                "S": "Author2"
                            }
                        ]
                    },
                    "Price": {
                        "N": "2000"
                    },
                    "Dimensions": {
                        "S": "8.5 x 11.0 x 1.5"
                    },
                    "PageCount": {
                        "N": "600"
                    },
                    "InPublication": {
                        "BOOL": false
                    },
                    "ProductCategory": {
                        "S": "Book"
                    }
```

```
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "201"
                    },
                    "Title": {
                        "S": "18-Bike-201"
                    },
                    "Description": {
                        "S": "201 Description"
                    },
                    "BicycleType": {
                        "S": "Road"
                    },
                    "Brand": {
                        "S": "Mountain A"
                    },
                    "Price": {
                        "N": "100"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
```

```
                    "N": "202"
                },
                "Title": {
                    "S": "21-Bike-202"
                },
                "Description": {
                    "S": "202 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Brand-Company A"
                },
                "Price": {
                    "N": "200"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Green"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "203"
                },
                "Title": {
                    "S": "19-Bike-203"
                },
                "Description": {
                    "S": "203 Description"
```

```
                    },
                    "BicycleType": {
                        "S": "Road"
                    },
                    "Brand": {
                        "S": "Brand-Company B"
                    },
                    "Price": {
                        "N": "300"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            },
                            {
                                "S": "Green"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "204"
                    },
                    "Title": {
                        "S": "18-Bike-204"
                    },
                    "Description": {
                        "S": "204 Description"
                    },
                    "BicycleType": {
                        "S": "Mountain"
                    },
```

```
                    "Brand": {
                        "S": "Brand-Company B"
                    },
                    "Price": {
                        "N": "400"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "205"
                    },
                    "Title": {
                        "S": "18-Bike-204"
                    },
                    "Description": {
                        "S": "205 Description"
                    },
                    "BicycleType": {
                        "S": "Hybrid"
                    },
                    "Brand": {
                        "S": "Brand-Company C"
                    },
                    "Price": {
                        "N": "500"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
```

```
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        }
    ]
}
```

## Forum 샘플 데이터

```
{
    "Forum": [
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon DynamoDB"},
                    "Category": {"S":"Amazon Web Services"},
                    "Threads": {"N":"2"},
                    "Messages": {"N":"4"},
                    "Views": {"N":"1000"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon S3"},
                    "Category": {"S":"Amazon Web Services"}
                }
            }
        }
    ]
}
```

## Thread 샘플 데이터

```
{
    "Thread": [
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon DynamoDB"
                    },
                    "Subject": {
                        "S": "DynamoDB Thread 1"
                    },
                    "Message": {
                        "S": "DynamoDB thread 1 message"
                    },
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-22T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "0"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "index"
                            },
                            {
                                "S": "primarykey"
                            },
                            {
                                "S": "table"
                            }
                        ]
```

```
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon DynamoDB"
                    },
                    "Subject": {
                        "S": "DynamoDB Thread 2"
                    },
                    "Message": {
                        "S": "DynamoDB thread 2 message"
                    },
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-15T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "3"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "items"
                            },
                            {
                                "S": "attributes"
                            },
                            {
                                "S": "throughput"
                            }
                        ]
                    }
```

```
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon S3"
                    },
                    "Subject": {
                        "S": "S3 Thread 1"
                    },
                    "Message": {
                        "S": "S3 thread 1 message"
                    },
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-29T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "0"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "largeobjects"
                            },
                            {
                                "S": "multipart upload"
                            }
                        ]
                    }
                }
            }
        }
    ]
```

```
}
```

## Reply 샘플 데이터

```
{
    "Reply": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 1"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-15T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 1 Reply 1 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 1"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-22T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 1 Reply 2 text"
                    },
                    "PostedBy": {
                        "S": "User B"
                    }
                }
            }
        },
        {
```

```
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 2"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-29T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 2 Reply 1 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 2"
                    },
                    "ReplyDateTime": {
                        "S": "2015-10-05T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 2 Reply 2 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        }
    ]
}
```

# DynamoDB에 예시 테이블 생성 및 데이터 업로드

이 부록에서는 테이블 생성부터 프로그래밍 방식의 데이터 추가까지 필요한 코드에 대해 살펴봅니다.

주제

- AWS SDK for Java를 사용한 예시 테이블 생성 및 데이터 업로드
- AWS SDK for .NET를 사용한 예시 테이블 생성 및 데이터 업로드

## AWS SDK for Java를 사용한 예시 테이블 생성 및 데이터 업로드

다음은 테이블을 생성하여 데이터를 테이블에 업로드하는 Java 코드 예제입니다. Eclipse의 코드 실행에 대한 단계별 지침은 Java 코드 예 단원을 참조하세요.

```
package com.amazonaws.codesamples;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashSet;
import java.util.TimeZone;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.LocalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class CreateTablesLoadData {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    static String productCatalogTableName = "ProductCatalog";
    static String forumTableName = "Forum";
```

```
    static String threadTableName = "Thread";
    static String replyTableName = "Reply";

    public static void main(String[] args) throws Exception {

        try {

            deleteTable(productCatalogTableName);
            deleteTable(forumTableName);
            deleteTable(threadTableName);
            deleteTable(replyTableName);

            // Parameter1: table name
            // Parameter2: reads per second
            // Parameter3: writes per second
            // Parameter4/5: partition key and data type
            // Parameter6/7: sort key and data type (if applicable)

            createTable(productCatalogTableName, 10L, 5L, "Id", "N");
            createTable(forumTableName, 10L, 5L, "Name", "S");
            createTable(threadTableName, 10L, 5L, "ForumName", "S", "Subject", "S");
            createTable(replyTableName, 10L, 5L, "Id", "S", "ReplyDateTime", "S");

            loadSampleProducts(productCatalogTableName);
            loadSampleForums(forumTableName);
            loadSampleThreads(threadTableName);
            loadSampleReplies(replyTableName);

        } catch (Exception e) {
            System.err.println("Program failed:");
            System.err.println(e.getMessage());
        }
        System.out.println("Success.");
    }

    private static void deleteTable(String tableName) {
        Table table = dynamoDB.getTable(tableName);
        try {
            System.out.println("Issuing DeleteTable request for " + tableName);
            table.delete();
            System.out.println("Waiting for " + tableName + " to be deleted...this may
take a while...");
            table.waitForDelete();
```

```
        } catch (Exception e) {
            System.err.println("DeleteTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
            String partitionKeyName, String partitionKeyType) {

        createTable(tableName, readCapacityUnits, writeCapacityUnits, partitionKeyName,
partitionKeyType, null, null);
    }

    private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
            String partitionKeyName, String partitionKeyType, String sortKeyName,
String sortKeyType) {

        try {

            ArrayList<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
            keySchema.add(new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH)); //
Partition

                            // key

            ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
            attributeDefinitions
                    .add(new AttributeDefinition().withAttributeName(partitionKeyName)
                            .withAttributeType(partitionKeyType));

            if (sortKeyName != null) {
                keySchema.add(new
KeySchemaElement().withAttributeName(sortKeyName).withKeyType(KeyType.RANGE)); // Sort

                            // key
                attributeDefinitions
                        .add(new
AttributeDefinition().withAttributeName(sortKeyName).withAttributeType(sortKeyType));
            }
```

```
            CreateTableRequest request = new
CreateTableRequest().withTableName(tableName).withKeySchema(keySchema)
                    .withProvisionedThroughput(new
ProvisionedThroughput().withReadCapacityUnits(readCapacityUnits)
                            .withWriteCapacityUnits(writeCapacityUnits));

            // If this is the Reply table, define a local secondary index
            if (replyTableName.equals(tableName)) {

                attributeDefinitions
                        .add(new
AttributeDefinition().withAttributeName("PostedBy").withAttributeType("S"));

                ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();
                localSecondaryIndexes.add(new
LocalSecondaryIndex().withIndexName("PostedBy-Index")
                        .withKeySchema(
                                new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH), //
Partition

                               // key
                                new
KeySchemaElement().withAttributeName("PostedBy").withKeyType(KeyType.RANGE)) // Sort

                          // key
                        .withProjection(new
Projection().withProjectionType(ProjectionType.KEYS_ONLY)));

                request.setLocalSecondaryIndexes(localSecondaryIndexes);
            }

            request.setAttributeDefinitions(attributeDefinitions);

            System.out.println("Issuing CreateTable request for " + tableName);
            Table table = dynamoDB.createTable(request);
            System.out.println("Waiting for " + tableName + " to be created...this may
take a while...");
            table.waitForActive();

        } catch (Exception e) {
            System.err.println("CreateTable request failed for " + tableName);
            System.err.println(e.getMessage());
```

```
        }
    }

    private static void loadSampleProducts(String tableName) {

        Table table = dynamoDB.getTable(tableName);

        try {

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("Id", 101).withString("Title", "Book
101 Title")
                    .withString("ISBN", "111-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author1"))).withNumber("Price", 2)
                    .withString("Dimensions", "8.5 x 11.0 x
0.5").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 102).withString("Title", "Book 102
Title")
                    .withString("ISBN", "222-2222222222")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author1", "Author2")))
                    .withNumber("Price", 20).withString("Dimensions", "8.5 x 11.0 x
0.8").withNumber("PageCount", 600)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 103).withString("Title", "Book 103
Title")
                    .withString("ISBN", "333-3333333333")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author1", "Author2")))
                    // Intentional. Later we'll run Scan to find price error. Find
                    // items > 1000 in price.
                    .withNumber("Price", 2000).withString("Dimensions", "8.5 x 11.0 x
1.5").withNumber("PageCount", 600)
                    .withBoolean("InPublication", false).withString("ProductCategory",
"Book");
```

```
            table.putItem(item);

            // Add bikes.

            item = new Item().withPrimaryKey("Id", 201).withString("Title", "18-
Bike-201")
                    // Size, followed by some title.
                    .withString("Description", "201
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Mountain A")
                    // Trek, Specialized.
                    .withNumber("Price", 100).withStringSet("Color", new
 HashSet<String>(Arrays.asList("Red", "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 202).withString("Title", "21-
Bike-202")
                    .withString("Description", "202
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company A").withNumber("Price", 200)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Green",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 203).withString("Title", "19-
Bike-203")
                    .withString("Description", "203
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 300)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Green", "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 204).withString("Title", "18-
Bike-204")
                    .withString("Description", "204
 Description").withString("BicycleType", "Mountain")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 400)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);
```

```
            item = new Item().withPrimaryKey("Id", 205).withString("Title", "20-
Bike-205")
                    .withString("Description", "205
 Description").withString("BicycleType", "Hybrid")
                    .withString("Brand", "Brand-Company C").withNumber("Price", 500)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }

    private static void loadSampleForums(String tableName) {

        Table table = dynamoDB.getTable(tableName);

        try {

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("Name", "Amazon DynamoDB")
                    .withString("Category", "Amazon Web
 Services").withNumber("Threads", 2).withNumber("Messages", 4)
                    .withNumber("Views", 1000);
            table.putItem(item);

            item = new Item().withPrimaryKey("Name", "Amazon
 S3").withString("Category", "Amazon Web Services")
                    .withNumber("Threads", 0);
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void loadSampleThreads(String tableName) {
```

```
        try {
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000); // 7
            // days
            // ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000); // 14
            // days
            // ago
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000); // 21
            // days
            // ago

            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("ForumName", "Amazon DynamoDB")
                    .withString("Subject", "DynamoDB Thread 1").withString("Message",
"DynamoDB thread 1 message")
                    .withString("LastPostedBy", "User
A").withString("LastPostedDateTime", dateFormatter.format(date2))
                    .withNumber("Views", 0).withNumber("Replies",
0).withNumber("Answered", 0)
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
"primarykey", "table")));
            table.putItem(item);

            item = new Item().withPrimaryKey("ForumName", "Amazon
DynamoDB").withString("Subject", "DynamoDB Thread 2")
                    .withString("Message", "DynamoDB thread 2
message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
dateFormatter.format(date3)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
```

```
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
"partitionkey", "sortkey")));
            table.putItem(item);

            item = new Item().withPrimaryKey("ForumName", "Amazon
S3").withString("Subject", "S3 Thread 1")
                    .withString("Message", "S3 Thread 3
message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
dateFormatter.format(date1)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new
HashSet<String>(Arrays.asList("largeobjects", "multipart upload")));
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }

    private static void loadSampleReplies(String tableName) {
        try {
            // 1 day ago
            long time0 = (new Date()).getTime() - (1 * 24 * 60 * 60 * 1000);
            // 7 days ago
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000);
            // 14 days ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000);
            // 21 days ago
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000);

            Date date0 = new Date();
            date0.setTime(time0);

            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);
```

```
            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);

            // Add threads.

            Item item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB
 Thread 1")
                    .withString("ReplyDateTime", (dateFormatter.format(date3)))
                    .withString("Message", "DynamoDB Thread 1 Reply 1
 text").withString("PostedBy", "User A");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 1")
                    .withString("ReplyDateTime", dateFormatter.format(date2))
                    .withString("Message", "DynamoDB Thread 1 Reply 2
 text").withString("PostedBy", "User B");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date1))
                    .withString("Message", "DynamoDB Thread 2 Reply 1
 text").withString("PostedBy", "User A");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date0))
                    .withString("Message", "DynamoDB Thread 2 Reply 2
 text").withString("PostedBy", "User A");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());

        }
    }

}
```

# AWS SDK for .NET를 사용한 예시 테이블 생성 및 데이터 업로드

다음은 테이블을 생성하여 데이터를 테이블에 업로드하는 C# 코드 예제입니다. Visual Studio의 코드 실행에 대한 단계별 지침은 .NET 코드 예시 단원을 참조하세요.

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class CreateTablesLoadData
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                //DeleteAllTables(client);
                DeleteTable("ProductCatalog");
                DeleteTable("Forum");
                DeleteTable("Thread");
                DeleteTable("Reply");

                // Create tables (using the AWS SDK for .NET low-level API).
                CreateTableProductCatalog();
                CreateTableForum();
                CreateTableThread(); // ForumTitle, Subject */
                CreateTableReply();

                // Load data (using the .NET SDK document API)
                LoadSampleProducts();
                LoadSampleForums();
                LoadSampleThreads();
                LoadSampleReplies();
                Console.WriteLine("Sample complete!");
                Console.WriteLine("Press ENTER to continue");
                Console.ReadLine();
```

```
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void DeleteTable(string tableName)
        {
            try
            {
                var deleteTableResponse = client.DeleteTable(new DeleteTableRequest()
                {
                    TableName = tableName
                });
                WaitTillTableDeleted(client, tableName, deleteTableResponse);
            }
            catch (ResourceNotFoundException)
            {
                // There is no such table.
            }
        }

        private static void CreateTableProductCatalog()
        {
            string tableName = "ProductCatalog";

            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                            {
                                new AttributeDefinition
                                {
                                    AttributeName = "Id",
                                    AttributeType = "N"
                                }
                            },
                KeySchema = new List<KeySchemaElement>()
                            {
                                new KeySchemaElement
                                {
                                    AttributeName = "Id",
                                    KeyType = "HASH"
                                }
                            },
```

```
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 10,
                WriteCapacityUnits = 5
            }
        });

        WaitTillTableCreated(client, tableName, response);
    }

    private static void CreateTableForum()
    {
        string tableName = "Forum";

        var response = client.CreateTable(new CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                        {
                            new AttributeDefinition
                            {
                                AttributeName = "Name",
                                AttributeType = "S"
                            }
                        },
            KeySchema = new List<KeySchemaElement>()
                        {
                            new KeySchemaElement
                            {
                                AttributeName = "Name", // forum Title
                                KeyType = "HASH"
                            }
                        },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 10,
                WriteCapacityUnits = 5
            }
        });

        WaitTillTableCreated(client, tableName, response);
    }

    private static void CreateTableThread()
```

```
        {
            string tableName = "Thread";

            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                            {
                                new AttributeDefinition
                                {
                                    AttributeName = "ForumName", // Hash attribute
                                    AttributeType = "S"
                                },
                                new AttributeDefinition
                                {
                                    AttributeName = "Subject",
                                    AttributeType = "S"
                                }
                            },
                KeySchema = new List<KeySchemaElement>()
                            {
                                new KeySchemaElement
                                {
                                    AttributeName = "ForumName", // Hash attribute
                                    KeyType = "HASH"
                                },
                                new KeySchemaElement
                                {
                                    AttributeName = "Subject", // Range attribute
                                    KeyType = "RANGE"
                                }
                            },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });

            WaitTillTableCreated(client, tableName, response);
        }

        private static void CreateTableReply()
        {
```

```
            string tableName = "Reply";
            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                            {
                                new AttributeDefinition
                                {
                                    AttributeName = "Id",
                                    AttributeType = "S"
                                },
                                new AttributeDefinition
                                {
                                    AttributeName = "ReplyDateTime",
                                    AttributeType = "S"
                                },
                                new AttributeDefinition
                                {
                                    AttributeName = "PostedBy",
                                    AttributeType = "S"
                                }
                            },
                KeySchema = new List<KeySchemaElement>()
                            {
                                new KeySchemaElement()
                                {
                                    AttributeName = "Id",
                                    KeyType = "HASH"
                                },
                                new KeySchemaElement()
                                {
                                    AttributeName = "ReplyDateTime",
                                    KeyType = "RANGE"
                                }
                            },
                LocalSecondaryIndexes = new List<LocalSecondaryIndex>()
                            {
                                new LocalSecondaryIndex()
                                {
                                    IndexName = "PostedBy_index",


                                    KeySchema = new List<KeySchemaElement>() {
                                        new KeySchemaElement() {
```

```
                                        AttributeName = "Id", KeyType = "HASH"
                                    },
                                    new KeySchemaElement() {
                                        AttributeName = "PostedBy", KeyType =
"RANGE"
                                    }
                                },
                                Projection = new Projection() {
                                    ProjectionType = ProjectionType.KEYS_ONLY
                                }
                            }
                        },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });

            WaitTillTableCreated(client, tableName, response);
        }

        private static void WaitTillTableCreated(AmazonDynamoDBClient client, string
tableName,
                                 CreateTableResponse response)
        {
            var tableDescription = response.TableDescription;

            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);

            // Let us wait until table is created. Call DescribeTable.
            while (status != "ACTIVE")
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
res.Table.TableName,
```

```
                            res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                // Try-catch to handle potential eventual-consistency issue.
                catch (ResourceNotFoundException)
                { }
            }
        }

        private static void WaitTillTableDeleted(AmazonDynamoDBClient client, string
tableName,
                                DeleteTableResponse response)
        {
            var tableDescription = response.TableDescription;

            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);

            // Let us wait until table is created. Call DescribeTable
            try
            {
                while (status == "DELETING")
                {
                    System.Threading.Thread.Sleep(5000); // wait 5 seconds

                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
res.Table.TableName,
                                res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
            }
            catch (ResourceNotFoundException)
            {
                // Table deleted.
            }
        }

        private static void LoadSampleProducts()
        {
```

```
            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
            // ********** Add Books ********************
            var book1 = new Document();
            book1["Id"] = 101;
            book1["Title"] = "Book 101 Title";
            book1["ISBN"] = "111-1111111111";
            book1["Authors"] = new List<string> { "Author 1" };
            book1["Price"] = -2; // *** Intentional value. Later used to illustrate
scan.
            book1["Dimensions"] = "8.5 x 11.0 x 0.5";
            book1["PageCount"] = 500;
            book1["InPublication"] = true;
            book1["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book1);

            var book2 = new Document();

            book2["Id"] = 102;
            book2["Title"] = "Book 102 Title";
            book2["ISBN"] = "222-2222222222";
            book2["Authors"] = new List<string> { "Author 1", "Author 2" }; ;
            book2["Price"] = 20;
            book2["Dimensions"] = "8.5 x 11.0 x 0.8";
            book2["PageCount"] = 600;
            book2["InPublication"] = true;
            book2["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book2);

            var book3 = new Document();
            book3["Id"] = 103;
            book3["Title"] = "Book 103 Title";
            book3["ISBN"] = "333-3333333333";
            book3["Authors"] = new List<string> { "Author 1", "Author2", "Author
3" }; ;
            book3["Price"] = 2000;
            book3["Dimensions"] = "8.5 x 11.0 x 1.5";
            book3["PageCount"] = 700;
            book3["InPublication"] = false;
            book3["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book3);

            // ************ Add bikes. ******************
            var bicycle1 = new Document();
            bicycle1["Id"] = 201;
```

```
            bicycle1["Title"] = "18-Bike 201"; // size, followed by some title.
            bicycle1["Description"] = "201 description";
            bicycle1["BicycleType"] = "Road";
            bicycle1["Brand"] = "Brand-Company A"; // Trek, Specialized.
            bicycle1["Price"] = 100;
            bicycle1["Color"] = new List<string> { "Red", "Black" };
            bicycle1["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle1);

            var bicycle2 = new Document();
            bicycle2["Id"] = 202;
            bicycle2["Title"] = "21-Bike 202Brand-Company A";
            bicycle2["Description"] = "202 description";
            bicycle2["BicycleType"] = "Road";
            bicycle2["Brand"] = "";
            bicycle2["Price"] = 200;
            bicycle2["Color"] = new List<string> { "Green", "Black" };
            bicycle2["ProductCategory"] = "Bicycle";
            productCatalogTable.PutItem(bicycle2);

            var bicycle3 = new Document();
            bicycle3["Id"] = 203;
            bicycle3["Title"] = "19-Bike 203";
            bicycle3["Description"] = "203 description";
            bicycle3["BicycleType"] = "Road";
            bicycle3["Brand"] = "Brand-Company B";
            bicycle3["Price"] = 300;
            bicycle3["Color"] = new List<string> { "Red", "Green", "Black" };
            bicycle3["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle3);

            var bicycle4 = new Document();
            bicycle4["Id"] = 204;
            bicycle4["Title"] = "18-Bike 204";
            bicycle4["Description"] = "204 description";
            bicycle4["BicycleType"] = "Mountain";
            bicycle4["Brand"] = "Brand-Company B";
            bicycle4["Price"] = 400;
            bicycle4["Color"] = new List<string> { "Red" };
            bicycle4["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle4);

            var bicycle5 = new Document();
            bicycle5["Id"] = 205;
```

```
            bicycle5["Title"] = "20-Title 205";
            bicycle4["Description"] = "205 description";
            bicycle5["BicycleType"] = "Hybrid";
            bicycle5["Brand"] = "Brand-Company C";
            bicycle5["Price"] = 500;
            bicycle5["Color"] = new List<string> { "Red", "Black" };
            bicycle5["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle5);
        }

        private static void LoadSampleForums()
        {
            Table forumTable = Table.LoadTable(client, "Forum");

            var forum1 = new Document();
            forum1["Name"] = "Amazon DynamoDB"; // PK
            forum1["Category"] = "Amazon Web Services";
            forum1["Threads"] = 2;
            forum1["Messages"] = 4;
            forum1["Views"] = 1000;

            forumTable.PutItem(forum1);

            var forum2 = new Document();
            forum2["Name"] = "Amazon S3"; // PK
            forum2["Category"] = "Amazon Web Services";
            forum2["Threads"] = 1;

            forumTable.PutItem(forum2);
        }

        private static void LoadSampleThreads()
        {
            Table threadTable = Table.LoadTable(client, "Thread");

            // Thread 1.
            var thread1 = new Document();
            thread1["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread1["Subject"] = "DynamoDB Thread 1"; // Range attribute.
            thread1["Message"] = "DynamoDB thread 1 message text";
            thread1["LastPostedBy"] = "User A";
            thread1["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
 0, 0, 0));
            thread1["Views"] = 0;
```

```
            thread1["Replies"] = 0;
            thread1["Answered"] = false;
            thread1["Tags"] = new List<string> { "index", "primarykey", "table" };

            threadTable.PutItem(thread1);

            // Thread 2.
            var thread2 = new Document();
            thread2["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread2["Subject"] = "DynamoDB Thread 2"; // Range attribute.
            thread2["Message"] = "DynamoDB thread 2 message text";
            thread2["LastPostedBy"] = "User A";
            thread2["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
0, 0, 0));
            thread2["Views"] = 0;
            thread2["Replies"] = 0;
            thread2["Answered"] = false;
            thread2["Tags"] = new List<string> { "index", "primarykey", "rangekey" };

            threadTable.PutItem(thread2);

            // Thread 3.
            var thread3 = new Document();
            thread3["ForumName"] = "Amazon S3"; // Hash attribute.
            thread3["Subject"] = "S3 Thread 1"; // Range attribute.
            thread3["Message"] = "S3 thread 3 message text";
            thread3["LastPostedBy"] = "User A";
            thread3["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7, 0,
0, 0));
            thread3["Views"] = 0;
            thread3["Replies"] = 0;
            thread3["Answered"] = false;
            thread3["Tags"] = new List<string> { "largeobjects", "multipart upload" };
            threadTable.PutItem(thread3);
        }

        private static void LoadSampleReplies()
        {
            Table replyTable = Table.LoadTable(client, "Reply");

            // Reply 1 - thread 1.
            var thread1Reply1 = new Document();
            thread1Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
```

```
            thread1Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
0, 0, 0)); // Range attribute.
            thread1Reply1["Message"] = "DynamoDB Thread 1 Reply 1 text";
            thread1Reply1["PostedBy"] = "User A";

            replyTable.PutItem(thread1Reply1);

            // Reply 2 - thread 1.
            var thread1reply2 = new Document();
            thread1reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
0, 0, 0)); // Range attribute.
            thread1reply2["Message"] = "DynamoDB Thread 1 Reply 2 text";
            thread1reply2["PostedBy"] = "User B";

            replyTable.PutItem(thread1reply2);

            // Reply 3 - thread 1.
            var thread1Reply3 = new Document();
            thread1Reply3["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1Reply3["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
0, 0, 0)); // Range attribute.
            thread1Reply3["Message"] = "DynamoDB Thread 1 Reply 3 text";
            thread1Reply3["PostedBy"] = "User B";

            replyTable.PutItem(thread1Reply3);

            // Reply 1 - thread 2.
            var thread2Reply1 = new Document();
            thread2Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
attribute.
            thread2Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
0, 0, 0)); // Range attribute.
            thread2Reply1["Message"] = "DynamoDB Thread 2 Reply 1 text";
            thread2Reply1["PostedBy"] = "User A";


            replyTable.PutItem(thread2Reply1);

            // Reply 2 - thread 2.
            var thread2Reply2 = new Document();
```

```
            thread2Reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
 attribute.
            thread2Reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(1,
 0, 0, 0)); // Range attribute.
            thread2Reply2["Message"] = "DynamoDB Thread 2 Reply 2 text";
            thread2Reply2["PostedBy"] = "User A";

            replyTable.PutItem(thread2Reply2);
        }
    }
}
```

# AWS SDK for Python (Boto)을 사용하는 DynamoDB 예시 애플리케이션: Tic-Tac-Toe

Tic-Tac-Toe 게임은 Amazon DynamoDB에 구축된 예제 웹 애플리케이션입니다. 이 애플리케이션은 AWS SDK for Python (Boto)을 사용하여 필요한 DynamoDB를 호출하고 DynamoDB 테이블에 게임 데이터를 저장하며, Python 웹 프레임워크인 Flask를 사용하여 데이터 모델링 방법을 비롯한 DynamoDB의 전반적인 애플리케이션 개발에 대해 설명합니다. 또한 게임 애플리케이션에 대한 테이블 생성, 기본 키 정의, 쿼리 요구 사항을 기준으로 필요한 추가 인덱스, 연결된 값 속성 사용을 포함하여 DynamoDB에서 데이터를 모델링하는 작업과 관련된 모범 사례를 설명합니다.

웹에서 Tic-Tac-Toe 애플리케이션을 플레이하는 방법은 다음과 같습니다.

1. 애플리케이션 홈 페이지에 로그인합니다.
2. 그런 다음 다른 사용자를 게임 플레이 상대방으로 초대합니다.

   다른 사용자가 초대를 수락할 때까지 게임 상태는 `PENDING`으로 유지됩니다. 상대방이 초대를 수락하면 게임 상태가 `IN_PROGRESS`로 변경됩니다.
3. 상대방이 로그인하고 초대를 수락하면 게임이 시작됩니다.
4. 이 애플리케이션은 게임의 모든 동작과 상태 정보를 DynamoDB 테이블에 저장합니다.
5. 게임은 승 또는 무승부로 끝나고 게임 상태는 `FINISHED`로 설정됩니다.

아래에서는 전반적 애플리케이션 구축 연습을 단계별로 설명합니다.

- 1단계: 로컬 배포 및 테스트 - 이 단원에서는 로컬 컴퓨터에 애플리케이션을 다운로드, 배포 및 테스트합니다. 다운로드 버전 DynamoDB에서 필요한 테이블을 생성합니다.

- 2단계: 데이터 모델 및 구현 세부 정보 검사 - 이 단원에서는 우선 연결된 값 속성의 사용, 인덱스 등을 포함하여 데이터 모델에 대해 자세히 설명합니다. 그런 다음 애플리케이션의 작동 방식에 대해 설명합니다.
- 3단계: DynamoDB 서비스를 사용한 프로덕션 내 배포 - 이 단원에서는 프로덕션 시 배포 고려 사항을 중심으로 설명합니다. 이 단계에서는 Amazon DynamoDB 서비스를 사용하여 테이블을 만들고 AWS Elastic Beanstalk을 사용하여 애플리케이션을 배포합니다. 애플리케이션이 프로덕션 단계인 경우 애플리케이션이 DynamoDB 테이블에 액세스할 수 있도록 적절한 권한도 부여합니다. 이 단원에서는 전반적 프로덕션 배포에 대한 단계별 지침을 소개합니다.
- 4단계: 리소스 정리 - 이 단원에서는 이 예제에서 다루지 않는 영역을 강조합니다. 또한 요금이 발생하지 않도록 앞선 단계에서 만든 AWS 리소스를 제거하는 단계도 제공합니다.

## 1단계: 로컬 배포 및 테스트

주제


- 1.1: 필수 패키지 다운로드 및 설치
- 1.2: 게임 애플리케이션 테스트


이 단계에서는 로컬 컴퓨터에 Tic-Tac-Toe 게임 애플리케이션을 다운로드, 배포, 테스트합니다. Amazon DynamoDB 웹 서비스를 사용하는 대신 DynamoDB를 컴퓨터로 다운로드하여 로컬에서 필요한 테이블을 생성합니다.

### 1.1: 필수 패키지 다운로드 및 설치

이 애플리케이션을 로컬로 테스트하려면 다음이 필요합니다.

- Python
- Flask(Python용 마이크로 프레임워크)
- AWS SDK for Python (Boto)
- 컴퓨터에서 실행되는 DynamoDB
- Git

이러한 툴을 다운로드하려면 다음과 같이 하세요.

1. Python을 설치합니다. 단계별 지침은 Python 다운로드 단원을 참조하세요.

   Tic-Tac-Toe 애플리케이션은 Python 버전 2.7을 사용하여 테스트되었습니다.

2. PIP(Python Package Installer)를 사용하여 Flask 및 AWS SDK for Python (Boto)을 설치합니다.
   - PIP를 설치합니다.

     자세한 내용은 PIP 설치 단원을 참조하세요. 설치 페이지에서 [get-pip.py] 링크를 선택하고 파일을 저장합니다. 그런 다음 관리자 권한으로 명령 터미널을 열고 명령 프롬프트에 다음을 입력합니다.

     ```
     python.exe get-pip.py
     ```

     Linux의 경우 `.exe` 확장자는 지정하지 마세요. `python get-pip.py`만 지정합니다.
   - PIP를 사용하여 다음 코드를 사용하는 Flask 및 Boto 패키지를 설치합니다.

     ```
     pip install Flask
     pip install boto
     pip install configparser
     ```
3. DynamoDB를 컴퓨터로 다운로드합니다. 실행 방법에 대한 지침은 DynamoDB local 설정(다운로드 가능 버전) 단원을 참조하세요.
4. Tic-Tac-Toe 애플리케이션을 다운로드합니다.
   a. Git을 설치합니다. 자세한 내용은 git 다운로드를 참조하세요.
   b. 다음 코드를 실행하여 애플리케이션을 다운로드합니다.

   ```
   git clone https://github.com/awslabs/dynamodb-tictactoe-example-app.git
   ```

## 1.2: 게임 애플리케이션 테스트

Tic-Tac-Toe 애플리케이션을 테스트하려면 컴퓨터에서 DynamoDB를 로컬로 실행해야 합니다.

Tic-Tac-Toe 애플리케이션을 실행하는 방법

1. DynamoDB를 시작합니다.
2. Tic-Tac-Toe 애플리케이션의 웹 서버를 시작합니다.

   그러려면 명령 터미널을 열고 Tic-Tac-Toe 애플리케이션을 다운로드한 폴더를 찾아 다음 코드를 사용하여 애플리케이션을 로컬로 실행합니다.

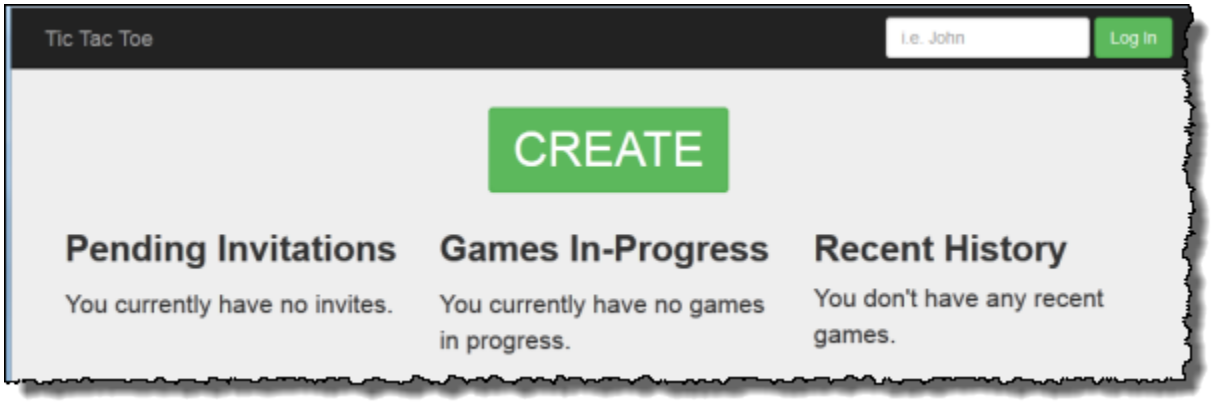
   ```
   python.exe application.py --mode local --serverPort 5000 --port 8000
   ```

Linux의 경우 `.exe` 확장자는 지정하지 마세요.

3. 웹 브라우저를 열고 다음을 입력합니다.

```
http://localhost:5000/
```

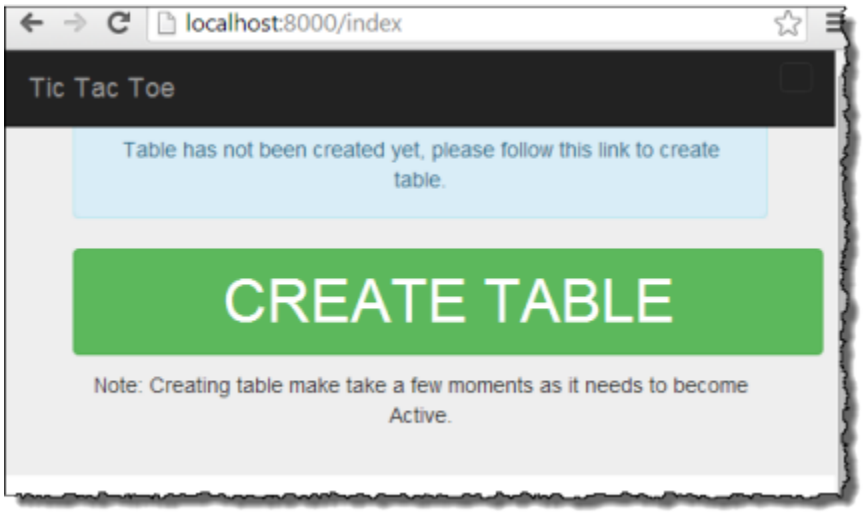
브라우저에 홈 페이지가 표시됩니다.




4. Log in(로그인) 상자에 **`user1`**을 입력하여 user1로 로그인합니다.

> **Note**
>
> 이 예제 애플리케이션에서는 다른 사용자 인증을 수행하지 않습니다. 사용자 ID만 사용하여 플레이어를 식별합니다. 두 명의 플레이어가 동일한 별칭으로 로그인할 경우 애플리케이션은 사용자가 두 가지 브라우저에서 플레이하는 것처럼 작동합니다.

5. 게임을 처음 플레이하는 경우, DynamoDB에서 필수 테이블(`Games`)을 만들라고 요청하는 페이지가 나타납니다. [CREATE TABLE]을 선택합니다.




6. [CREATE]를 선택하여 첫 번째 Tic-Tac-Toe 게임을 만듭니다.
7. Choose an Opponent(상대방 선택) 상자에 **`user2`**를 입력하고 Create Game!(게임 생성!)을 선택합니다.




그러면 `Games` 테이블에 항목이 추가되고 게임이 생성됩니다. 게임 상태는 `PENDING`이 됩니다.

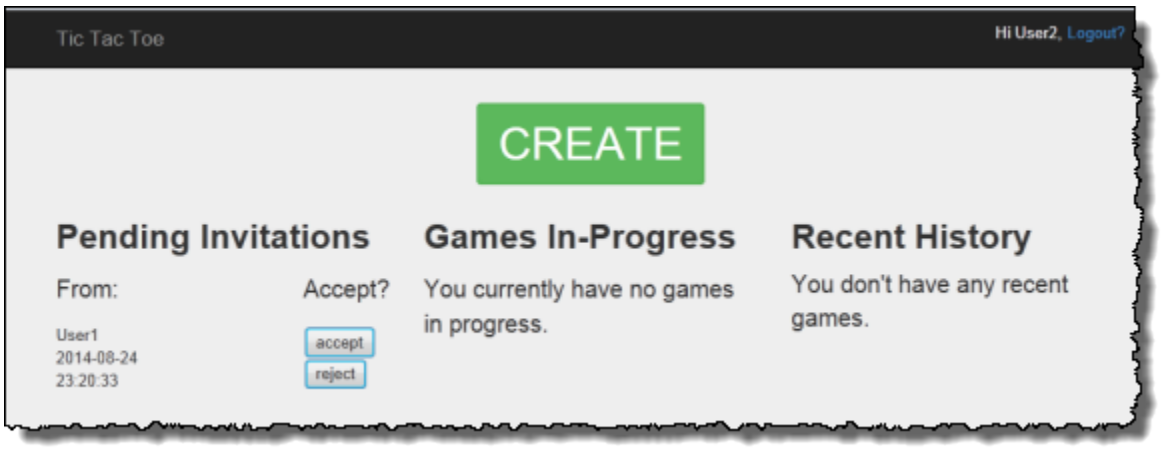
8. 다른 브라우저 창을 열고 다음을 입력합니다.

```
http://localhost:5000/
```

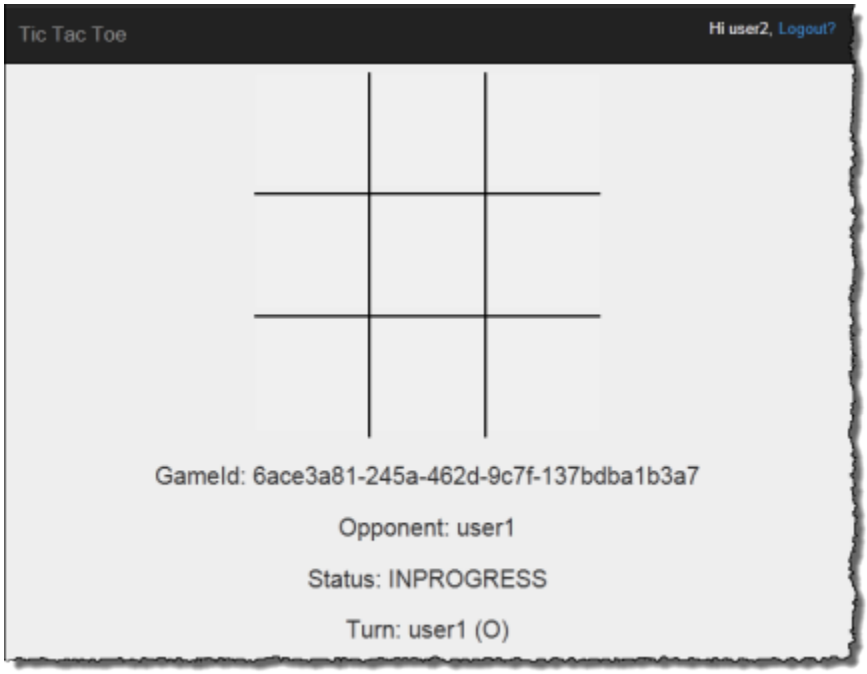
브라우저는 쿠키를 통해 정보를 전달합니다. 따라서 쿠키가 전달되지 않도록 익명 모드 또는 비공개 브라우징을 사용해야 합니다.

9. user2로 로그인합니다.

user1의 초대가 보류 중임을 나타내는 페이지가 표시됩니다.




10. [accept]를 선택하여 초대를 수락합니다.




    빈 그리드가 표시된 게임 페이지가 나타납니다. 이 페이지에는 게임 ID, 차례가 된 사용자, 게임 상태와 같은 관련 정보도 표시됩니다.

11. 게임을 플레이합니다.

웹 서비스는 각 사용자 동작에 대해 Games 테이블의 게임 항목을 조건부 업데이트하라는 요청을 DynamoDB에 전송합니다. 예를 들어 조건은 동작이 유효한지, 사용자가 선택한 사각형이 사용 가능한지, 움직인 사용자의 차례가 맞는지 확인합니다. 유효한 동작일 경우 보드상의 선택에 해당하는 새 속성을 업데이트 작업에서 추가합니다. 또한 업데이트 작업은 다음 동작을 할 수 있는 사용자에게 기존 특성의 값을 설정합니다.

게임 페이지에서 애플리케이션은 DynamoDB의 게임 상태가 변경되었는지 확인하기 위해 1초에 한 번씩 최대 5분간 비동기 JavaScript 호출을 수행합니다. 변경된 경우 애플리케이션은 새 정보로 페이지를 업데이트합니다. 5분 후에는 애플리케이션이 요청을 중지하며 업데이트된 정보를 불러오려면 페이지를 새로 고쳐야 합니다.

# 2단계: 데이터 모델 및 구현 세부 정보 검사

주제


- 2.1: 기본 데이터 모델
- 2.2: 애플리케이션 실행(코드 단계별 안내)


## 2.1: 기본 데이터 모델

이 예제 애플리케이션에서는 다음 DynamoDB 데이터 모델 개념을 중심으로 설명합니다.

- 테이블 - DynamoDB에서 테이블은 항목 모음(즉, 레코드)이고 각 항목은 속성이라고 하는 이름-값 쌍의 모음입니다.

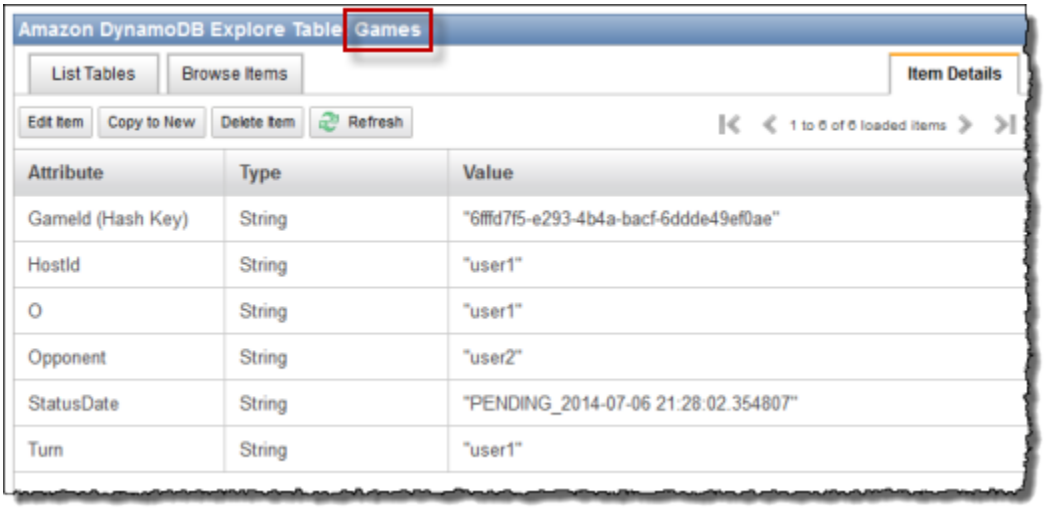
  이 Tic-Tac-Toe 예제에서 애플리케이션은 모든 게임 데이터를 `Games`라고 하는 테이블에 저장합니다. 이 애플리케이션은 테이블에 게임당 한 항목씩 생성하고 모든 게임 데이터를 속성으로 저장합니다. Tic-Tac-Toe 게임은 최대 9회까지 동작이 가능합니다. 기본 키만 필수 속성인 경우 DynamoDB 테이블은 스키마가 없으므로 애플리케이션에서 게임 항목당 다양한 개수의 속성을 저장할 수 있습니다.

  `Games` 테이블에는 문자열 형식의 `GameId` 속성 한 가지로 구성된 단순 기본 키가 있습니다. 애플리케이션이 각 게임에 고유한 ID를 할당합니다. DynamoDB 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요.

  사용자가 다른 사용자를 초대하여 Tic-Tac-Toe 게임을 시작할 경우 애플리케이션은 다음과 같은 게임 메타데이터를 저장하는 속성을 사용하여 `Games` 테이블에 새 항목을 만듭니다.

  - `HostId`는 게임을 시작한 사용자입니다.
  - `Opponent`는 초대를 받은 사용자입니다.
  - 플레이 차례가 된 사용자로, 게임을 시작한 사용자가 먼저 플레이합니다.
  - 보드에서 O 기호를 사용하는 사용자로, 게임을 시작한 사용자가 O 기호를 사용합니다.

  또한 이 애플리케이션은 `StatusDate` 연결된 속성을 생성하여 초기 게임 상태를 `PENDING`으로 표시합니다. 다음 스크린샷은 DynamoDB 콘솔에 나타난 예제 항목을 보여 줍니다.



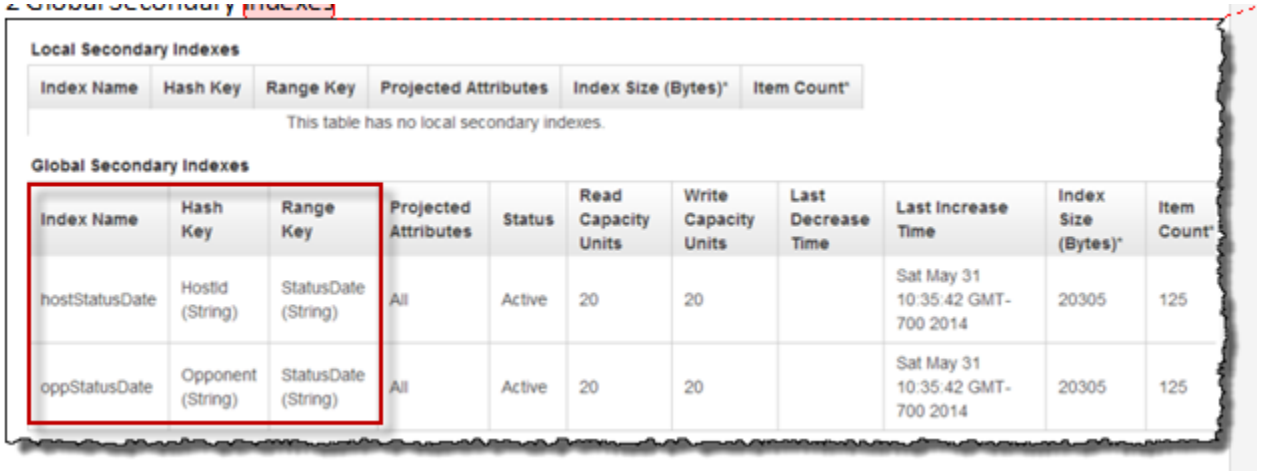

게임이 진행되면 애플리케이션이 각 게임 동작에 대한 하나의 속성을 테이블에 추가합니다. 속성 이름은 보드상의 위치(예: `TopLeft` 또는 `BottomRight`)입니다. 예를 들어 동작은 `O` 값의 `TopLeft` 속성, `O` 값의 `TopRight` 속성, `X` 값의 `BottomRight` 속성을 가질 수 있습니다. 사용자에 따라 속성 값이 `O` 또는 `X`가 됩니다. 예를 들어 다음 보드를 가정해 보세요.




- 연결된 값 속성 - `StatusDate` 속성은 연결된 값 속성을 나타냅니다. 이 방식에서는 게임 상태(`PENDING`, `IN_PROGRESS`, `FINISHED`) 및 날짜(동작을 한 시간)를 저장하기 위한 별도의 속성을 만드는 대신 둘을 하나의 속성으로 결합합니다(예: `IN_PROGRESS_2014-04-30 10:20:32`).

  그런 다음 애플리케이션은 `StatusDate`를 인덱스의 정렬 키로 지정함으로써 `StatusDate` 속성을 사용하여 보조 인덱스를 생성합니다. `StatusDate` 연결된 값 속성을 사용하는 이점에 대해서는 다음 단원의 인덱스 설명에서 자세히 다룹니다.

- 글로벌 보조 인덱스 - 테이블의 기본 키인 GameId를 사용하여 테이블을 효율적으로 쿼리하고 게임 항목을 찾을 수 있습니다. 기본 키 속성이 아닌 속성으로 테이블을 쿼리하기 위해 DynamoDB는 보조 인덱스 생성을 지원합니다. 이 예제 애플리케이션에서는 다음 두 가지 인덱스를 만듭니다.



Local Secondary Indexes

| Index Name | Hash Key | Range Key | Projected Attributes | Index Size (Bytes)* | Item Count* |
|---|---|---|---|---|---|
| This table has no local secondary indexes. | | | | | |

Global Secondary Indexes

| Index Name | Hash Key | Range Key | Projected Attributes | Status | Read Capacity Units | Write Capacity Units | Last Decrease Time | Last Increase Time | Index Size (Bytes)* | Item Count* |
|---|---|---|---|---|---|---|---|---|---|---|
| hostStatusDate | HostId (String) | StatusDate (String) | All | Active | 20 | 20 | | Sat May 31 10:35:42 GMT-700 2014 | 20305 | 125 |
| oppStatusDate | Opponent (String) | StatusDate (String) | All | Active | 20 | 20 | | Sat May 31 10:35:42 GMT-700 2014 | 20305 | 125 |

  - [HostId-StatusDate-index]. 이 인덱스는 파티션 키가 HostId이고 정렬 키가 StatusDate입니다. 이 인덱스를 사용하면, 예를 들어 HostId를 쿼리하여 특정 사용자가 호스팅한 게임을 찾을 수 있습니다.
  - [OpponentId-StatusDate-index]. 이 인덱스는 파티션 키가 OpponentId이고 정렬 키가 StatusDate입니다. 이 인덱스를 사용하면, 예를 들어 Opponent를 쿼리하여 특정 사용자가 상대방인 게임을 찾을 수 있습니다.

  이러한 인덱스는 이러한 인덱스의 파티션 키가 테이블의 기본 키에 사용된 파티션 키 속성(GameId)과 동일하지 않기 때문에 글로벌 보조 인덱스라고 불립니다.

  두 인덱스 모두 정렬 키로 StatusDate를 지정합니다. 이렇게 하면 다음과 같은 작업을 할 수 있습니다.

  - BEGINS_WITH 비교 연산자를 사용하여 쿼리할 수 있습니다. 예를 들어 특정 사용자가 호스팅한 게임 중 IN_PROGRESS 속성을 가진 모든 게임을 검색할 수 있습니다. 이 경우 BEGINS_WITH 연산자가 IN_PROGRESS로 시작하는 StatusDate 값을 확인합니다.
  - DynamoDB는 인덱스 항목을 정렬 키 값을 기준으로 정렬된 순서대로 저장합니다. 따라서 모든 상태 접두사가 동일할 경우(예: IN_PROGRESS) 날짜 부분에 ISO 형식을 사용해 가장 오래된 항목에서 최신 항목의 순서로 항목을 정렬합니다. 이 방식에서는 예를 들어 다음과 같은 특정 쿼리를 효율적으로 수행할 수 있습니다.
    - 로그인한 사용자가 호스팅한 최근 IN_PROGRESS 게임을 최대 10개까지 검색합니다. 이 쿼리를 수행하려면 HostId-StatusDate-index 인덱스를 지정합니다.
    - 로그인한 사용자가 상대방인 최근 IN_PROGRESS 게임을 최대 10개까지 검색합니다. 이 쿼리를 수행하려면 OpponentId-StatusDate-index 인덱스를 지정합니다.

보조 인덱스에 대한 자세한 내용은 DynamoDB에서 보조 인덱스를 사용하여 데이터 액세스 개선 단원을 참조하세요.

## 2.2: 애플리케이션 실행(코드 단계별 안내)

이 애플리케이션에는 두 개의 메인 페이지가 있습니다.

- 홈 페이지 - 이 페이지는 사용자에게 간단한 로그인, 새 Tic-Tac-Toe 게임을 만들 수 있는 CREATE(생성) 버튼, 진행 중인 게임 목록, 게임 기록 및 활성 상태로 보류 중인 게임 초대를 제공합니다.

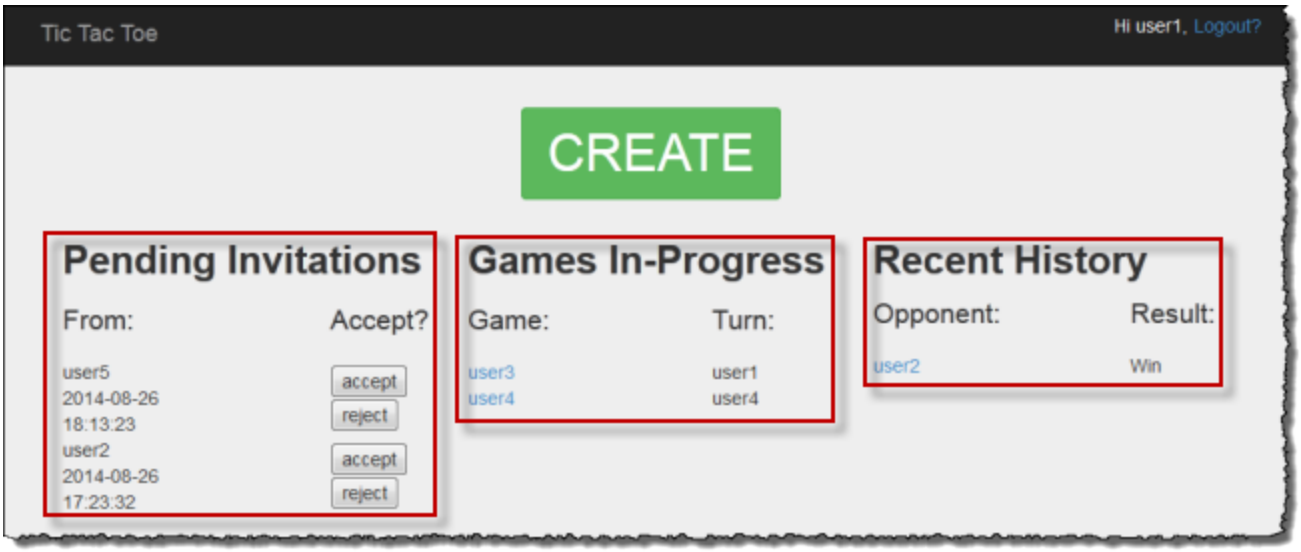
  홈 페이지는 자동으로 새로 고쳐지지 않으며 페이지를 새로 고쳐 목록을 새로 고쳐야 합니다.
- 게임 페이지 - 이 페이지에는 사용자가 플레이하는 Tic-Tac-Toe 그리드가 표시됩니다.

  애플리케이션은 매초 자동으로 게임 페이지를 업데이트합니다. 브라우저의 JavaScript가 매초 Python 웹 서버를 호출하여 Games 테이블을 쿼리하고 테이블의 게임 항목이 변경되었는지 여부를 확인합니다. 변경된 경우 JavaScript에서 사용자가 업데이트된 보드를 볼 수 있도록 페이지 새로 고침을 트리거합니다.

이제 애플리케이션의 작동 방식에 대해 자세히 설명하겠습니다.

홈 페이지

사용자가 로그인하면 애플리케이션에 다음과 같은 세 가지 정보 목록이 표시됩니다.



- 초대 - 이 목록에는 로그인한 사용자가 가장 최근에 다른 사용자로부터 받은 수락 보류 중인 초대가 최대 10개까지 표시됩니다. 위 스크린샷에서 user1에게는 user5와 user2가 보낸 보류 중인 초대가 있습니다.
- 진행 중인 게임 - 이 목록에는 진행 중인 최근 게임이 최대 10개까지 표시됩니다. 이러한 게임은 사용자가 현재 플레이하고 있고 상태가 IN_PROGRESS인 게임입니다. 스크린샷에서 user1은 user3 및 user4와 Tic-Tac-Toe 게임을 플레이하고 있습니다.
- 최근 기록 - 이 목록에는 사용자가 가장 최근에 완료하고 상태가 FINISHED인 게임이 최대 10개까지 표시됩니다. 스크린샷의 게임에서 user1은 이전에 user2와 플레이한 적이 있습니다. 완료된 각 게임에 대해 게임 결과가 표시됩니다.

코드에서 application.py의 index 함수는 다음과 같은 세 번의 호출로 게임 상태 정보를 검색합니다.

```
inviteGames     = controller.getGameInvites(session["username"])
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

각 호출에서 Game 객체로 래핑된 DynamoDB의 항목 목록이 반환됩니다. 보기에서 이러한 객체에서 쉽게 데이터를 추출할 수 있습니다. index 함수는 HTML을 렌더링하기 위해 이러한 객체 목록을 보기에 전달합니다.

```
return render_template("index.html",
                       user=session["username"],
                       invites=inviteGames,
                       inprogress=inProgressGames,
                       finished=finishedGames)
```

Tic-Tac-Toe 애플리케이션은 기본적으로 DynamoDB에서 가져온 게임 데이터를 저장하기 위해 Game 클래스를 정의합니다. 이러한 함수는 Amazon DynamoDB 항목과 관련된 코드에서 나머지 애플리케이션을 격리할 수 있는 Game 객체 목록을 반환합니다. 따라서 이러한 함수를 통해 데이터 스토어 계층의 세부 정보에서 애플리케이션 코드를 분리할 수 있습니다.

여기에서 설명하는 아키텍처 패턴은 MVC(모델-보기-컨트롤러) UI 패턴이라고도 합니다. 이 경우 데이터를 나타내는 Game 객체 인스턴스는 모델이며 HTML 페이지는 보기입니다. 컨트롤러는 두 개 파일로 나누어집니다. application.py 파일에는 Flask 프레임워크를 위한 컨트롤러 로직이 있으며 비즈니스 로직은 gameController.py 파일에 격리되어 있습니다. 즉, 애플리케이션은 DynamoDB SDK와 관련된 모든 항목을 dynamodb 폴더의 별도 파일에 저장합니다.

이제 세 가지 함수를 살펴보고 이러한 함수가 어떻게 글로벌 보조 인덱스를 사용하여 Games 테이블을 쿼리하고 관련 데이터를 검색하는지 알아보겠습니다.

getGameInvites를 사용하여 보류 중인 게임 초대 목록 불러오기

`getGameInvites` 함수는 가장 최근에 받은 보류 중인 초대 10개의 목록을 검색합니다. 이러한 게임은 사용자가 생성했지만 상대방이 초대를 수락하지 않은 게임입니다. 이러한 게임은 상대방이 초대를 수락할 때까지 상태가 `PENDING`으로 유지됩니다. 상대방이 초대를 거부할 경우, 애플리케이션이 테이블에서 해당 항목을 제거합니다.

이 함수는 다음과 같이 쿼리를 지정합니다.

- 다음 인덱스 키 값 및 비교 연산자에 사용할 `OpponentId-StatusDate-index` 인덱스를 지정합니다.
  - 파티션 키는 `OpponentId`이고 인덱스 키로 *`user ID`*를 사용합니다.
  - 정렬 키는 `StatusDate`이고 비교 연산자 및 인덱스 키 값으로 `beginswith="PENDING_"`을 사용합니다.

  `OpponentId-StatusDate-index` 인덱스를 사용하여 로그인한 사용자가 초대받은 게임, 즉 로그인한 사용자가 상대방인 게임을 검색합니다.
- 쿼리는 결과를 10개 항목으로 제한합니다.

```
gameInvitesIndex = self.cm.getGamesTable().query(
                                            Opponent__eq=user,
                                            StatusDate__beginswith="PENDING_",
                                            index="OpponentId-StatusDate-index",
                                            limit=10)
```

인덱스에서 각 `OpponentId`(파티션 키)에 대해 DynamoDB는 `StatusDate`(정렬 키)를 기준으로 항목을 정렬합니다. 따라서 쿼리가 반환하는 게임은 최근 10개 게임이 됩니다.

getGamesWithStatus를 사용하여 특정 상태의 게임 목록 불러오기

상대방이 게임 초대를 수락하면 게임 상태가 `IN_PROGRESS`로 변경됩니다. 게임이 완료된 다음에는 상태가 `FINISHED`로 변경됩니다.

진행 중이거나 완료된 게임을 검색하는 쿼리는 상태 값이 다른 것을 제외하고 동일합니다. 따라서 애플리케이션은 상태 값을 파라미터로 사용하는 `getGamesWithStatus` 함수를 정의합니다.

```
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

다음 단원은 진행 중인 게임에 대한 설명이지만 완료된 게임에도 동일하게 적용됩니다.

특정 사용자가 진행 중인 게임 목록에는 다음이 모두 포함됩니다.

- 사용자가 호스팅하여 진행 중인 게임
- 사용자가 상대방으로 진행 중인 게임

getGamesWithStatus 함수는 다음 두 쿼리를 실행하며 매번 적절한 보조 인덱스를 사용합니다.

- 이 함수는 HostId-StatusDate-index 인덱스를 사용하여 Games 테이블을 쿼리합니다. 인덱스의 경우 쿼리는 기본 키 값(파티션 키(HostId) 및 정렬 키(StatusDate) 값 모두)을 비교 연산자와 함께 지정합니다.

  ```
  hostGamesInProgress = self.cm.getGamesTable ().query(HostId__eq=user,
                                                       StatusDate__beginswith=status,
                                                       index="HostId-StatusDate-index",
                                                       limit=10)
  ```

  비교 연산자의 Python 구문은 다음과 같습니다.

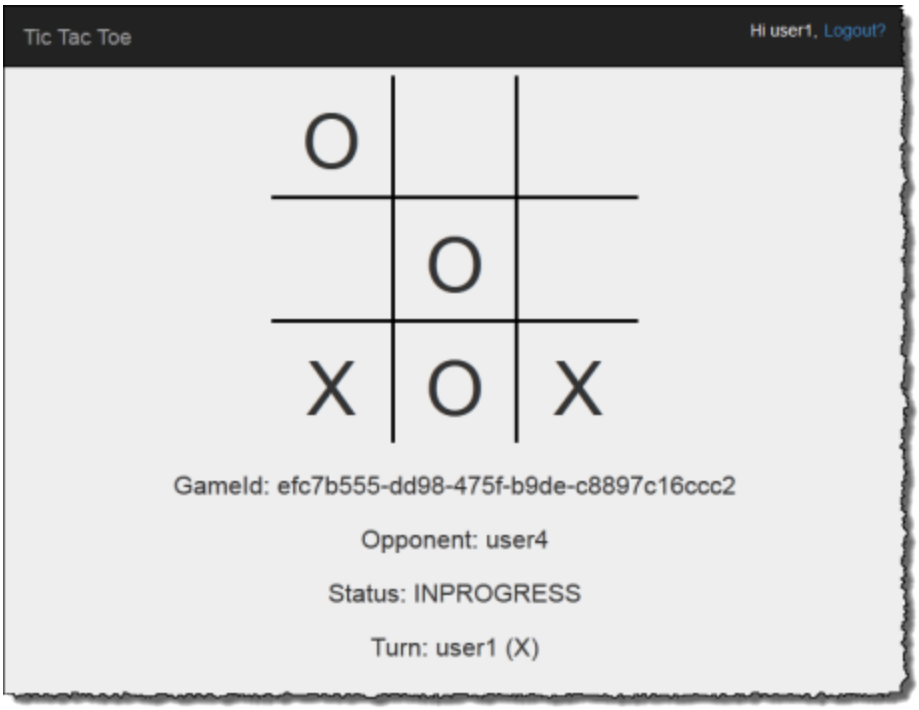
  - HostId__eq=user는 같음 비교 연산자를 지정하는 구문입니다.
  - StatusDate__beginswith=status는 BEGINS_WITH 비교 연산자를 지정하는 구문입니다.
- 이 함수는 OpponentId-StatusDate-index 인덱스를 사용하여 Games 테이블을 쿼리합니다.

  ```
  oppGamesInProgress = self.cm.getGamesTable().query(Opponent__eq=user,
                                                     StatusDate__beginswith=status,
                                                     index="OpponentId-StatusDate-index",
                                                     limit=10)
  ```
- 그런 다음 이 함수는 두 목록을 결합하고 정렬하여 첫 0번 ~ 10번 항목에 대해 Game 객체 목록을 만들고 호출 함수(인덱스)에 목록을 반환합니다.

  ```
  games = self.mergeQueries(hostGamesInProgress,
                            oppGamesInProgress)
  return games
  ```

### 게임 페이지

게임 페이지는 사용자가 Tic-Tac-Toe 게임을 플레이하는 곳입니다. 여기에는 게임 그리드와 함께 게임 관련 정보가 표시됩니다. 다음 스크린샷은 진행 중인 예제 게임을 보여줍니다.




애플리케이션은 다음 상황에서 게임 페이지를 표시합니다.

- 사용자가 게임을 만들고 플레이할 다른 사용자를 초대합니다.

  이 경우 페이지에 호스트로 사용자가 표시되고 게임 상태는 상대방의 수락을 기다리는 `PENDING`으로 표시됩니다.
- 사용자가 홈 페이지에서 대기 중인 초대 중 하나를 수락합니다.

  이 경우 페이지에 사용자가 상대방으로 표시되고 게임 상태는 `IN_PROGRESS`로 표시됩니다.

사용자가 보드에서 선택하면 애플리케이션에 대해 양식 `POST` 요청이 생성됩니다. 즉, Flask가 `application.py`에서 HTML 양식 데이터로 `selectSquare`를 호출합니다. 그 다음 이 함수가 `gameController.py`에서 `updateBoardAndTurn` 함수를 호출하여 다음과 같이 게임 항목을 업데이트합니다.

- 동작에 해당하는 새 속성을 추가합니다.
- `Turn` 속성 값을 다음 차례가 돌아온 사용자로 업데이트합니다.

```
controller.updateBoardAndTurn(item, value, session["username"])
```

이 함수는 항목 업데이트가 성공적일 경우 true를 반환하고 그렇지 않으면 false를 반환합니다. updateBoardAndTurn 함수에 대한 다음 내용을 참조하세요.

- 함수가 SDK for Python의 update_item 함수를 호출하여 기존 항목에 대해 유한한 업데이트 집합을 만듭니다. 이 함수는 DynamoDB의 UpdateItem 작업에 매핑됩니다. 자세한 내용은 UpdateItem 단원을 참조하세요.

  > **Note**
  >
  > UpdateItem과 PutItem 작업의 차이는 PutItem의 경우 전체 항목을 대체한다는 점입니다. 자세한 내용은 PutItem 단원을 참조하세요.

update_item 호출의 경우 코드가 다음을 식별합니다.

- Games 테이블의 기본 키(즉, ItemId)입니다.

  ```
  key = { "GameId" : { "S" : gameId } }
  ```

- 현재 사용자 동작에 해당하는 새로 추가할 속성 및 해당 값(예: TopLeft="X")

  ```
  attributeUpdates = {
      position : {
          "Action" : "PUT",
          "Value" : { "S" : representation }
      }
  }
  ```

- 업데이트되기 위해 충족되어야 하는 조건은 다음과 같습니다.
  - 게임이 진행 중이어야 합니다. 즉, StatusDate 속성이 IN_PROGRESS로 시작되어야 합니다.
  - 현재 차례가 Turn 속성에 지정된 대로 유효한 사용자 차례여야 합니다.
  - 사용자가 선택한 사각형이 사용 가능해야 합니다. 즉, 사각형에 해당하는 속성이 없어야 합니다.

  ```
  expectations = {"StatusDate" : {"AttributeValueList": [{"S" : "IN_PROGRESS_"}],
      "ComparisonOperator": "BEGINS_WITH"},
      "Turn" : {"Value" : {"S" : current_player}},
      position : {"Exists" : False}}
  ```

이제 함수가 update_item을 호출하여 항목을 업데이트합니다.

```
self.cm.db.update_item("Games", key=key,
    attribute_updates=attributeUpdates,
    expected=expectations)
```

함수가 반환되면 다음 예제와 같이 selectSquare 함수 호출이 리디렉션됩니다.

```
redirect("/game="+gameId)
```

이 호출에 따라 브라우저가 새로 고침 됩니다. 애플리케이션이 새로 고침 중에 게임이 승 또는 무승부로 종료되었는지 여부를 확인합니다. 확인을 마치면 애플리케이션에서는 그에 따라 게임 항목을 업데이트합니다.

## 3단계: DynamoDB 서비스를 사용한 프로덕션 내 배포

주제

- 3.1: Amazon EC2의 IAM 역할 생성
- 3.2: Amazon DynamoDB에서 Games 테이블 생성
- 3.3: Tic-Tac-Toe 애플리케이션 코드 번들링 및 배포
- 3.4: AWS Elastic Beanstalk 환경 설정

위 섹션에서는 DynamoDB Local을 사용하여 로컬 컴퓨터에 Tic-Tac-Toe 게임 애플리케이션을 배포 및 테스트했습니다. 이제 다음과 같이 프로덕션에 애플리케이션을 배포합니다.

- 웹 애플리케이션과 서비스의 배포 및 규모 조정을 위한 간편한 서비스인 AWS Elastic Beanstalk를 사용하여 애플리케이션을 배포합니다. 자세한 내용은 AWS Elastic Beanstalk에 Flask 애플리케이션 배포를 참조하세요.

  Elastic Beanstalk에서 하나 이상의 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스를 시작하면 Tic-Tac-Toe 애플리케이션을 실행할 Elastic Beanstalk를 통해 해당 인스턴스를 구성합니다.
- Amazon DynamoDB 서비스를 사용하여 컴퓨터에 로컬로가 아니라 AWS에 있는 Games 테이블을 만듭니다.

또한 권한을 구성해야 합니다. DynamoDB의 Games 테이블과 같이 사용자가 만드는 모든 AWS 리소스는 기본적으로 프라이빗입니다. 리소스 소유자, 즉, Games 테이블을 만든 AWS 계정만 이 테이블에 액

세스할 수 있습니다. 따라서, 사용자의 Tic-Tac-Toe 애플리케이션은 기본적으로 Games 테이블을 업데이트할 수 없습니다.

필요한 권한을 부여하려면 AWS Identity and Access Management(IAM) 역할을 만들고 이 역할에 Games 테이블 액세스 권한을 부여합니다. Amazon EC2 인스턴스가 먼저 이 역할을 맡습니다. 그 응답으로 AWS는 Amazon EC2 인스턴스가 Tic-Tac-Toe 애플리케이션을 대신하여 Games 테이블을 업데이트하는 데 사용할 수 있는 임시 보안 자격 증명을 반환합니다. Elastic Beanstalk 애플리케이션을 구성할 때 Amazon EC2 인스턴스가 맡을 수 있는 IAM 역할을 지정합니다. IAM 역할에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2의 IAM 역할을 참조하세요.

> **Note**
>
> Tic-Tac-Toe 애플리케이션의 Amazon EC2 인스턴스를 만들기 전에 우선 Elastic Beanstalk이 인스턴스를 만들 AWS 리전을 결정해야 합니다. Elastic Beanstalk 애플리케이션을 만든 다음에는 구성 파일에 동일한 리전 이름과 엔드포인트를 입력해야 합니다. Tic-Tac-Toe 애플리케이션은 이 파일의 정보를 사용하여 Games 테이블을 만들고 특정 AWS 리전의 후속 요청을 전송합니다. Elastic Beanstalk이 시작하는 DynamoDB Games 테이블 및 Amazon EC2 인스턴스는 동일한 리전에 있어야 합니다. 사용 가능한 리전의 목록은 Amazon Web Services 일반 참조에서 Amazon DynamoDB를 참조하세요.

프로덕션에 Tic-Tac-Toe 애플리케이션을 배포하는 단계를 요약하면 다음과 같습니다.

1. IAM 서비스를 사용하여 IAM 역할을 만듭니다. 이 역할에 정책을 연결하여 Games 테이블에 액세스하는 DynamoDB 작업을 위한 권한을 부여합니다.
2. Tic-Tac-Toe 애플리케이션 코드와 구성 파일의 번들을 생성한 후 .zip 파일을 만듭니다. 이 .zip 파일을 사용하여 서버에 배포할 Elastic Beanstalk에 Tic-Tac-Toe 애플리케이션 코드를 제공합니다. 번들 생성에 대한 자세한 AWS Elastic Beanstalk 개발자 안내서에서 애플리케이션 소스 번들 생성을 참조하세요.

   구성 파일(beanstalk.config)에 AWS 리전과 엔드포인트 정보를 입력합니다. Tic-Tac-Toe 애플리케이션은 이 정보를 사용하여 연결할 DynamoDB 리전을 확인합니다.
3. Elastic Beanstalk 환경을 설정합니다. Elastic Beanstalk이 Amazon EC2 인스턴스를 실행하고 해당 인스턴스에 Tic-Tac-Toe 애플리케이션 번들을 배포합니다. Elastic Beanstalk 환경이 준비되면 CONFIG_FILE 환경 변수를 추가하여 구성 파일 이름을 입력합니다.
4. DynamoDB 테이블을 생성합니다. Amazon DynamoDB 서비스를 사용하여 컴퓨터에서 로컬로가 아니라 AWS에 Games 테이블을 생성합니다. 이 테이블에는 문자열 형식의 GameId 파티션 키로 구성된 단순 기본 키가 있습니다.

5. 프로덕션에서 게임을 테스트합니다.

## 3.1: Amazon EC2의 IAM 역할 생성

Amazon EC2 유형의 IAM 역할을 만들면 Tic-Tac-Toe 애플리케이션을 실행 중인 Amazon EC2 인스턴스가 올바른 IAM 역할을 맡고 `Games` 테이블에 액세스하기 위한 애플리케이션 요청을 보낼 수 있습니다. 역할을 만들 때에는 [Custom Policy(사용자 지정 정책)] 옵션을 선택하고 다음 정책을 복사하여 붙여 넣습니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Action":[
            "dynamodb:ListTables"
         ],
         "Effect":"Allow",
         "Resource":"*"
      },
      {
         "Action":[
            "dynamodb:*"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:dynamodb:us-west-2:922852403271:table/Games",
            "arn:aws:dynamodb:us-west-2:922852403271:table/Games/index/*"
         ]
      }
   ]
}
```

자세한 지침은 IAM 사용 설명서의 AWS 서비스에 대한 역할 생성(AWS Management Console)을 참조하세요.

## 3.2: Amazon DynamoDB에서 Games 테이블 생성

DynamoDB의 Games 테이블은 게임 데이터를 저장합니다. 테이블이 존재하지 않을 경우, 애플리케이션이 해당 테이블을 만듭니다. 이 예제에서는 애플리케이션이 Games 테이블을 만들도록 합니다.

## 3.3: Tic-Tac-Toe 애플리케이션 코드 번들링 및 배포

이 예제의 단계를 따른 경우 이미 Tic-Tac-Toe 애플리케이션 다운로드를 완료한 상태여야 합니다. 그렇지 않을 경우 애플리케이션을 다운로드하고 로컬 컴퓨터에 모든 파일을 추출하세요. 지침은 1단계: 로컬 배포 및 테스트 단원을 참조하세요.

모든 파일을 추출하면 code 폴더가 나타납니다. 이 폴더를 Elastic Beanstalk에 전달하려면 이 폴더의 콘텐츠를 .zip 파일로 번들링해야 합니다. 우선 해당 폴더에 구성 파일을 추가합니다. 애플리케이션이 리전 및 엔드포인트 정보를 사용하여 지정된 리전에 DynamoDB 테이블을 만들고, 지정된 엔드포인트를 사용하여 후속 테이블 작업 요청을 보냅니다.

1. Tic-Tac-Toe 애플리케이션을 다운로드한 폴더로 전환합니다.
2. 애플리케이션 루트 폴더에서 다음 콘텐츠가 포함된 텍스트 파일을 만들고 이름을 beanstalk.config로 지정합니다.

   ```
   [dynamodb]
   region=<AWS region>
   endpoint=<DynamoDB endpoint>
   ```

   예를 들어 다음과 같은 콘텐츠를 사용할 수 있습니다.

   ```
   [dynamodb]
   region=us-west-2
   endpoint=dynamodb.us-west-2.amazonaws.com
   ```

   사용 가능한 리전 목록은 Amazon Web Services 일반 참조의 Amazon DynamoDB를 참조하세요.

   > **Important**
   >
   > 구성 파일에 지정된 리전은 Tic-Tac-Toe 애플리케이션이 DynamoDB에 Games 테이블을 만드는 위치입니다. 다음 단원에서 설명하는 Elastic Beanstalk 애플리케이션을 동일한 리전에 만들어야 합니다.

> **Note**
>
> Elastic Beanstalk 애플리케이션을 만들면 환경 시작을 요청하고 환경 유형을 선택할 수 있습니다. Tic-Tac-Toe 예제 애플리케이션을 테스트하려면 [단일 인스턴스] 환경 유형을 선택하여 아래 항목을 생략하고 다음 단계로 건너뛸 수 있습니다.
> 하지만 [로드 밸런싱, 자동 조정] 환경 유형은 가용성과 확장성이 높은 환경을 제공합니다. 다른 애플리케이션을 만들고 배포할 경우 이러한 환경을 사용하는 것이 좋습니다. 이 환경 유형을 선택할 경우, 다음과 같이 UUID를 만들고 구성 파일에 추가해야 합니다.
>
> ```
> [dynamodb]
> region=us-west-2
> endpoint=dynamodb.us-west-2.amazonaws.com
> [flask]
> secret_key= 284e784d-1a25-4a19-92bf-8eeb7a9example
> ```
>
> 클라이언트-서버 통신에서 서버가 응답을 보내면 보안상의 이유로 서버가 서명된 쿠키를 보내고 클라이언트는 다음 요청 시 이 쿠키를 다시 서버로 보냅니다. 서버가 한 대인 경우 서버가 시작될 때 로컬에서 암호화 키를 만들 수 있습니다. 서버가 여러 대일 경우 모두 동일한 암호화 키를 알아야 합니다. 그렇지 않으면 피어 서버가 설정한 쿠키를 읽을 수 없습니다. 구성 파일에 `secret_key`를 추가하면 모든 서버가 이 암호화 키를 사용하도록 알릴 수 있습니다.

3. 애플리케이션 루트 폴더의 내용(`beanstalk.config` 파일 포함)을 압축합니다(예: `TicTacToe.zip`).
4. `.zip` 파일을 Amazon Simple Storage Service(Amazon S3) 버킷에 업로드합니다. 다음 단원에서는 이 `.zip` 파일을 Elastic Beanstalk에 제공하여 한 서버 또는 여러 서버에 업로드합니다.

   Amazon S3 버킷에 업로드하는 방법에 관한 지침은 Amazon Simple Storage Service 사용 설명서에서 버킷 생성 및 버킷에 객체 추가 섹션을 참조하세요.

## 3.4: AWS Elastic Beanstalk 환경 설정

이 단계에서는 환경을 포함한 구성 요소의 컬렉션인 Elastic Beanstalk 애플리케이션을 만듭니다. 이 예제에서는 하나의 Amazon EC2 인스턴스를 시작하여 Tic-Tac-Toe 애플리케이션을 배포 및 실행합니다.

1. 다음 사용자 지정 URL을 입력하여 Elastic Beanstalk 콘솔을 설정하고 환경을 설정합니다.

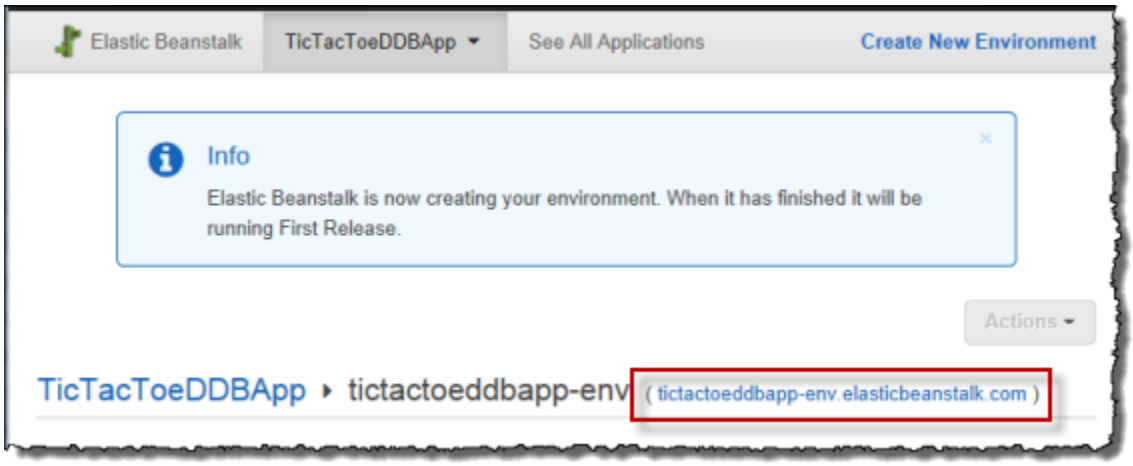
```
https://console.aws.amazon.com/elasticbeanstalk/?region=<AWS-Region>#/
newApplication
?applicationName=TicTacToeyour-name
&solutionStackName=Python
&sourceBundleUrl=https://s3.amazonaws.com/<bucket-name>/TicTacToe.zip
&environmentType=SingleInstance
&instanceType=t1.micro
```

   사용자 지정 URL에 대한 자세한 내용은 AWS Elastic Beanstalk 개발자 안내서에서 Launch Now URL 생성 단원을 참조하세요. URL에 대해 다음 사항을 참조하세요.

   - AWS 리전 이름(구성 파일에 입력한 이름과 동일), Amazon S3 버킷 이름, 객체 이름을 제공해야 합니다.
   - URL은 테스트를 위해 [SingleInstance] 환경 유형과 `t1.micro` 인스턴스 유형을 요청합니다.
   - 애플리케이션 이름은 고유해야 합니다. 따라서 이전 URL에서 `applicationName` 앞에 사용자 이름을 추가하는 것이 좋습니다.

   이렇게 하면 Elastic Beanstalk 콘솔이 열리며 로그인이 필요한 경우도 있습니다.

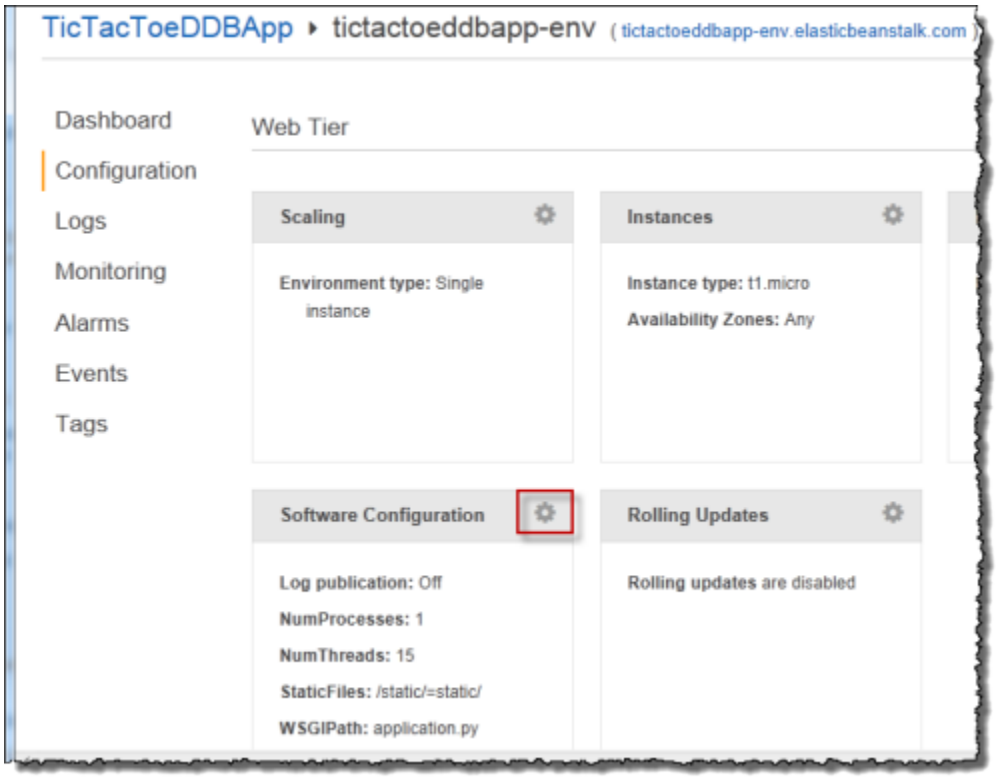
2. Elastic Beanstalk 콘솔에서 [Review and Launch]를 선택한 다음 [Launch]를 선택합니다.
3. 나중에 참조할 수 있도록 URL을 기록합니다. 이 URL에서 Tic-Tac-Toe 애플리케이션 홈 페이지가 열립니다.




4. Tic-Tac-Toe 애플리케이션이 구성 파일의 위치를 알 수 있도록 구성합니다.

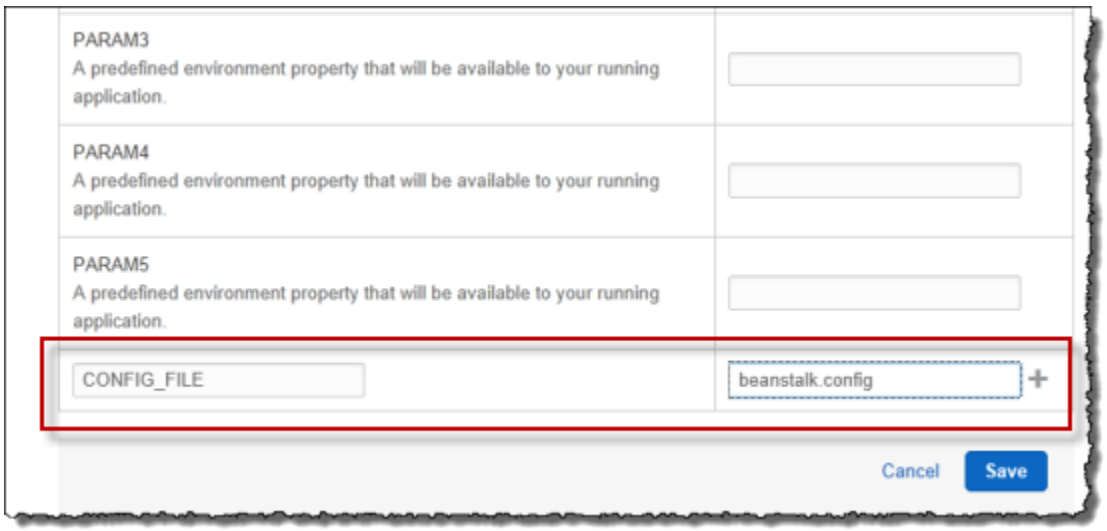
   Elastic Beanstalk가 애플리케이션을 만들면 [Configuration]을 선택합니다.

a. 다음 스크린샷과 같이 [Software Configuration] 옆에 있는 기어 모양 아이콘을 선택합니다.




b. [Environment Properties] 섹션의 끝부분에서 **`CONFIG_FILE`** 및 해당 값 **`beanstalk.config`**를 입력한 다음 [Save]를 선택합니다.

이 환경 업데이트가 완료되는 데 몇 분이 소요될 수 있습니다.

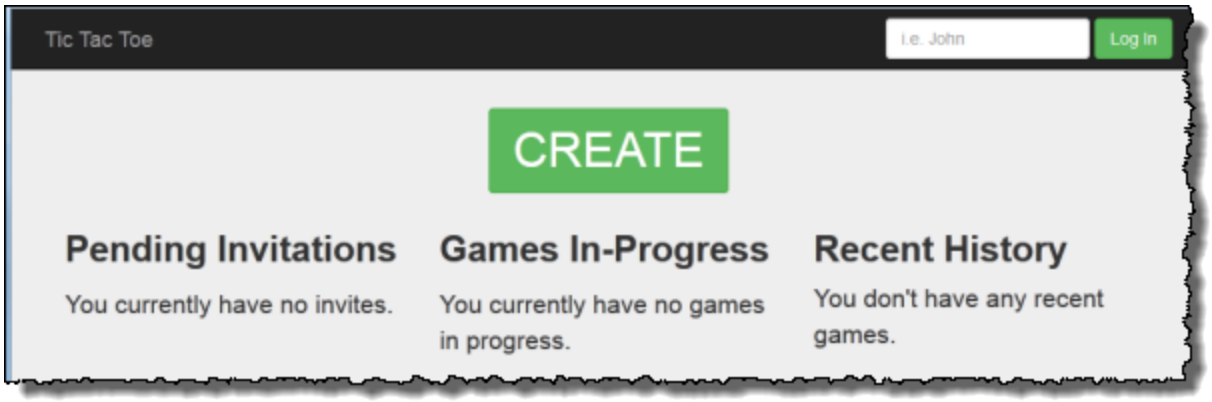



업데이트가 완료되면 게임을 플레이할 수 있습니다.

5. 다음 예제와 같이 브라우저에 이전 단계에서 복사한 URL을 입력합니다.

```
http://<pen-name>.elasticbeanstalk.com
```

그러면 애플리케이션 홈 페이지가 열립니다.




6. testuser1로 로그인하고 [CREATE]를 선택하여 새 Tic-Tac-Toe 게임을 시작합니다.
7. Choose an Opponent(상대방 선택) 상자에 **testuser2**를 입력합니다.




8. 다른 브라우저 창을 엽니다.

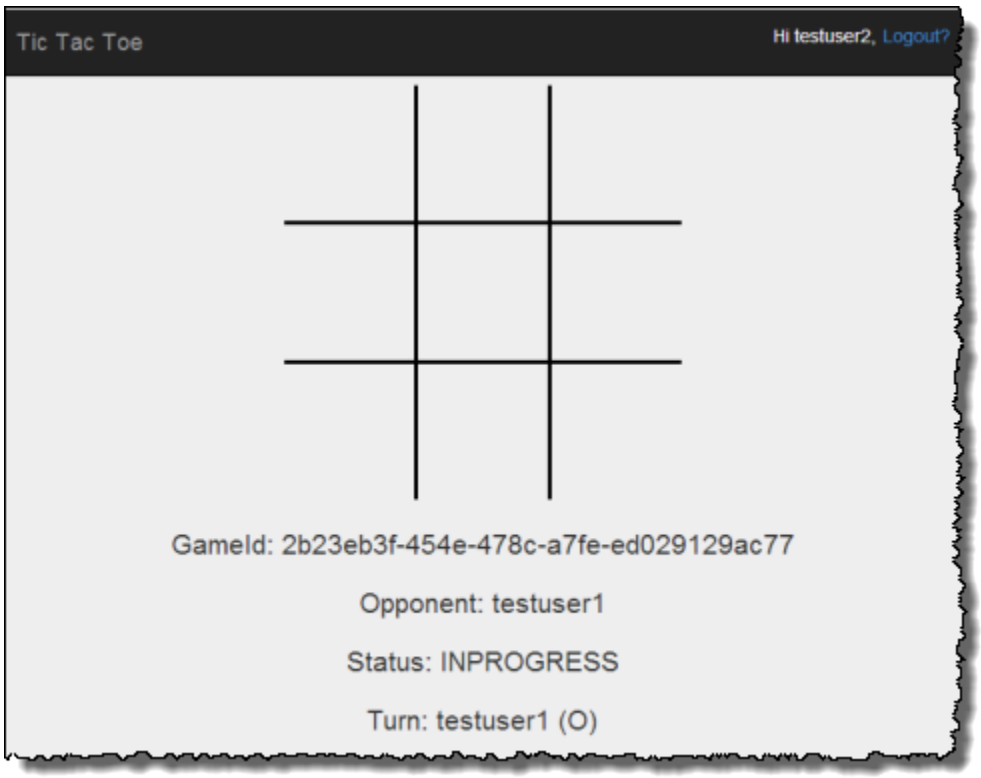
   동일한 사용자로 로그인되지 않도록 브라우저 창의 모든 쿠키를 지웠는지 반드시 확인하세요.
9. 다음 예제와 같이 동일한 URL을 입력하여 애플리케이션 홈 페이지를 엽니다.

```
http://<env-name>.elasticbeanstalk.com
```

10. testuser2로 로그인합니다.
11. 보류 중인 초대 목록에 있는 testuser1의 초대에 대해 [accept]를 선택합니다.




12. 이제 게임 페이지가 나타납니다.




testuser1 및 testuser2 모두 게임을 플레이할 수 있습니다. 동작이 일어날 때마다 애플리케이션 `Games` 테이블의 해당 항목에 저장합니다.

## 4단계: 리소스 정리

이제 Tic-Tac-Toe 애플리케이션 배포 및 테스트를 마쳤습니다. 이 애플리케이션은 사용자 인증을 제외하고 Amazon DynamoDB에서 웹 애플리케이션 개발을 전반적으로 다룹니다. 이 애플리케이션은 게임을 생성할 때 플레이어 이름을 추가하기 위해서만 홈 페이지에서 로그인 정보를 사용합니다. 프로덕션 애플리케이션에서는 필요한 코드를 추가하여 사용자 로그인 및 인증을 수행해야 합니다.

테스트를 마쳤으면 요금이 발생하지 않도록 Tic-Tac-Toe 애플리케이션 테스트를 위해 만든 리소스를 제거할 수 있습니다.

생성한 리소스를 제거하려면

1. DynamoDB에서 생성한 `Games` 테이블을 제거합니다.
2. Elastic Beanstalk 환경을 종료하여 Amazon EC2 인스턴스를 확보합니다.
3. 생성한 IAM 역할을 삭제합니다.
4. Amazon S3에 생성된 객체를 제거합니다.

# DynamoDB의 예약어

다음 키워드는 DynamoDB에서 사용하기 위해 예약되어 있습니다. 이러한 단어를 표현식에서 속성 이름으로 사용하지 마세요. 이 목록은 대/소문자를 구분하지 않습니다.

DynamoDB 예약어와 충돌하는 속성 이름을 포함한 식을 작성해야 하는 경우, 식 속성 이름을 정의하여 예약어 대신 사용할 수 있습니다. 자세한 내용은 DynamoDB의 표현식 속성 이름(별칭) 단원을 참조하십시오.

```
ABORT
ABSOLUTE
ACTION
ADD
AFTER
AGENT
AGGREGATE
ALL
ALLOCATE
ALTER
ANALYZE
AND
ANY
```

```
ARCHIVE
ARE
ARRAY
AS
ASC
ASCII
ASENSITIVE
ASSERTION
ASYMMETRIC
AT
ATOMIC
ATTACH
ATTRIBUTE
AUTH
AUTHORIZATION
AUTHORIZE
AUTO
AVG
BACK
BACKUP
BASE
BATCH
BEFORE
BEGIN
BETWEEN
BIGINT
BINARY
BIT
BLOB
BLOCK
BOOLEAN
BOTH
BREADTH
BUCKET
BULK
BY
BYTE
CALL
CALLED
CALLING
CAPACITY
CASCADE
CASCADED
CASE
```

```
CAST
CATALOG
CHAR
CHARACTER
CHECK
CLASS
CLOB
CLOSE
CLUSTER
CLUSTERED
CLUSTERING
CLUSTERS
COALESCE
COLLATE
COLLATION
COLLECTION
COLUMN
COLUMNS
COMBINE
COMMENT
COMMIT
COMPACT
COMPILE
COMPRESS
CONDITION
CONFLICT
CONNECT
CONNECTION
CONSISTENCY
CONSISTENT
CONSTRAINT
CONSTRAINTS
CONSTRUCTOR
CONSUMED
CONTINUE
CONVERT
COPY
CORRESPONDING
COUNT
COUNTER
CREATE
CROSS
CUBE
CURRENT
```

```
CURSOR
CYCLE
DATA
DATABASE
DATE
DATETIME
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DEFINE
DEFINED
DEFINITION
DELETE
DELIMITED
DEPTH
DEREF
DESC
DESCRIBE
DESCRIPTOR
DETACH
DETERMINISTIC
DIAGNOSTICS
DIRECTORIES
DISABLE
DISCONNECT
DISTINCT
DISTRIBUTE
DO
DOMAIN
DOUBLE
DROP
DUMP
DURATION
DYNAMIC
EACH
ELEMENT
ELSE
ELSEIF
EMPTY
```

```
ENABLE
END
EQUAL
EQUALS
ERROR
ESCAPE
ESCAPED
EVAL
EVALUATE
EXCEEDED
EXCEPT
EXCEPTION
EXCEPTIONS
EXCLUSIVE
EXEC
EXECUTE
EXISTS
EXIT
EXPLAIN
EXPLODE
EXPORT
EXPRESSION
EXTENDED
EXTERNAL
EXTRACT
FAIL
FALSE
FAMILY
FETCH
FIELDS
FILE
FILTER
FILTERING
FINAL
FINISH
FIRST
FIXED
FLATTERN
FLOAT
FOR
FORCE
FOREIGN
FORMAT
FORWARD
```

```
FOUND
FREE
FROM
FULL
FUNCTION
FUNCTIONS
GENERAL
GENERATE
GET
GLOB
GLOBAL
GO
GOTO
GRANT
GREATER
GROUP
GROUPING
HANDLER
HASH
HAVE
HAVING
HEAP
HIDDEN
HOLD
HOUR
IDENTIFIED
IDENTITY
IF
IGNORE
IMMEDIATE
IMPORT
IN
INCLUDING
INCLUSIVE
INCREMENT
INCREMENTAL
INDEX
INDEXED
INDEXES
INDICATOR
INFINITE
INITIALLY
INLINE
INNER
```

```
INNTER
INOUT
INPUT
INSENSITIVE
INSERT
INSTEAD
INT
INTEGER
INTERSECT
INTERVAL
INTO
INVALIDATE
IS
ISOLATION
ITEM
ITEMS
ITERATE
JOIN
KEY
KEYS
LAG
LANGUAGE
LARGE
LAST
LATERAL
LEAD
LEADING
LEAVE
LEFT
LENGTH
LESS
LEVEL
LIKE
LIMIT
LIMITED
LINES
LIST
LOAD
LOCAL
LOCALTIME
LOCALTIMESTAMP
LOCATION
LOCATOR
LOCK
```

```
LOCKS
LOG
LOGED
LONG
LOOP
LOWER
MAP
MATCH
MATERIALIZED
MAX
MAXLEN
MEMBER
MERGE
METHOD
METRICS
MIN
MINUS
MINUTE
MISSING
MOD
MODE
MODIFIES
MODIFY
MODULE
MONTH
MULTI
MULTISET
NAME
NAMES
NATIONAL
NATURAL
NCHAR
NCLOB
NEW
NEXT
NO
NONE
NOT
NULL
NULLIF
NUMBER
NUMERIC
OBJECT
OF
```

```
OFFLINE
OFFSET
OLD
ON
ONLINE
ONLY
OPAQUE
OPEN
OPERATOR
OPTION
OR
ORDER
ORDINALITY
OTHER
OTHERS
OUT
OUTER
OUTPUT
OVER
OVERLAPS
OVERRIDE
OWNER
PAD
PARALLEL
PARAMETER
PARAMETERS
PARTIAL
PARTITION
PARTITIONED
PARTITIONS
PATH
PERCENT
PERCENTILE
PERMISSION
PERMISSIONS
PIPE
PIPELINED
PLAN
POOL
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
```

```
PRIOR
PRIVATE
PRIVILEGES
PROCEDURE
PROCESSED
PROJECT
PROJECTION
PROPERTY
PROVISIONING
PUBLIC
PUT
QUERY
QUIT
QUORUM
RAISE
RANDOM
RANGE
RANK
RAW
READ
READS
REAL
REBUILD
RECORD
RECURSIVE
REDUCE
REF
REFERENCE
REFERENCES
REFERENCING
REGEXP
REGION
REINDEX
RELATIVE
RELEASE
REMAINDER
RENAME
REPEAT
REPLACE
REQUEST
RESET
RESIGNAL
RESOURCE
RESPONSE
```

```
RESTORE
RESTRICT
RESULT
RETURN
RETURNING
RETURNS
REVERSE
REVOKE
RIGHT
ROLE
ROLES
ROLLBACK
ROLLUP
ROUTINE
ROW
ROWS
RULE
RULES
SAMPLE
SATISFIES
SAVE
SAVEPOINT
SCAN
SCHEMA
SCOPE
SCROLL
SEARCH
SECOND
SECTION
SEGMENT
SEGMENTS
SELECT
SELF
SEMI
SENSITIVE
SEPARATE
SEQUENCE
SERIALIZABLE
SESSION
SET
SETS
SHARD
SHARE
SHARED
```

```
SHORT
SHOW
SIGNAL
SIMILAR
SIZE
SKEWED
SMALLINT
SNAPSHOT
SOME
SOURCE
SPACE
SPACES
SPARSE
SPECIFIC
SPECIFICTYPE
SPLIT
SQL
SQLCODE
SQLERROR
SQLEXCEPTION
SQLSTATE
SQLWARNING
START
STATE
STATIC
STATUS
STORAGE
STORE
STORED
STREAM
STRING
STRUCT
STYLE
SUB
SUBMULTISET
SUBPARTITION
SUBSTRING
SUBTYPE
SUM
SUPER
SYMMETRIC
SYNONYM
SYSTEM
TABLE
```

```
TABLESAMPLE
TEMP
TEMPORARY
TERMINATED
TEXT
THAN
THEN
THROUGHPUT
TIME
TIMESTAMP
TIMEZONE
TINYINT
TO
TOKEN
TOTAL
TOUCH
TRAILING
TRANSACTION
TRANSFORM
TRANSLATE
TRANSLATION
TREAT
TRIGGER
TRIM
TRUE
TRUNCATE
TTL
TUPLE
TYPE
UNDER
UNDO
UNION
UNIQUE
UNIT
UNKNOWN
UNLOGGED
UNNEST
UNPROCESSED
UNSIGNED
UNTIL
UPDATE
UPPER
URL
USAGE
```

```
USE
USER
USERS
USING
UUID
VACUUM
VALUE
VALUED
VALUES
VARCHAR
VARIABLE
VARIANCE
VARINT
VARYING
VIEW
VIEWS
VIRTUAL
VOID
WAIT
WHEN
WHENEVER
WHERE
WHILE
WINDOW
WITH
WITHIN
WITHOUT
WORK
WRAPPED
WRITE
YEAR
ZONE
```

# AWS SDK for Java 1.x 예제

이 단원에는 SDK for Java 1.x를 사용하는 DAX 애플리케이션의 예제 코드가 포함되어 있습니다.

주제

- AWS SDK for Java 1.x와 함께 DAX 사용
- DAX를 사용하도록 기존 SDK for Java 1.x 애플리케이션 수정
- SDK for Java 1.x를 사용하여 글로벌 보조 인덱스 쿼리

# AWS SDK for Java 1.x와 함께 DAX 사용

다음 절차를 따라 Amazon EC2 인스턴스에서 Amazon DynamoDB Accelerator(DAX)용 Java 샘플을 실행합니다.

> **Note**
>
> 이 지침은 AWS SDK for Java 1.x를 사용하는 애플리케이션에 대한 것입니다. AWS SDK for Java 2.x를 사용하는 애플리케이션의 경우 Java 및 DAX 단원을 참조하세요.

DAX에 대한 Java 샘플을 실행하려면

1. JDK(Java Development Kit)를 설치합니다.

   ```
   sudo yum install -y java-devel
   ```

2. AWS SDK for Java(`.zip` 파일)를 다운로드한 후 압축을 풉니다.

   ```
   wget http://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip

   unzip aws-java-sdk.zip
   ```

3. 최신 버전의 DAX Java 클라이언트(`.jar` 파일)를 다운로드합니다.

   ```
   wget http://dax-sdk.s3-website-us-west-2.amazonaws.com/java/DaxJavaClient-
   latest.jar
   ```

   > **Note**
   >
   > DAX SDK for Java용 클라이언트는 Apache Maven에서 받을 수 있습니다. 자세한 내용은 클라이언트를 Apache Maven 종속 항목으로 사용 단원을 참조하십시오.

4. `CLASSPATH` 변수를 설정합니다. 이 예제에서는 *`sdkVersion`*을 AWS SDK for Java의 실제 버전 번호로 바꿉니다(예: `1.11.112`).

   ```
   export SDKVERSION=sdkVersion
   ```

```
export CLASSPATH=$(pwd)/TryDax/java:$(pwd)/DaxJavaClient-latest.jar:$(pwd)/
aws-java-sdk-$SDKVERSION/lib/aws-java-sdk-$SDKVERSION.jar:$(pwd)/aws-java-sdk-
$SDKVERSION/third-party/lib/*
```

5. 샘플 프로그램 소스 코드(`.zip` 파일)를 다운로드합니다.

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

다운로드가 완료되면 소스 파일의 압축을 풉니다.

```
unzip TryDax.zip
```

6. Java 코드 디렉터리로 이동하여 다음과 같이 코드를 컴파일합니다.

```
cd TryDax/java/
javac TryDax*.java
```

7. 프로그램을 실행합니다.

```
java TryDax
```

다음과 유사한 출력 화면이 표시되어야 합니다.

```
Creating a DynamoDB client

Attempting to create table; please wait...
Successfully created table.  Table status: ACTIVE
Writing data to the table...
Writing 10 items for partition key: 1
Writing 10 items for partition key: 2
Writing 10 items for partition key: 3
Writing 10 items for partition key: 4
Writing 10 items for partition key: 5
Writing 10 items for partition key: 6
Writing 10 items for partition key: 7
Writing 10 items for partition key: 8
Writing 10 items for partition key: 9
Writing 10 items for partition key: 10

Running GetItem, Scan, and Query tests...
First iteration of each test will result in cache misses
```

```
Next iterations are cache hits

GetItem test - partition key 1 and sort keys 1-10
 Total time: 136.681 ms - Avg time: 13.668 ms
 Total time: 122.632 ms - Avg time: 12.263 ms
 Total time: 167.762 ms - Avg time: 16.776 ms
 Total time: 108.130 ms - Avg time: 10.813 ms
 Total time: 137.890 ms - Avg time: 13.789 ms
Query test - partition key 5 and sort keys between 2 and 9
 Total time: 13.560 ms - Avg time: 2.712 ms
 Total time: 11.339 ms - Avg time: 2.268 ms
 Total time: 7.809 ms - Avg time: 1.562 ms
 Total time: 10.736 ms - Avg time: 2.147 ms
 Total time: 12.122 ms - Avg time: 2.424 ms
Scan test - all items in the table
 Total time: 58.952 ms - Avg time: 11.790 ms
 Total time: 25.507 ms - Avg time: 5.101 ms
 Total time: 37.660 ms - Avg time: 7.532 ms
 Total time: 26.781 ms - Avg time: 5.356 ms
 Total time: 46.076 ms - Avg time: 9.215 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

`GetItem`, `Query` 및 `Scan` 테스트에 필요한 시간 정보(밀리초)를 기록해 둡니다.

8. 전 단계에서 DynamoDB 엔드포인트에 대해 프로그램을 실행했습니다. 이제 프로그램을 다시 실행하되, 이번에는 `GetItem`, `Query` 및 `Scan` 작업을 DAX 클러스터에서 처리합니다.

   DAX 클러스터의 엔드포인트를 정의하려면 다음 중 하나를 선택합니다.

   - Using the DynamoDB console(DynamoDB 콘솔 사용) - DAX 클러스터를 선택합니다. 다음 예제와 같이 클러스터 엔드포인트가 콘솔에 표시됩니다.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - AWS CLI 사용 - 다음 명령을 입력합니다.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     다음 예제와 같이 클러스터 엔드포인트가 출력에 표시됩니다.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

이제 프로그램을 다시 실행합니다. 이번에는 클러스터 엔드포인트를 명령줄 파라미터로 지정합니다.

```
java TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

나머지 출력을 확인하여 시간 정보를 기록해 둡니다. `GetItem`, `Query`, `Scan`에 대한 경과 시간은 DAX가 DynamoDB보다 현저히 적어야 합니다.

이 프로그램에 대한 자세한 내용은 다음 단원을 참조하세요.

- TryDax.java
- TryDaxHelper.java
- TryDaxTests.java

## 클라이언트를 Apache Maven 종속 항목으로 사용

애플리케이션에서 DAX SDK for Java용 클라이언트를 종속 항목으로 사용하려면 다음 단계를 따릅니다.

클라이언트를 Maven 종속 항목으로 사용하려면

1. Apache Maven을 다운로드하고 설치합니다. 자세한 내용은 Downloading Apache Maven 및 Installing Apache Maven을 참조하세요.
2. 애플리케이션의 POM(Project Object Model) 파일에 클라이언트 Maven 종속 항목을 추가합니다. 이 예제에서는 `x.x.x.x`를 클라이언트의 실제 버전 번호로 바꿉니다(예: `1.0.200704.0`).

```
<!--Dependency:-->
<dependencies>
    <dependency>
     <groupId>com.amazonaws</groupId>
```

```
      <artifactId>amazon-dax-client</artifactId>
      <version>x.x.x.x</version>
    </dependency>
</dependencies>
```

## TryDax.java

`TryDax.java` 파일에는 `main` 메서드가 포함되어 있습니다. 명령줄 파라미터 없이 프로그램을 실행하면 Amazon DynamoDB 클라이언트가 생성되며 모든 API 작업에 해당 클라이언트가 사용됩니다. 명령줄에 DynamoDB 액셀러레이터(DAX) 클러스터 엔드포인트를 지정하면 DAX 클라이언트도 생성되며 `GetItem`, `Query` 및 `Scan` 작업에 이 클라이언트가 사용됩니다.

여러 가지 방법으로 프로그램을 수정할 수 있습니다.

- DAX 클라이언트 대신 DynamoDB 클라이언트를 사용합니다. 자세한 내용은 Java 및 DAX 단원을 참조하십시오.
- 테스트 테이블에 다른 이름을 선택합니다.
- `helper.writeData` 파라미터를 변경하여 기록된 항목 수를 수정합니다. 두 번째 파라미터는 파티션 키 개수이며 세 번째 파라미터는 정렬 키 개수입니다. 기본적으로 프로그램은 파티션 키 값에 1~10개 항목을, 정렬 키 값에 1~10개 항목을 사용하여 총 100개의 항목을 테이블에 기록합니다. 자세한 내용은 TryDaxHelper.java 단원을 참조하십시오.
- `GetItem`, `Query` 및 `Scan` 테스트 개수를 수정하고 해당 파라미터를 수정합니다.
- `helper.createTable` 및 `helper.deleteTable`을 포함하는 행을 주석으로 처리합니다(프로그램을 실행할 때마다 테이블을 생성 및 삭제하지 않으려는 경우).

> **Note**
>
> 이 프로그램을 실행하려면 DAX SDK for Java 클라이언트와 AWS SDK for Java를 종속 항목으로 사용하도록 Maven을 설정합니다. 자세한 내용은 클라이언트를 Apache Maven 종속 항목으로 사용 단원을 참조하십시오.
> 또는 DAX Java 클라이언트와 AWS SDK for Java를 다운로드하여 해당 클래스 경로(classpath)에 포함할 수 있습니다. Java 및 DAX 변수를 설정하는 예제는 CLASSPATH 단원을 참조하세요.

```
public class TryDax {

    public static void main(String[] args) throws Exception {

        TryDaxHelper helper = new TryDaxHelper();
        TryDaxTests tests = new TryDaxTests();

        DynamoDB ddbClient = helper.getDynamoDBClient();
        DynamoDB daxClient = null;
        if (args.length >= 1) {
            daxClient = helper.getDaxClient(args[0]);
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        helper.createTable(tableName, ddbClient);
        System.out.println("Populating table...");
        helper.writeData(tableName, ddbClient, 10, 10);

        DynamoDB testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem, Scan, and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
        tests.getItemTest(tableName, testClient, 1, 10, 5);

        // Query
        tests.queryTest(tableName, testClient, 5, 2, 9, 5);

        // Scan
        tests.scanTest(tableName, testClient, 5);

        helper.deleteTable(tableName, ddbClient);
    }

}
```

## TryDaxHelper.java

`TryDaxHelper.java` 파일에는 유틸리티 메서드가 포함되어 있습니다.

`getDynamoDBClient` 및 `getDaxClient` 메서드는 Amazon DynamoDB 및 DynamoDB 액셀러레이터(DAX) 클라이언트를 제공합니다. 제어 플레인 작업(`CreateTable`, `DeleteTable`) 및 쓰기 작업의 경우 프로그램은 DynamoDB 클라이언트를 사용합니다. DAX 클러스터 엔드포인트를 지정하면 기본 프로그램에서 DAX 클라이언트를 생성하여 읽기 작업(`GetItem`, `Query`, `Scan`)을 수행합니다.

다른 `TryDaxHelper` 메서드(`createTable`, `writeData`, `deleteTable`)는 DynamoDB 테이블 및 해당 데이터 설정 및 삭제용입니다.

여러 가지 방법으로 프로그램을 수정할 수 있습니다.

- 테이블에 다른 할당 처리량 설정을 사용합니다.
- 기록된 각 항목의 크기를 수정합니다(`writeData` 메서드의 `stringSize` 변수 참조).
- `GetItem`, `Query` 및 `Scan` 테스트 개수를 수정하고 해당 파라미터를 수정합니다.
- `helper.CreateTable` 및 `helper.DeleteTable`을 포함하는 행을 주석으로 처리합니다(프로그램을 실행할 때마다 테이블을 생성 및 삭제하지 않으려는 경우).

> **Note**
>
> 이 프로그램을 실행하려면 DAX SDK for Java 클라이언트와 AWS SDK for Java를 종속 항목으로 사용하도록 Maven을 설정합니다. 자세한 내용은 클라이언트를 Apache Maven 종속 항목으로 사용 단원을 참조하십시오.
> 또는 DAX Java 클라이언트와 AWS SDK for Java를 모두 다운로드하여 클래스 경로(classpath)에 포함할 수 있습니다. Java 및 DAX 변수를 설정하는 예제는 CLASSPATH 단원을 참조하세요.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
```

```
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.util.EC2MetadataUtils;

public class TryDaxHelper {

    private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

    DynamoDB getDynamoDBClient() {
        System.out.println("Creating a DynamoDB client");
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
                .withRegion(region)
                .build();
        return new DynamoDB(client);
    }

    DynamoDB getDaxClient(String daxEndpoint) {
        System.out.println("Creating a DAX client with cluster endpoint " +
 daxEndpoint);
        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
        AmazonDynamoDB client = daxClientBuilder.build();
        return new DynamoDB(client);
    }

    void createTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("Attempting to create table; please wait...");

            table = client.createTable(tableName,
                    Arrays.asList(
                            new KeySchemaElement("pk", KeyType.HASH), // Partition key
                            new KeySchemaElement("sk", KeyType.RANGE)), // Sort key
                    Arrays.asList(
                            new AttributeDefinition("pk", ScalarAttributeType.N),
                            new AttributeDefinition("sk", ScalarAttributeType.N)),
                    new ProvisionedThroughput(10L, 10L));
            table.waitForActive();
            System.out.println("Successfully created table.  Table status: " +
                    table.getDescription().getTableStatus());
```

```
        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    void writeData(String tableName, DynamoDB client, int pkmax, int skmax) {
        Table table = client.getTable(tableName);
        System.out.println("Writing data to the table...");

        int stringSize = 1000;
        StringBuilder sb = new StringBuilder(stringSize);
        for (int i = 0; i < stringSize; i++) {
            sb.append('X');
        }
        String someData = sb.toString();

        try {
            for (Integer ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
                for (Integer isk = 1; isk <= skmax; isk++) {
                    table.putItem(new Item()
                            .withPrimaryKey("pk", ipk, "sk", isk)
                            .withString("someData", someData));
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
    }

    void deleteTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            table.delete();
            table.waitForDelete();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
```

```
            e.printStackTrace();
        }
    }

}
```

## TryDaxTests.java

`TryDaxTests.java` 파일에는 Amazon DynamoDB의 테스트 테이블에 읽기 작업을 수행하는 메서드가 포함되어 있습니다. 이러한 메서드는 액세스하는 데이터 양에 영향을 받지 않으므로(DynamoDB 클라이언트를 사용하든 DAX 클라이언트를 사용하든) 애플리케이션 로직을 수정할 필요가 없습니다.

여러 가지 방법으로 프로그램을 수정할 수 있습니다.

- `queryTest` 메서드를 수정하여 다른 `KeyConditionExpression`을 사용하도록 합니다.
- 일부 항목만 반환되도록 `scanTest` 메서드에 `ScanFilter`를 추가합니다.

> **Note**
>
> 이 프로그램을 실행하려면 DAX SDK for Java 클라이언트와 AWS SDK for Java를 종속 항목으로 사용하도록 Maven을 설정합니다. 자세한 내용은 클라이언트를 Apache Maven 종속 항목으로 사용 단원을 참조하십시오.
> 또는 DAX Java 클라이언트와 AWS SDK for Java를 모두 다운로드하여 클래스 경로(classpath)에 포함할 수 있습니다. Java 및 DAX 변수를 설정하는 예제는 `CLASSPATH` 단원을 참조하세요.

```
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.ScanOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;

public class TryDaxTests {
```

```
   void getItemTest(String tableName, DynamoDB client, int pk, int sk, int iterations)
{
       long startTime, endTime;
       System.out.println("GetItem test - partition key " + pk + " and sort keys 1-" +
sk);
       Table table = client.getTable(tableName);

       for (int i = 0; i < iterations; i++) {
           startTime = System.nanoTime();
           try {
               for (Integer ipk = 1; ipk <= pk; ipk++) {
                   for (Integer isk = 1; isk <= sk; isk++) {
                       table.getItem("pk", ipk, "sk", isk);
                   }
               }
           } catch (Exception e) {
               System.err.println("Unable to get item:");
               e.printStackTrace();
           }
           endTime = System.nanoTime();
           printTime(startTime, endTime, pk * sk);
       }
   }

   void queryTest(String tableName, DynamoDB client, int pk, int sk1, int sk2, int
iterations) {
       long startTime, endTime;
       System.out.println("Query test - partition key " + pk + " and sort keys between
" + sk1 + " and " + sk2);
       Table table = client.getTable(tableName);

       HashMap<String, Object> valueMap = new HashMap<String, Object>();
       valueMap.put(":pkval", pk);
       valueMap.put(":skval1", sk1);
       valueMap.put(":skval2", sk2);

       QuerySpec spec = new QuerySpec()
               .withKeyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
               .withValueMap(valueMap);

       for (int i = 0; i < iterations; i++) {
           startTime = System.nanoTime();
           ItemCollection<QueryOutcome> items = table.query(spec);
```

```
            try {
                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to query table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
    }

    void scanTest(String tableName, DynamoDB client, int iterations) {
        long startTime, endTime;
        System.out.println("Scan test - all items in the table");
        Table table = client.getTable(tableName);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            ItemCollection<ScanOutcome> items = table.scan();
            try {

                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to scan table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
    }

    public void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
(1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
1000000.0));
    }
```

```
}
```

## DAX를 사용하도록 기존 SDK for Java 1.x 애플리케이션 수정

Amazon DynamoDB를 사용하는 Java 애플리케이션이 이미 있는 경우 DynamoDB 액셀러레이터(DAX) 클러스터에 액세스할 수 있도록 이를 수정해야 합니다. DAX Java 클라이언트는 AWS SDK for Java에 포함되어 있는 DynamoDB 하위 수준 클라이언트와 매우 유사하므로 전체 애플리케이션을 다시 작성할 필요가 없습니다.

> **Note**
>
> 이 지침은 AWS SDK for Java 1.x를 사용하는 애플리케이션에 대한 것입니다. AWS SDK for Java 2.x를 사용하는 애플리케이션의 경우 DAX를 사용하도록 기존 애플리케이션 수정 단원을 참조하세요.

`Music`이라는 DynamoDB 테이블이 있다고 가정합니다. 테이블의 파티션 키는 `Artist`이고 정렬 키는 `SongTitle`입니다. 다음은 `Music` 테이블에서 직접 항목을 읽는 프로그램입니다.

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class GetMusicItem {

    public static void main(String[] args) throws Exception {

        // Create a DynamoDB client
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

        HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
        key.put("Artist", new AttributeValue().withS("No One You Know"));
        key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

        GetItemRequest request = new GetItemRequest()
            .withTableName("Music").withKey(key);
```

```
        try {
            System.out.println("Attempting to read the item...");
            GetItemResult result = client.getItem(request);
            System.out.println("GetItem succeeded: " + result);

        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }
    }
}
```

프로그램을 수정하기 위해 DynamoDB 클라이언트를 DAX 클라이언트로 바꿉니다.

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class GetMusicItem {

    public static void main(String[] args) throws Exception {

    //Create a DAX client

    AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
    daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
    AmazonDynamoDB client = daxClientBuilder.build();


       /*
       ** ...
       ** Remaining code omitted (it is identical)
       ** ...
       */

    }
}
```

## DynamoDB 문서 API 사용

AWS SDK for Java는 DynamoDB용 문서 인터페이스를 제공합니다. Document API는 하위 수준 DynamoDB 클라이언트를 둘러싼 래퍼 역할을 합니다. 자세한 내용은 문서 인터페이스를 참조하세요.

아래 예제와 같이 문서 인터페이스는 하위 수준 DAX 클라이언트와 함께 사용할 수도 있습니다.

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.GetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;

public class GetMusicItemWithDocumentApi {

    public static void main(String[] args) throws Exception {

        //Create a DAX client

        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
        AmazonDynamoDB client = daxClientBuilder.build();

        // Document client wrapper
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");

        try {
            System.out.println("Attempting to read the item...");
            GetItemOutcome outcome = table.tgetItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Scared of My Shadow");
            System.out.println(outcome.getItem());
            System.out.println("GetItem succeeded: " + outcome);
        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }

    }
}
```

## DAX 비동기 클라이언트

AmazonDaxClient는 동기식입니다. 라지 테이블의 Scan과 같이 실행 시간이 긴 DAX API 작업의 경우, 동기식은 작업이 완료될 때까지 프로그램 실행이 차단될 수 있습니다. 프로그램에서 DAX API 작업이 진행되는 중에 다른 작업을 수행해야 하는 경우 ClusterDaxAsyncClient를 대신 사용할 수 있습니다.

다음은 ClusterDaxAsyncClient와 Java Future를 함께 사용하여 비차단형 솔루션을 구현하는 방법을 보여주는 프로그램입니다.

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;

import com.amazon.dax.client.dynamodbv2.ClientConfig;
import com.amazon.dax.client.dynamodbv2.ClusterDaxAsyncClient;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.handlers.AsyncHandler;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBAsync;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class DaxAsyncClientDemo {
 public static void main(String[] args) throws Exception {

  ClientConfig daxConfig = new ClientConfig().withCredentialsProvider(new
 ProfileCredentialsProvider())
    .withEndpoints("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111");

  AmazonDynamoDBAsync client = new ClusterDaxAsyncClient(daxConfig);

  HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
  key.put("Artist", new AttributeValue().withS("No One You Know"));
  key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

  GetItemRequest request = new GetItemRequest()
    .withTableName("Music").withKey(key);

  // Java Futures
  Future<GetItemResult> call = client.getItemAsync(request);
  while (!call.isDone()) {
   // Do other processing while you're waiting for the response
   System.out.println("Doing something else for a few seconds...");
```

```
    Thread.sleep(3000);
   }
   // The results should be ready by now

   try {
    call.get();

   } catch (ExecutionException ee) {
    // Futures always wrap errors as an ExecutionException.
    // The *real* exception is stored as the cause of the
    // ExecutionException
    Throwable exception = ee.getCause();
    System.out.println("Error getting item: " + exception.getMessage());
   }

   // Async callbacks
   call = client.getItemAsync(request, new AsyncHandler<GetItemRequest, GetItemResult>()
  {

    @Override
    public void onSuccess(GetItemRequest request, GetItemResult getItemResult) {
     System.out.println("Result: " + getItemResult);
    }

    @Override
    public void onError(Exception e) {
     System.out.println("Unable to read item");
     System.err.println(e.getMessage());
     // Callers can also test if exception is an instance of
     // AmazonServiceException or AmazonClientException and cast
     // it to get additional information
    }

   });
   call.get();

  }
}
```

# SDK for Java 1.x를 사용하여 글로벌 보조 인덱스 쿼리

Amazon DynamoDB Accelerator(DAX)를 사용하면 DynamoDB 프로그래밍 인터페이스를 사용하여 글로벌 보조 인덱스를 쿼리할 수 있습니다.

다음 예에서는 DAX를 사용하여 예: AWS SDK for Java 문서 API를 사용하는 글로벌 보조 인덱스에서 생성된 `CreateDateIndex` 글로벌 보조 인덱스를 쿼리하는 방법을 설명합니다.

`DAXClient` 클래스는 DynamoDB 프로그래밍 인터페이스와 상호 작용하는 데 필요한 클라이언트 객체를 인스턴스화합니다.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import software.amazon.dynamodb.document.DynamoDB;
import com.amazonaws.util.EC2MetadataUtils;
import software.amazon.dynamodb.AmazonDynamoDB;

public class DaxClient {

 private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

 DynamoDB getDaxDocClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDB(client);
 }

 DynamoDBMapper getDaxMapperClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDBMapper(client);
 }
}
```

다음 방법으로 글로벌 보조 인덱스를 쿼리할 수 있습니다.

- 다음 예제에서 정의된 QueryIndexDax 클래스의 queryIndex 메서드를 사용합니다. QueryIndexDax는 DaxClient 클래스의 getDaxDocClient 메서드에서 반환된 클라이언트 객체를 파라미터로 사용합니다.
- 객체 지속성 인터페이스를 사용하고 있는 경우 다음 예제에서 정의된 QueryIndexDax 클래스의 queryIndexMapper 메서드를 사용합니다. queryIndexMapper는 DaxClient 클래스에 정의된 getDaxMapperClient 메서드에서 반환된 클라이언트 객체를 파라미터로 사용합니다.

```
import java.util.Iterator;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import java.util.List;
import software.amazon.dynamodb.datamodeling.DynamoDBQueryExpression;
import software.amazon.dynamodb.model.AttributeValue;
import java.util.HashMap;
import software.amazon.dynamodb.document.Item;
import software.amazon.dynamodb.document.utils.ValueMap;
import software.amazon.dynamodb.document.spec.QuerySpec;
import software.amazon.dynamodb.document.QueryOutcome;
import software.amazon.dynamodb.document.ItemCollection;
import software.amazon.dynamodb.document.Index;
import software.amazon.dynamodb.document.Table;
import software.amazon.dynamodb.document.DynamoDB;

public class QueryIndexDax {

 //This is used to query Index using the low-level interface.
 public static void queryIndex(DynamoDB client, String tableName, String indexName) {
  Table table = client.getTable(tableName);

  System.out.println("\n***********************************************************
\n");
  System.out.print("Querying index " + indexName + "...");

  Index index = table.getIndex(indexName);

  ItemCollection<QueryOutcome> items = null;

  QuerySpec querySpec = new QuerySpec();

  if (indexName == "CreateDateIndex") {
   System.out.println("Issues filed on 2013-11-01");
```

```
   querySpec.withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
     .withValueMap(new ValueMap().withString(":v_date",
"2013-11-01").withString(":v_issue", "A-"));
   items = index.query(querySpec);
  } else {
   System.out.println("\nNo valid index name provided");
   return;
  }

  Iterator<Item> iterator = items.iterator();

  System.out.println("Query: printing results...");

  while (iterator.hasNext()) {
   System.out.println(iterator.next().toJSONPretty());
  }

}

//This is used to query Index using the high-level mapper interface.
public static void queryIndexMapper(DynamoDBMapper mapper, String tableName, String
indexName) {
 HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
 eav.put(":v_date", new AttributeValue().withS("2013-11-01"));
 eav.put(":v_issue", new AttributeValue().withS("A-"));
 DynamoDBQueryExpression<CreateDate> queryExpression = new
DynamoDBQueryExpression<CreateDate>()
    .withIndexName("CreateDateIndex").withConsistentRead(false)
    .withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
    .withExpressionAttributeValues(eav);

 List<CreateDate> items = mapper.query(CreateDate.class, queryExpression);
 Iterator<CreateDate> iterator = items.iterator();

 System.out.println("Query: printing results...");

 while (iterator.hasNext()) {
  CreateDate iterObj = iterator.next();
  System.out.println(iterObj.getCreateDate());
  System.out.println(iterObj.getIssueId());
 }
}
```

```
}
```

아래의 클래스 정의는 Issues 테이블을 나타내며 queryIndexMapper 메서드에서 사용됩니다.

```
import software.amazon.dynamodb.datamodeling.DynamoDBTable;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexRangeKey;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;

@DynamoDBTable(tableName = "Issues")
public class CreateDate {
 private String createDate;
 @DynamoDBHashKey(attributeName = "IssueId")
 private String issueId;

 @DynamoDBIndexHashKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "CreateDate")
 public String getCreateDate() {
  return createDate;
 }

 public void setCreateDate(String createDate) {
  this.createDate = createDate;
 }

 @DynamoDBIndexRangeKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "IssueId")
 public String getIssueId() {
  return issueId;
 }

 public void setIssueId(String issueId) {
  this.issueId = issueId;
 }
}
```

# AWS SDK for Go 1.x 예제

이 섹션에는 Go 1.x를 사용하는 DAX 애플리케이션의 예제 코드가 포함되어 있습니다.

주제

- DAX SDK for Go

# DAX SDK for Go

다음 절차에 따라 Amazon EC2 인스턴스에서 Amazon DynamoDB Accelerator(DAX) SDK for Go 샘플 애플리케이션을 실행합니다.

DAX용 SDK for Go 샘플을 실행하려면

1. Amazon EC2 인스턴스에 SDK for Go를 설치합니다.

   a. Go 프로그래밍 언어(Golang)를 설치합니다.

   ```
   sudo yum install -y golang
   ```

   b. Golang가 설치되었고 올바르게 실행되는지 테스트합니다.

   ```
   go version
   ```

   다음과 같은 메시지가 나타납니다.

   ```
   go version go1.15.5 linux/amd64
   ```

   나머지 지침에서는 Go 버전 1.13에서 기본값이 된 모듈 지원을 사용합니다.

2. 샘플 Golang 애플리케이션을 설치합니다.

   ```
   go get github.com/aws-samples/aws-dax-go-sample
   ```

3. 다음 Golang 프로그램을 실행합니다. 첫 프로그램은 `TryDaxGoTable`이라는 DynamoDB 테이블을 생성합니다. 두 번째 프로그램은 해당 테이블에 데이터를 씁니다.

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
   service dynamodb -command create-table
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
   service dynamodb -command put-item
   ```

4. 다음 Golang 프로그램을 실행합니다.

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
   service dynamodb -command get-item
   ```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command query
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command scan
```

`GetItem`, `Query` 및 `Scan` 테스트에 필요한 시간 정보(밀리초)를 기록해 둡니다.

5. 전 단계에서 DynamoDB 엔드포인트에 대해 프로그램을 실행했습니다. 이제 프로그램을 다시 실행하되, 이번에는 `GetItem`, `Query` 및 `Scan` 작업을 DAX 클러스터에서 처리합니다.

   DAX 클러스터의 엔드포인트를 정의하려면 다음 중 하나를 선택합니다.

   - Using the DynamoDB console(DynamoDB 콘솔 사용) - DAX 클러스터를 선택합니다. 다음 예제와 같이 클러스터 엔드포인트가 콘솔에 표시됩니다.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - AWS CLI 사용 - 다음 명령을 입력합니다.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     다음 예제와 같이 클러스터 엔드포인트가 출력에 표시됩니다.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   이제 프로그램을 다시 실행합니다. 이번에는 클러스터 엔드포인트를 명령줄 파라미터로 지정합니다.

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
    -service dax -command get-item -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
 -service dax -command query -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
 -service dax -command scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

나머지 출력을 확인하여 시간 정보를 기록해 둡니다. `GetItem`, `Query`, `Scan`에 대한 경과 시간은 DAX가 DynamoDB보다 현저히 적어야 합니다.

6. 다음 Golang 프로그램을 실행하여 `TryDaxGoTable`을 삭제합니다.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command delete-table
```

# AWS SDK for Node.js 2.x 예제

이 섹션에는 AWS SDK for Node.js 2.x를 사용하는 DAX 애플리케이션의 예제 코드가 포함되어 있습니다.

주제

- Node.js 및 DAX

## Node.js 및 DAX

다음 단계에 따라 Amazon EC2 인스턴스에서 Node.js 샘플 애플리케이션을 실행합니다.

DAX에 대한 Node.js 샘플을 실행하려면

1. 다음과 같이 Amazon EC2 인스턴스에서 Node.js를 설치합니다.

   a. 노드 버전 관리자(nvm)를 설치합니다.

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
   ```

   b. nvm을 사용하여 Node.js를 설치합니다.

```
nvm install 12.16.3
```

c. Node.js가 설치되었고 올바르게 실행되는지 테스트합니다.

```
node -e "console.log('Running Node.js ' + process.version)"
```

다음 메시지가 표시되어야 합니다.

```
Running Node.js v12.16.3
```

2. 노드 패키지 관리자(npm)를 사용하여 DAX Node.js 클라이언트를 설치합니다.

```
npm install amazon-dax-client
```

3. 샘플 프로그램 소스 코드(`.zip` 파일)를 다운로드합니다.

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

다운로드가 완료되면 소스 파일의 압축을 풉니다.

```
unzip TryDax.zip
```

4. 다음 Node.js 프로그램을 실행합니다. 첫 프로그램은 `TryDaxTable`이라는 Amazon DynamoDB 테이블을 생성합니다. 두 번째 프로그램은 해당 테이블에 데이터를 씁니다.

```
node 01-create-table.js
node 02-write-data.js
```

5. 다음 Node.js 프로그램을 실행합니다.

```
node 03-getitem-test.js
node 04-query-test.js
node 05-scan-test.js
```

`GetItem`, `Query` 및 `Scan` 테스트에 필요한 시간 정보(밀리초)를 기록해 둡니다.

6. 전 단계에서 DynamoDB 엔드포인트에 대해 프로그램을 실행했습니다. 프로그램을 다시 실행하되, 이번에는 `GetItem`, `Query` 및 `Scan` 작업을 DAX 클러스터에서 처리합니다.

DAX 클러스터의 엔드포인트를 결정하려면 다음 중 하나를 선택합니다.

- Using the DynamoDB console(DynamoDB 콘솔 사용) - DAX 클러스터를 선택합니다. 다음 예제와 같이 클러스터 엔드포인트가 콘솔에 표시됩니다.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- AWS CLI 사용 - 다음 명령을 입력합니다.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

다음 예제와 같이 클러스터 엔드포인트가 출력에 표시됩니다.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

이제 프로그램을 다시 실행합니다. 이번에는 클러스터 엔드포인트를 명령줄 파라미터로 지정합니다.

```
node 03-getitem-test.js dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
node 04-query-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
node 05-scan-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

나머지 출력을 확인하여 시간 정보를 기록해 둡니다. `GetItem`, `Query`, `Scan`에 대한 경과 시간은 DAX가 DynamoDB보다 현저히 적어야 합니다.

7. 다음 Node.js 프로그램을 실행하여 `TryDaxTable`을 삭제합니다.

```
node 06-delete-table
```

이러한 프로그램에 대한 자세한 내용은 다음 단원을 참조하세요.

- 01-create-table.js

- 02-write-data.js
- 03-getitem-test.js
- 04-query-test.js
- 05-scan-test.js
- 06-delete-table.js

## 01-create-table.js

`01-create-table.js` 프로그램은 테이블(`TryDaxTable`)을 만듭니다. 이 단원의 나머지 Node.js 프로그램은 이 테이블을 사용합니다.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
  KeySchema: [
    { AttributeName: "pk", KeyType: "HASH" }, //Partition key
    { AttributeName: "sk", KeyType: "RANGE" }, //Sort key
  ],
  AttributeDefinitions: [
    { AttributeName: "pk", AttributeType: "N" },
    { AttributeName: "sk", AttributeType: "N" },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 10,
    WriteCapacityUnits: 10,
  },
};

dynamodb.createTable(params, function (err, data) {
```

```
  if (err) {
    console.error(
      "Unable to create table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
    console.log(
      "Created table. Table description JSON:",
      JSON.stringify(data, null, 2)
    );
  }
});
```

## 02-write-data.js

`02-write-data.js` 프로그램은 테스트 데이터를 `TryDaxTable`에 씁니다.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();

var tableName = "TryDaxTable";

var someData = "X".repeat(1000);
var pkmax = 10;
var skmax = 10;

for (var ipk = 1; ipk <= pkmax; ipk++) {
  for (var isk = 1; isk <= skmax; isk++) {
    var params = {
      TableName: tableName,
      Item: {
        pk: ipk,
        sk: isk,
        someData: someData,
```

```
      },
    };

    //
    //put item

    ddbClient.put(params, function (err, data) {
      if (err) {
        console.error("Unable to write data: ", JSON.stringify(err, null, 2));
      } else {
        console.log("PutItem succeeded");
      }
    });
  }
}
```

## 03-getitem-test.js

`03-getitem-test.js` 프로그램은 `GetItem`에서 `TryDaxTable` 작업을 수행합니다.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";
```

```
var pk = 1;
var sk = 10;
var iterations = 5;

for (var i = 0; i < iterations; i++) {
  var startTime = new Date().getTime();

  for (var ipk = 1; ipk <= pk; ipk++) {
    for (var isk = 1; isk <= sk; isk++) {
      var params = {
        TableName: tableName,
        Key: {
          pk: ipk,
          sk: isk,
        },
      };

      client.get(params, function (err, data) {
        if (err) {
          console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
        } else {
          // GetItem succeeded
        }
      });
    }
  }

  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

## 04-query-test.js

`04-query-test.js` 프로그램은 Query에서 `TryDaxTable` 작업을 수행합니다.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 5;
var sk1 = 2;
var sk2 = 9;
var iterations = 5;

var params = {
  TableName: tableName,
  KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
  ExpressionAttributeValues: {
    ":pkval": pk,
    ":skval1": sk1,
    ":skval2": sk2,
  },
};

for (var i = 0; i < iterations; i++) {
  var startTime = new Date().getTime();
```

```
  client.query(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
    } else {
      // Query succeeded
    }
  });

  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

## 05-scan-test.js

`05-scan-test.js` 프로그램은 Scan에서 `TryDaxTable` 작업을 수행합니다.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
```

```
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var iterations = 5;

var params = {
  TableName: tableName,
};
var startTime = new Date().getTime();
for (var i = 0; i < iterations; i++) {
  client.scan(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
    } else {
      // Scan succeeded
    }
  });
}

var endTime = new Date().getTime();
console.log(
  "\tTotal time: ",
  endTime - startTime,
  "ms - Avg time: ",
  (endTime - startTime) / iterations,
  "ms"
);
```

## 06-delete-table.js

`06-delete-table.js` 프로그램은 `TryDaxTable`을 삭제합니다. 테스트를 완료한 후 이 프로그램을 실행합니다.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");
```

```
var region = "us-west-2";

AWS.config.update({
  region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
};

dynamodb.deleteTable(params, function (err, data) {
  if (err) {
    console.error(
      "Unable to delete table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
    console.log(
      "Deleted table. Table description JSON:",
      JSON.stringify(data, null, 2)
    );
  }
});
```

# DynamoDB 문서 기록

다음 표에는 2018년 7월 3일부터 DynamoDB 개발자 안내서의 각 릴리스에서 변경된 중요 사항이 나와 있습니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드(이 페이지의 왼쪽 상단 모서리 부분에 있음)를 구독하면 됩니다.

| 변경 사항 | 설명 | 날짜 |
| --- | --- | --- |
| AWS SDK for Java V1에서 V2로 마이그레이션하는 DynamoDB 로컬 버전 3.0 릴리스 | 이 업데이트는 V1 종속성을 제거하고 성능을 개선하며 AWS SDK V2 패키지 구조에 부합합니다. | 2025년 7월 17일 |
| DynamoDB Streams에서 최적화된 샤드 검색에 대한 지원 추가 | 새로운 샤드 검색 메커니즘은 초기 검색 시간을 줄이고, 안정 상태 처리 성능을 개선하고, 처리 상태에 대한 가시성을 제공하고, 대규모 작업 전반에서 일관된 p100 지연 시간을 유지합니다. 자세한 내용은 AWS PrivateLink for Amazon DynamoDB Streams를 참조하세요. | 2025년 7월 17일 |
| DynamoDB 글로벌 테이블에 대한 다중 리전 강력한 일관성(MRSC) 지원 추가 | DynamoDB는 이제 선택적 감시자 리전이 있는 글로벌 테이블에 대해 다중 리전 강력한 일관성(MRSC)을 지원합니다. MRSC는 리전 장애 발생 시 목표 복구 시점(RPO)이 0인 리전 간 강력하게 일관된 읽기를 허용합니다. MRSC 글로벌 테이블은 정확히 세 개의 리전에 배포해야 하며, 세 개의 복제본 또는 두 개의 복제본과 하나의 감시자 구성이어야 합니다. 자세한 내용과 사용 가능한 리전은 | 2025년 6월 30일 |

| | | |
|---|---|---|
| | Multi-Region strong consistency (MRSC)를 참조하세요. | |
| Amazon DynamoDB Streams Kinesis Client Library(KCL) 2.x 어댑터 | DynamoDB는 이제 KCL 2.x 및 AWS SDK v2와 호환되는 업그레이드된 DynamoDB Streams 어댑터를 지원합니다. 자세한 내용은 KCL1.x에서 KCL 3.x로 마이그레이션을 참조하세요. | 2025년 6월 11일 |
| AWS CloudShell에서 DynamoDB 로컬 실행에 대한 지원 추가 | 이제 소프트웨어를 다운로드하거나 설치하지 않고도 AWS Management Console을 통해 브라우저에서 DynamoDB 로컬을 직접 실행할 수 있습니다. 자세한 내용은 컴퓨터에 로컬로 DynamoDB 배포를 참조하세요. | 2025년 5월 19일 |
| DAX의 r7i 인스턴스 유형에 대한 지원 추가 | DAX의 인스턴스 유형 옵션을 확장하여 성능과 용량 측면의 유연성을 높였습니다. 새로운 r7i 인스턴스 유형 및 기능에 대한 자세한 내용은 DAX 저장 데이터 암호화를 참조하세요. | 2025년 4월 29일 |
| 범위가 축소된 새로운 전체 액세스 정책 `AmazonDynamoDBFullAccess_v2` 로 `AmazonDynamoDBFullAccess` 대체 | 2025년 4월 25일부터 DynamoDB에서 `AmazonDynamoDBFullAccess` 정책이 더 이상 사용되지 않으며 `AmazonDynamoDBFullAccess_v2` 라는 범위 축소 정책으로 대체되었습니다. 자세한 내용은 AWS 관리형 정책: AmazonDynamoDBFullAccess_v2를 참조하세요. | 2025년 4월 28일 |

| | | |
|---|---|---|
| DynamoDB에서 DynamoDB Streams에 대한 AWS PrivateLink 지원 도입 | AWS에 DynamoDB Streams에 대한 PrivateLink 지원이 추가되어 고객이 프라이빗 네트워크 연결을 사용하여 Amazon Virtual Private Cloud(VPC) 내에서 DynamoDB Streams API를 간접적으로 호출할 수 있습니다. 이 기능을 사용하면 퍼블릭 인터넷을 통과하지 않고도 프라이빗 네트워크 액세스를 간소화하여 DynamoDB 워크로드에 대한 규정 준수 및 보안 요구 사항을 지원할 수 있습니다. 자세한 내용은 AWS PrivateLink for Amazon DynamoDB Streams를 참조하세요. | 2025년 3월 26일 |
| 이제 JS 3.x 및 Go 2.x용 DAX SDK 사용 가능 | 이제 JS 3.x용 DynamoDB Accelerator(DAX) SDK가 사용 가능하며 Java 3.x용 AWS SDK와 호환됩니다. | 2025년 3월 17일 |
| 기본 테이블 설정을 위한 NoSQL Workbench 3.13.5 온디맨드 용량 모드 릴리스 | 기본 설정으로 테이블을 생성하면 DynamoDB가 프로비저닝된 용량 모드 대신 온디맨드 용량 모드를 사용하는 테이블을 생성합니다. | 2025년 2월 24일 |
| DAX 클라이언트 구성에 대한 새로운 모범 사례 | DAX 클라이언트 구성에 대한 새로운 DAX 권장 가이드 모범 사례 주제를 게시했습니다. 자세한 내용은 DAX 클라이언트 구성을 참조하세요. | 2025년 2월 17일 |

| | | |
|---|---|---|
| 대량 데이터 작업의 새로운 모범 사례 | DynamoDB에서 복잡한 데이터 모델 작업을 수행하는 새로운 모범 사례 주제를 게시했습니다. 자세한 내용은 DynamoDB에서 대량 데이터 작업을 사용하는 모범 사례를 참조하세요. | 2025년 1월 9일 |
| 이제 Amazon DynamoDB에서 구성 가능한 시점 복구(PITR) 지원 | DynamoDB는 이제 PITR에 대해 구성 가능한 복구 기간을 지원합니다. 각 테이블의 PITR 기간을 1~35일로 설정할 수 있습니다. 자세한 내용은 DynamoDB의 시점 백업을 참조하세요. | 2025년 1월 7일 |
| 이제 Amazon DynamoDB에서 구성 가능한 시점 복구(PITR) 지원 | DynamoDB는 이제 PITR에 대해 구성 가능한 복구 기간을 지원합니다. 각 테이블의 PITR 기간을 1~35일로 설정할 수 있습니다. 자세한 내용은 DynamoDB의 시점 백업을 참조하세요. | 2025년 1월 7일 |
| Amazon Managed Streaming for Apache Kafka를 Amazon DynamoDB와 통합하는 방법에 대한 새로운 주제 게시 | Amazon Managed Streaming for Apache Kafka가 Apache Kafka 주제에서 데이터를 읽고 DynamoDB에 저장하여 Amazon DynamoDB와 통합하는 방법을 알아봅니다. 자세한 내용은 Integrating DynamoDB with Amazon Managed Streaming for Apache Kafka를 참조하세요. | 2024년 12월 26일 |

| | | |
|---|---|---|
| Amazon Managed Streaming for Apache Kafka를 Amazon DynamoDB와 통합하는 방법에 대한 새로운 주제 게시 | Amazon Managed Streaming for Apache Kafka가 Apache Kafka 주제에서 데이터를 읽고 DynamoDB에 저장하여 Amazon DynamoDB와 통합하는 방법을 알아봅니다. 자세한 내용은 Integrating DynamoDB with Amazon Managed Streaming for Apache Kafka를 참조하세요. | 2024년 12월 26일 |
| DynamoDB, SageMaker AI Lakehouse와의 제로 ETL 통합 지원 도입 | DynamoDB는 Amazon DynamoDB에서 데이터를 추출하여 고객의 데이터 레이크로 로드하는 작업을 자동화하는 제로 ETL 통합을 도입합니다. 자세한 내용은 SageMaker AI Lakehouse와 DynamoDB 제로 ETL 통합을 참조하세요. | 2024년 12월 3일 |
| DynamoDB, Amazon SageMaker Lakehouse와의 제로 ETL 통합 지원 도입 | DynamoDB는 Amazon DynamoDB에서 데이터를 추출하여 고객의 데이터 레이크로 로드하는 작업을 자동화하는 제로 ETL 통합을 도입합니다. 자세한 내용은 DynamoDB zero-ETL integration with Amazon SageMaker Lakehouse를 참조하세요. | 2024년 12월 3일 |

| | | |
|---|---|---|
| DynamoDB 글로벌 테이블, 이제 다중 리전 강력한 일관성 지원 | 다중 리전 강력한 일관성을 통해 목표 복구 시점(RPO)이 0인 가용성이 높은 다중 리전 애플리케이션을 구축하여 가장 높은 수준의 복원력을 달성할 수 있습니다. 자세한 내용은 Global tables with multi-Region strong consistency를 참조하세요. | 2024년 12월 3일 |
| DynamoDB 글로벌 테이블, 이제 다중 리전 강력한 일관성 지원 | 다중 리전 강력한 일관성을 통해 목표 복구 시점(RPO)이 0인 가용성이 높은 다중 리전 애플리케이션을 구축하여 가장 높은 수준의 복원력을 달성할 수 있습니다. 자세한 내용은 Global tables with multi-Region strong consistency를 참조하세요. | 2024년 12월 3일 |
| DynamoDB 관리형 정책 업데이트 | `AmazonDynamoDBReadOnlyAccess` 관리형 정책에 두 개의 새로운 권한(`dynamodb:GetAbacStatus` 및 `dynamodb:UpdateAbacStatus`)을 추가했습니다. 이러한 권한을 통해 ABAC 상태를 보고 현재 리전의 AWS 계정에 ABAC를 활성화할 수 있습니다. 자세한 내용은 AWS 관리형 정책: AmazonDynamoDBReadOnlyAccess를 참조하세요. | 2024년 11월 18일 |

| | | |
|---|---|---|
| DynamoDB, 속성 기반 액세스 제어(ABAC) 지원 도입 | ABAC는 사용자, 역할 및 AWS 리소스에 연결되는 태그를 기반으로 액세스 권한을 정의하는 권한 부여 전략입니다. ABAC는 AWS Identity and Access Management(IAM) 정책 또는 기타 정책의 태그 기반 조건을 사용하여 IAM 위탁자의 태그가 테이블의 태그와 일치할 때 테이블 또는 인덱스에 대한 특정 작업을 허용하거나 거부합니다. 자세한 내용은 Using attribute-based access control with DynamoDB를 참조하세요. | 2024년 11월 18일 |
| DynamoDB, 온디맨드 및 프로비저닝된 테이블에 대해 웜 처리량 도입 | DynamoDB가 이제 웜 처리량을 지원합니다. 웜 처리량은 DynamoDB 테이블이 즉시 지원할 수 있는 읽기 및 쓰기 작업 수와 DynamoDB 테이블을 사전 워밍하는 기능에 대한 가시성을 제공합니다. 자세한 내용은 DynamoDB 웜 처리량을 참조하세요. | 2024년 11월 13일 |
| Apache Flink를 사용하여 Amazon DynamoDB Streams 레코드를 사용하는 기능 | 이제 Apache Flink에서 Amazon DynamoDB Streams 레코드를 사용하고 Amazon Managed Service for Apache Flink를 활용하여 엔드 투 엔드 스트림 처리 애플리케이션을 빠르게 빌드하고 관리할 수 있습니다. 자세한 내용은 DynamoDB Streams and Apache Flink를 참조하세요. | 2024년 11월 12일 |

| | | |
|---|---|---|
| 글로벌 테이블 및 백업에 대한 새 청구 주제 게시 | 글로벌 테이블 및 백업의 청구와 관련된 두 가지 새로운 주제를 게시했습니다. 자세한 내용은 글로벌 테이블에 대한 Amazon DynamoDB 요금 청구 이해 및 백업에 대한 Amazon DynamoDB 요금 청구 이해를 참조하세요. | 2024년 10월 16일 |
| Amazon Redshift와 Amazon DynamoDB의 제로 ETL 통합 | Amazon Redshift와 Amazon DynamoDB의 제로 ETL 통합은 DynamoDB에서 Amazon Redshift로 복제하는 완전 관리형 노코드 ETL 파이프라인을 제공합니다. 자세한 내용은 Amazon Redshift와 Amazon DynamoDB의 제로 ETL 통합을 참조하세요. | 2024년 10월 15일 |
| DynamoDB에서 생성형 AI 사용과 관련된 주제를 추가하여 설명서만 업데이트 | DynamoDB의 생성형 AI 사용 사례 예시를 포함하여 DynamoDB에서 생성형 AI 사용에 대한 정보를 제공하는 새로운 주제를 게시했습니다. 자세한 내용은 DynamoDB에서 생성형 AI 사용을 참조하세요. | 2024년 10월 11일 |
| 이제 SDK에서 AWS 계정 기반 엔드포인트 지원 | 계정 기반 엔드포인트와 SDK 클라이언트 관련 `ACCOUNT_ID_ENDPOINT_MODE` 설정에 대한 설명서가 추가되었습니다. 자세한 내용은 SDK에서 AWS 계정 기반 엔드포인트 지원을 참조하세요. | 2024년 9월 3일 |

| | | |
|---|---|---|
| 시작하기 환경 재설계 | 정보를 통합하고 더 빠르게 온보딩을 시작할 수 있도록 시작하기 환경을 재설계했습니다. 자세한 내용은 DynamoDB 시작하기를 참조하세요. | 2024년 8월 1일 |
| 스페인과 스웨덴의 새 리전으로 DAX 확장 | 이제 스페인 및 스웨덴 리전에서 DAX를 사용할 수 있습니다. 자세한 내용은 DAX 클러스터 구성 요소를 참조하세요. | 2024년 7월 30일 |
| DynamoDB 백업 및 복원 문서 재구성 및 통합 | DynamoDB 개발자 안내서의 백업 및 복원 관련 내용이 새롭게 편성되었습니다. 자세한 내용은 DynamoDB에 대한 백업 및 복원을 참조하세요. | 2024년 7월 2일 |
| What is Amazon DynamoDB? 주제 재작성 | What is Amazon DynamoDB? 주제의 수정 및 업데이트된 버전을 게시했습니다. 자세한 내용은 What is Amazon DynamoDB?를 참조하세요. | 2024년 6월 21일 |
| DynamoDB Streams와 EventBridge의 통합 | DynamoDB Streams와 EventBridge의 통합에 대한 새 주제를 게시했습니다. 자세한 내용은 Integrating with EventBridge를 참조하세요. | 2024년 6월 21일 |

| | | |
|---|---|---|
| DAX 권장 가이드 | DynamoDB Accelerator를 효과적으로 사용하기 위한 포괄적인 인사이트를 제공하는 새로운 모범 사례 주제를 게시했습니다. 이 주제에서는 성능 최적화, 비용 관리 및 운영 모범 사례를 다룹니다. 자세한 내용은 DAX prescriptive guidance를 참조하세요. | 2024년 6월 3일 |
| DynamoDB 테이블을 한 계정에서 다른 계정으로 마이그레이션 | DynamoDB 테이블을 한 계정에서 다른 계정으로 마이그레이션하는 방법에 대한 새로운 주제를 추가했습니다. 자세한 내용은 Migrating a DynamoDB table from one account to another을 참조하세요. | 2024년 5월 29일 |
| DynamoDB 모니터링 및 로깅 설명서 재구성 및 통합 | DynamoDB 모니터링 및 로깅의 새로운 구조로 지표, 로깅 작업 및 Contributor Insights와 관련된 간결한 세 개의 장이 포함되어 있습니다. | 2024년 5월 3일 |

| | | |
|---|---|---|
| DynamoDB 용량 모드 설명서 재구성 및 통합 | DynamoDB 가이드에 이제 DynamoDB 용량 모드(온디맨드 및 프로비저닝)에 대한 모든 정보가 포함된 새 장이 있습니다. 이번 업데이트에서는 읽기/쓰기 용량 모드 변경 시 고려 사항 항목이 모범 사례 장으로 이동되었습니다. 이 항목은 이제 용량 모드 전환 시 고려 사항으로 이름이 변경되었으며 용량 모드 간 전환 시 모범 사례에 대한 자세한 정보를 포함하고 있습니다. 또한 이 가이드에는 이제 DynamoDB 읽기 및 쓰기, 읽기 및 쓰기 작업의 용량 단위 소비에 대한 모든 정보가 포함된 새 장이 있습니다. 자세한 내용은 DynamoDB 처리량 용량, 용량 모드 전환 시 고려 사항, DynamoDB 읽기 및 쓰기를 참조하세요. | 2024년 5월 1일 |
| 최대 온디맨드 요청 수 | 이제 개별 테이블, 인덱스 또는 둘 다에서 수행할 수 있는 온디맨드 요청의 최대 수를 지정할 수 있습니다. 최대 온디맨드 처리량을 지정하면 테이블 수준의 사용량과 비용을 제한하고 리소스 소비가 의도치 않게 급증하는 것을 방지할 수 있습니다. 자세한 내용은 온디맨드 테이블의 최대 처리량을 참조하세요. | 2024년 5월 1일 |

| | | |
|---|---|---|
| NoSQL Workbench 작업 빌더 개선 | NoSQL Workbench에는 이제 다크 모드에 대한 기본 지원이 포함됩니다. 작업 빌더의 테이블 및 항목 작업이 개선되었습니다. 항목 결과 및 작업 빌더 요청 정보가 JSON 형식으로 제공됩니다. 자세한 내용은 NoSQL Workbench 작업 빌더 섹션을 참조하세요. | 2024년 4월 24일 |
| Amazon DynamoDB 리소스에 대한 리소스 기반 정책 | DynamoDB는 이제 테이블, 인덱스 및 스트림에 리소스 기반 정책을 지원합니다. 리소스 기반 정책을 사용하면 각 리소스에 액세스할 수 있는 사람과 각 리소스에서 수행할 수 있는 작업을 지정하여 액세스 권한을 정의할 수 있습니다. 자세한 내용은 Using resource-based policies for DynamoDB를 참조하세요. | 2024년 3월 20일 |
| DynamoDB 관리형 정책 업데이트 | `AmazonDynamoDBReadOnlyAccess` 관리형 정책에 새로운 권한인 `dynamodb:GetResourcePolicy` 를 추가했습니다. 이 권한은 DynamoDB 리소스에 연결된 리소스 기반 정책을 읽을 수 있는 액세스를 제공합니다. 자세한 내용은 AWS 관리형 정책: AmazonDynamoDBReadOnlyAccess를 참조하세요. | 2024년 3월 20일 |

| | | |
|---|---|---|
| AWS PrivateLink for Amazon DynamoDB | Amazon DynamoDB는 이제 AWS PrivateLink를 지원합니다. AWS PrivateLink를 사용하면 인터페이스 VPC 엔드포인트와 프라이빗 IP 주소를 사용하여 Virtual Private Cloud(VPC), DynamoDB 및 온프레미스 데이터 센터 간의 프라이빗 네트워크 연결을 단순화할 수 있습니다. 자세한 내용은 AWS PrivateLink for DynamoDB를 참조하세요. | 2024년 3월 19일 |
| JavaScript를 사용한 프로그래밍 안내서 | Amazon DynamoDB는 AWS SDK for JavaScript에 대한 프로그래밍 가이드를 제공합니다. AWS SDK for JavaScript, 추상화 계층, 연결 구성, 오류 처리, 재시도 정책 정의, 연결 유지 관리 등에 대해 알아보세요. 자세한 내용은 JavaScript를 사용한 프로그래밍을 참조하세요. | 2024년 3월 6일 |
| AWS SDK for Java 2.x를 사용한 프로그래밍 안내서 | 상위 수준, 하위 수준 및 문서 인터페이스, HTTP 클라이언트 및 해당 구성, 오류 처리에 대해 자세히 설명하고 SDK for Java 2.x를 사용할 때 고려해야 하는 가장 일반적인 구성 설정을 설명하는 새 프로그래밍 안내서를 만들었습니다. 자세한 내용은 AWS SDK for Java 2.x를 사용한 Amazon DynamoDB 프로그래밍을 참조하세요. | 2024년 3월 5일 |

| | | |
|---|---|---|
| NoSQL Workbench로 테이블 복제 | 개발자가 NoSQL Workbench를 사용하여 개발 환경 및 리전(DynamoDB 로컬 및 DynamoDB 웹) 간에 테이블을 복사하거나 복제할 수 있습니다. 자세한 내용은 Cloning tables with NoSQL Workbench를 참조하세요. | 2024년 2월 26일 |
| Python을 사용한 프로그래밍 안내서 | 상위 수준 및 하위 수준 라이브러리에 대해 자세히 설명하고 Python SDK를 사용할 때 고려해야 하는 가장 일반적인 구성 설정을 설명하는 새 안내서를 만들었습니다. 자세한 내용은 Python을 사용한 프로그래밍을 참조하세요. | 2024년 1월 5일 |
| Time to live(TTL) 주제 재작성 | 안내서의 TTL 섹션을 완전히 다시 작성했습니다. 새 안내서는 바로 사용할 수 있는 코드 스니펫을 제공하여 TTL을 시작하는 데 도움이 됩니다. 현재 제공되는 코드 스니펫은 Python과 Javascript용입니다. 자세한 내용은 TTL을 참조하세요. | 2023년 12월 20일 |
| AWS 결제 및 사용량 보고서 이해의 모범 사례 | DynamoDB의 다양한 사용량 유형과 해당 사용량 유형에 대한 요금을 설명하는 새 섹션을 추가했습니다. 자세한 내용은 결제 및 사용량 보고서를 참조하세요. | 2023년 12월 15일 |

| | | |
|---|---|---|
| Amazon OpenSearch Service와 Amazon DynamoDB의 제로 ETL 통합 | Amazon DynamoDB는 이제 Amazon OpenSearch Service와의 제로 ETL 통합을 지원합니다. 이 통합을 통해 사용자 지정 코드나 인프라 없이 DynamoDB 데이터를 자동으로 복제 및 변환하여 검색할 수 있습니다. 자세한 내용은 DynamoDB zero-ETL integration with Amazon OpenSearch Service를 참조하세요. | 2023년 11월 28일 |
| 관계형 데이터베이스에서 DynamoDB로 마이그레이션 | 사용자가 관계형 데이터베이스에서 DynamoDB로 마이그레이션하는 방법을 이해하는 데 도움이 되도록 마이그레이션 안내서를 만들었습니다. | 2023년 11월 27일 |
| NoSQL Workbench로 샘플 데이터 생성 | Amazon DynamoDB용 NoSQL Workbench는 이제 샘플 데이터 모델 템플릿에서 직접 데이터 모델을 생성할 수 있도록 지원하므로 워크로드에 맞는 데이터 스키마를 설계하는 데 도움이 됩니다. 이 기능을 사용하면 DynamoDB에서 애플리케이션을 구축할 때 NoSQL 데이터 모델링 모범 사례를 익힐 수 있습니다. | 2023년 9월 28일 |

| | | |
|---|---|---|
| S3로 증분 내보내기 | 이제 삽입, 업데이트 또는 삭제된 데이터를 조금씩 내보낼 수 있습니다. 증분 내보내기를 사용하면 AWS Management Console, API 호출 또는 AWS 명령줄 인터페이스에서 몇 번의 클릭만으로 몇 메가바이트에서 테라바이트에 이르는 변경된 데이터를 내보낼 수 있습니다. | 2023년 9월 26일 |
| DynamoDB를 위한 데이터 모델링 | 이제 특정 사용 사례, 액세스 패턴 및 이러한 액세스 패턴을 실현하기 위한 단계별 지침에 초점을 맞춘 DynamoDB 예제를 통해 데이터 모델링에 대해 자세히 알아볼 수 있습니다. | 2023년 7월 14일 |
| 문제 해결 섹션 | 이제 DynamoDB 테이블에서 발생할 수 있는 지연 및 제한 문제에 대한 문제 해결 콘텐츠를 찾을 수 있습니다. | 2023년 3월 13일 |
| Amazon DynamoDB에 대한 삭제 보호 기능 | 이제 모든 AWS 리전의 Amazon DynamoDB 테이블에 대해 삭제 보호 기능을 사용할 수 있습니다. 이제 DynamoDB를 사용하면 정규 테이블 관리 작업을 수행할 때 실수로 테이블이 삭제되지 않도록 보호할 수 있습니다. | 2023년 3월 8일 |

| | | |
|---|---|---|
| 글로벌 테이블에서 KDSD에 대한 AWS CloudFormation 지원 | 이제 DynamoDB용 Amazon Kinesis Data Streams가 AWS CloudFormation을 지원합니다. 즉, CloudFormation 템플릿을 사용하여 DynamoDB 테이블에서 Amazon Kinesis Data Streams로 스트리밍을 활성화할 수 있습니다. | 2023년 2월 15일 |
| DynamoDB 로컬에서 트랜잭션당 100개의 작업 지원 | 이제 DynamoDB 로컬에서 단일 트랜잭션에서 최대 100개의 작업을 수행할 수 있습니다. | 2023년 2월 9일 |
| DynamoDB Well-Architected 렌즈를 사용하여 DynamoDB 워크로드 최적화 | 이제 효율적으로 구조화된 DynamoDB 워크로드를 설계하기 위한 설계 원칙 및 지침 모음인 Amazon DynamoDB Well-Architected 렌즈를 사용할 수 있습니다. | 2023년 2월 3일 |
| PartiQL GovGloud 가용성 | Amazon DynamoDB의 SQL 호환 쿼리 언어인 PartiQL이 이제 AWS GovCloud(미국 동부) 및 AWS GovCloud(미국 서부)에서 지원됩니다. | 2022년 12월 21일 |
| NoSQL Workbench 및 DynamoDB 로컬에 대한 단일 설치 제품군 | DynamoDB용 NoSQL Workbench에는 이제 가이드형 DynamoDB 로컬 설치 프로세스가 포함되어 DynamoDB 로컬 개발 환경 설정이 간소화됩니다. | 2022년 12월 6일 |

| | | |
|---|---|---|
| S3에서 대량 가져오기 | 이제 Amazon DynamoDB에서 Amazon S3에서 대량 데이터 가져오기를 지원하여 데이터를 새 DynamoDB 테이블로 쉽게 마이그레이션하고 로드할 수 있습니다. | 2022년 8월 18일 |
| Service Quotas와의 향상된 통합 | Service Quotas를 통해 이제 계정 및 테이블 할당량을 사전에 관리할 수 있습니다. 현재 값을 보고 할당량 사용률이 구성 가능한 임계값을 초과하는 경우에 대한 경보를 설정하는 등의 작업을 수행할 수 있습니다. | 2022년 6월 15일 |
| NoSQL 워크벤치에 테이블 및 GSI 지원 추가 | 이제 테이블 및 글로벌 보조 인덱스(GSI) 컨트롤 플레인 작업(예: CreateTable, UpdateTable, DeleteTable)에 NoSQL Workbench를 사용할 수 있습니다. | 2022년 6월 2일 |
| Standard-Infrequent Access 테이블 클래스를 이제 중국에서 사용 가능 | Amazon DynamoDB Standard-Infrequent Access 테이블 클래스를 이제 중국 리전에서 사용할 수 있습니다. 자주 액세스하지 않는 데이터가 저장된 테이블에 이 새로운 테이블 클래스를 사용하여 DynamoDB 비용을 최대 60%까지 줄이세요. | 2022년 4월 18일 |
| 기본 서비스 할당량 및 테이블 관리 작업 수 증가 | DynamoDB에서 계정 및 리전당 테이블 수에 대한 기본 할당량이 256개에서 2,500개의 테이블로 증가하고, 동시 테이블 관리 작업 수가 50개에서 500개로 증가했습니다. | 2022년 3월 9일 |

| | | |
|---|---|---|
| DynamoDB용 PartiQL의 항목 제한(선택 사항) | DynamoDB에서 각 요청에 대한 선택적 파라미터로 DynamoDB용 PartiQL 작업에서 처리되는 항목 수를 제한할 수 있습니다. | 2022년 3월 8일 |
| 중국(베이징 및 닝샤) 리전에서 AWS Backup 통합 기능 사용 가능 | 이제 중국(베이징 및 닝샤) 리전에서 AWS Backup이 DynamoDB와 통합됩니다. AWS Backup의 향상된 백업 기능(예: 교차 계정 백업 및 교차 리전 백업)을 통해 규정 준수 및 비즈니스 연속성 요구 사항을 보다 쉽게 충족할 수 있습니다. | 2022년 1월 26일 |
| PartiQL API 호출을 통한 처리량 용량 정보 확인 | 쿼리 및 처리량 비용을 최적화하는 데 도움이 되도록 DynamoDB에서 PartiQL API 호출에 소비된 처리량 용량을 반환할 수 있습니다. | 2022년 1월 18일 |
| AWS Backup 통합 | AWS Backup의 향상된 백업 기능(예: 교차 계정 백업 및 교차 리전 백업)을 통해 이제 DynamoDB에서 규정 준수 및 비즈니스 연속성 요구 사항을 보다 쉽게 충족할 수 있습니다. | 2021년 11월 24일 |
| CSV로 NoSQL Workbench 데이터 세트 가져오기/내보내기 | 이제 Amazon DynamoDB용 NoSQL Workbench를 사용하여 샘플 데이터를 가져오고 자동으로 채워 데이터 모델을 구축하고 시각화할 수 있습니다. | 2021년 10월 11일 |

| | | |
|---|---|---|
| AWS CloudTrail을 사용하여 Amazon DynamoDB Streams 데이터 영역 활동 필터링 및 검색 | 이제 AWS CloudTrail로 Streams 데이터 영역 API 활동을 필터링함으로써 Amazon DynamoDB에서 감사 로깅을 보다 세부적으로 제어할 수 있습니다. | 2021년 9월 22일 |
| 콘솔 업데이트 | 이제 DynamoDB 콘솔을 기본 콘솔로 사용하여 데이터를 보다 쉽게 관리하고, 일반적인 작업을 간소화하며, 리소스와 기능에 더 빠르게 액세스할 수 있습니다. | 2021년 8월 25일 |
| 이제 Java 2.x용 DAX SDK 사용 가능 | 이제 Java 2.x용 DynamoDB Accelerator(DAX) SDK가 사용 가능하며 Java 2.x용 AWS SDK와 호환됩니다. 비차단 I/O를 비롯한 최신 기능을 활용할 수 있습니다. | 2021년 7월 29일 |
| 제어 영역 작업을 포함한 NoSQL Workbench 기능 업데이트 | 이제 Amazon DynamoDB용 NoSQL Workbench에서 자주 처리하는 작업을 더 쉽게 실행하여 테이블 데이터를 수정하고 액세스할 수 있습니다. | 2021년 7월 28일 |
| 이제 아시아 태평양 리전에서 DynamoDB 글로벌 테이블 사용 가능 | 이제 아시아 태평양(오사카) 리전에서 DynamoDB 글로벌 테이블을 사용할 수 있습니다. DynamoDB 테이블을 22개 중 선택한 AWS 리전 간에 자동으로 복제할 수 있습니다 | 2021년 7월 28일 |

| | | |
|---|---|---|
| 이제 중국에서 DAX 사용 가능 | 이제 Sinnet이 운영하는 중국(베이징) 리전에서 DynamoDB Accelerator(DAX)를 사용할 수 있습니다. | 2021년 7월 28일 |
| DAX 전송 중 데이터 암호화 | 이제 DynamoDB Accelerator(DAX)에서 애플리케이션과 DAX 클러스터 간에 그리고 DAX 클러스터 내의 노드 간에 데이터 전송 시 암호화를 지원합니다. | 2021년 7월 24일 |
| CloudFormation 및 CloudTrail 통합 | AWS CloudFormation과 통합되며 CloudFormation 데이터 영역 로깅을 통해 보안이 향상됩니다. | 2021년 6월 18일 |
| 이제 글로벌 테이블에 CloudFormation 지원 | 이제 Amazon DynamoDB 글로벌 테이블이 AWS CloudFormation을 지원하므로 CloudFormation 템플릿을 사용하여 글로벌 테이블을 생성하고 해당 설정을 관리할 수 있습니다. | 2021년 5월 14일 |
| Amazon DynamoDB Local에서 Java 2.x 지원 | 이제 다운로드 가능한 Amazon DynamoDB 버전인 DynamoDB Local과 함께 Java 2.x용 AWS SDK를 사용할 수 있습니다. DynamoDB Local을 사용하면 추가 비용 없이 로컬 개발 환경에서 실행되는 DynamoDB 버전을 사용하여 애플리케이션을 개발하고 테스트할 수 있습니다. | 2021년 5월 3일 |

| | | |
|---|---|---|
| 이제 NoSQL Workbench에서 AWS CloudFormation 지원 | 이제 Amazon DynamoDB용 NoSQL Workbench가 AWS CloudFormation을 지원하므로 CloudFormation 템플릿을 사용하여 DynamoDB 데이터 모델을 관리하고 수정할 수 있습니다. 또한 이제 NoSQL Workbench에서 테이블 용량 설정을 구성할 수 있습니다. | 2021년 4월 22일 |
| 이제 DynamoDB와 AWS Amplify의 기능 통합 | 이제 AWS Amplify가 단일 배포에서 여러 DynamoDB 글로벌 보조 인덱스 업데이트를 오케스트레이션합니다. | 2021년 4월 20일 |
| AWS CloudTrail로 Amazon DynamoDB Streams 데이터 영역 API 로깅 | 이제 AWS CloudTrail을 사용하여 Amazon DynamoDB Streams 데이터 영역 API 활동을 로그하고 DynamoDB 테이블의 항목 수준 변경 사항을 모니터링 및 조사할 수 있습니다. | 2021년 4월 20일 |
| 이제 Amazon DynamoDB용 Amazon Kinesis Data Streams에서 AWS CloudFormation 지원 | 이제 Amazon DynamoDB용 Amazon Kinesis Data Streams가 AWS CloudFormation을 지원하므로 CloudFormation 템플릿을 사용하여 DynamoDB 테이블에서 Amazon Kinesis 데이터 스트림으로 스트리밍을 사용 설정할 수 있습니다. DynamoDB 데이터 변경 사항을 Kinesis 데이터 스트림으로 스트리밍함으로써 Amazon Kinesis 서비스를 사용하여 고급 스트리밍 애플리케이션을 구축할 수 있습니다. | 2021년 4월 12일 |

| | | |
|---|---|---|
| 이제 Amazon Keyspaces에서 FIPS 140-2를 준수하는 엔드포인트 제공 | 이제 엄격히 규제되는 워크로드를 보다 쉽게 실행할 수 있도록 Amazon Keyspaces (Apache Cassandra용)에서 FIPS(Federal Information Processing Standards) 140-2를 준수하는 엔드포인트를 제공합니다. FIPS 140-2는 미국 및 캐나다 정부 표준으로서, 기밀 정보를 보호하는 암호화 모듈의 보안 요건을 규정하고 있습니다. | 2021년 4월 8일 |
| DAX용 Amazon EC2 T3 인스턴스 | 이제 DAX에서 기본 수준의 CPU 성능 외에 필요할 경우 기본 이상으로 높일 수 있는 기능을 제공하는 Amazon EC2 T3 인스턴스 유형을 지원합니다. | 2021년 2월 15일 |
| Amazon DynamoDB용 NoSQL Workbench의 PartiQL 지원 | 이제 DynamoDB용 NoSQL Workbench를 사용하여 DynamoDB를 위한 PartiQL 문을 빌드할 수 있습니다. | 2020년 12월 4일 |
| DynamoDB용 PartiQL | 이제 DynamoDB용 PartiQL(SQL 호환 쿼리 언어)을 사용하여 DynamoDB 테이블과 상호 작용하고 AWS Management Console,AWS Command Line Interface 및 PartiQL용 DynamoDB API를 사용하여 임시 쿼리를 실행할 수 있습니다. | 2020년 11월 23일 |

| | | |
|---|---|---|
| Amazon DynamoDB용 Amazon Kinesis Data Streams | 이제 Amazon DynamoDB용 Amazon Kinesis Data Streams를 DynamoDB 테이블과 함께 사용하여 항목 수준 변경 사항을 캡처하고 Kinesis 데이터 스트림에 복제할 수 있습니다. | 2020년 11월 23일 |
| DynamoDB 테이블 내보내기 | 이제 DynamoDB 테이블을 Amazon S3로 내보내서 Athena, AWS Glue, Lake Formation 등의 서비스로 데이터에 대한 분석과 복잡한 쿼리를 수행할 수 있습니다. | 2020년 11월 9일 |
| 빈 값 지원 | DynamoDB는 이제 DynamoDB 테이블에서 키가 아닌 문자열 및 이진 속성에 대해 빈 값을 지원합니다. 빈 값 지원 덕분에 광범위한 사용 사례에서 유연하게 속성을 사용할 수 있으며 속성을 DynamoDB로 보내기 전에 변환하지 않아도 됩니다. 목록,맵, 집합 데이터 형식에서도 빈 문자열 및 이진 값을 사용할 수 있습니다. | 2020년 5월 18일 |
| Amazon DynamoDB용 NoSQL Workbench의 Linux 지원 | Amazon DynamoDB용 NoSQL Workbench는 이제 Linux(Ubuntu, Fedora 및 Debian)에서 지원됩니다. | 2020년 5월 4일 |

| | | |
|---|---|---|
| DynamoDB용 CloudWatch Contributor Insights - 정식 버전 | DynamoDB용 CloudWatch Contributor Insights 정식 버전을 사용할 수 있습니다. DynamoDB용 CloudWatch Contributor Insights는 DynamoDB 테이블의 트래픽 추세를 한 눈에 확인하고 테이블에서 가장 자주 액세스되는 키(바로 가기 키라고도 함)를 식별할 수 있는 진단 도구입니다. | 2020년 4월 2일 |
| 글로벌 테이블 업그레이드 | 이제 DynamoDB 콘솔에서 몇 번의 클릭으로 전역 테이블을 버전 2017.11.29에서 최신 버전의 전역 테이블(2019.11.21)로 업데이트할 수 있습니다. 전역 테이블의 버전을 업그레이드하면 테이블 재작성 없이도 기존 테이블을 추가 AWS 리전으로 확장하여 DynamoDB 테이블의 가용성을 쉽게 높일 수 있습니다. | 2020년 3월 16일 |
| Amazon DynamoDB용 NoSQL Workbench – 정식 버전 | Amazon DynamoDB용 NoSQL Workbench가 정식 버전입니다. NoSQL Workbench를 사용하여 DynamoDB 테이블을 설계하고, 생성하고, 쿼리하고, 관리합니다. | 2020년 3월 2일 |
| DAX 캐시 클러스터 지표 | DAX는 DAX 클러스터의 성능을 더 잘 파악할 수 있는 새로운 CloudWatch 지표를 지원합니다. | 2020년 2월 6일 |

| | | |
|---|---|---|
| DynamoDB용 CloudWatch Contributor Insights - 미리 보기 | DynamoDB용 CloudWatch Contributor Insights는 DynamoDB 테이블의 트래픽 추세를 한 눈에 확인하고 테이블에서 가장 자주 액세스되는 키(바로 가기 키라고도 함)를 식별할 수 있는 진단 도구입니다. | 2019년 11월 26일 |
| 조정 용량의 불균형 워크로드 지원 | Amazon DynamoDB 조정 용량에서는 이제 자주 액세스하는 항목을 자동으로 격리하여 불균형 워크로드를 보다 효율적으로 처리합니다. 애플리케이션에서 하나 이상의 항목으로 트래픽을 불균형적으로 많이 이동하는 경우 DynamoDB가 파티션 균형을 재조정하여 자주 액세스하는 항목이 동일한 파티션에 상주하지 않도록 합니다. | 2019년 11월 26일 |
| 고객 관리형 키에 대한 지원 | 이제 DynamoDB는 고객 관리형 키를 지원하므로 DynamoDB 데이터의 보안을 암호화하고 관리하는 방법을 완전하게 제어할 수 있습니다. | 2019년 11월 25일 |
| NoSQL Workbench에서 DynamoDB Local(다운로드 가능 버전) 지원 | NoSQL Workbench는 이제 DynamoDB Local(다운로드 가능 버전)에 대한 연결을 지원하므로 DynamoDB 테이블을 설계, 생성, 쿼리 및 관리할 수 있습니다. | 2019년 11월 8일 |

| | | |
|---|---|---|
| NoSQL Workbench - 미리 보기 | 이것은 DynamoDB용 NoSQL Workbench의 최초 릴리스입니다. NoSQL Workbench를 사용하여 DynamoDB 테이블을 설계하고, 생성하고, 쿼리하고, 관리합니다. 자세한 내용은 Amazon DynamoDB용 NoSQL Workbench(평가판)를 참조하세요. | 2019년 9월 16일 |
| DAX에서 Python 및 .NET를 사용한 트랜잭션 작업에 대한 지원 추가 | DAX는 Go, Java, .NET, Node.js 및 Python으로 작성된 애플리케이션에 대해 `TransactWriteItems` 및 `TransactGetItems` API를 지원합니다. 자세한 내용은 DAX로 인 메모리 가속화 섹션을 참조하세요. | 2019년 2월 14일 |
| Amazon DynamoDB Local(다운로드 가능 버전) 업데이트 | DynamoDB Local(다운로드 버전)은 이제 트랜잭션 API, 온디맨드 읽기/쓰기 용량, 읽기 및 쓰기 작업에 대한 용량 보고, 20개의 글로벌 보조 인덱스를 지원합니다. 자세한 내용은 DynamoDB 다운로드 버전과 DynamoDB 웹 서비스 간 차이 섹션을 참조하세요. | 2019년 2월 4일 |

| | | |
|---|---|---|
| Amazon DynamoDB On-Demand | DynamoDB on-demand는 용량 계획 없이 초당 수천 개의 요청을 처리할 수 있는 유연한 청구 옵션입니다. DynamoDB on-demand는 읽기 및 쓰기 요청에 대해 요청당 지불 가격을 제공하므로 사용하는 만큼에 대해서만 비용을 지불하면 됩니다. 자세한 내용은 DynamoDB 처리량 용량을 참조하세요. | 2018년 11월 28일 |
| Amazon DynamoDB Transactions | DynamoDB transactions을 사용하면 테이블 내부와 전체에서 여러 항목을 모두 변경하거나 전혀 변경하지 않고 조정하여 DynamoDB에서 원자가, 일관성, 격리성 및 지속성(ACID)을 제공할 수 있습니다. 자세한 내용은 Amazon DynamoDB Transactions을 참조하세요. | 2018년 11월 27일 |
| Amazon DynamoDB에서 저장된 모든 고객 데이터 암호화 | 유휴 DynamoDB 암호화는 내구성이 뛰어난 미디어에 데이터가 자정될 때마다 기본 키, 로컬 및 글로벌 보조 인덱스, 스트림, 글로벌 테이블, 백업, DAX 클러스터 등을 비롯한 데이터를 암호화된 테이블에서 보호하여 추가 계층의 데이터 보호를 제공합니다. 자세한 내용은 미사용 Amazon DynamoDB 암호화를 참조하세요. | 2018년 11월 15일 |

| | | |
|---|---|---|
| 새로운 Docker 이미지로 Amazon DynamoDB Local을 보다 간편하게 사용 | 이제 보다 간편하게 DynamoDB의 다운로드 가능 버전인 Amazon DynamoDB Local을 활용해 새로운 DynamoDB Local 도커 이미지로 DynamoDB 애플리케이션을 개발 및 테스트할 수 있습니다. 자세한 내용은 DynamoDB(다운로드 버전) 및 도커를 참조하세요. | 2018년 8월 22일 |
| DynamoDB Accelerator(DAX), 저장 데이터 암호화 지원 추가 | DynamoDB Accelerator(DAX)는 이제 새로운 DAX 클러스터에 대한 저장 데이터 암호화를 지원하여 엄격한 규정 준수 및 규제 요구 사항이 적용되는 보안이 중요한 애플리케이션에서 Amazon DynamoDB 테이블 읽기를 가속화하도록 도와줍니다. 자세한 내용은 유휴 DAX 암호화를 참조하세요. | 2018년 8월 9일 |
| DynamoDB PITR(특정 시점으로 복구)에서 삭제된 테이블을 복원하기 위한 지원 추가 | 특정 시점으로 복구가 활성화된 테이블을 삭제하면 시스템 백업이 자동으로 생성되며 35일 동안 유지됩니다(추가 비용 없음). 자세한 내용은 특정 시점으로 복구를 시작하기 전을 참조하세요. | 2018년 8월 7일 |
| RSS에서 현재 사용 가능한 업데이트 | 이제 RSS 피드(이 페이지의 왼쪽 상단 모서리 부분에 있음)를 구독하여 Amazon DynamoDB 개발자 안내서에 대한 업데이트 알림을 받을 수 있습니다. | 2018년 7월 3일 |

# 이전 업데이트

다음 표에는 2018년 7월 3일 이전에 DynamoDB 개발자 안내서에서 변경된 중요 사항이 나와 있습니다.

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| DAX에 대해 Go 지원 | 새로운 DynamoDB Accelerator(DAX) SDK for Go를 사용하여 Go 프로그래밍 언어로 작성된 애플리케이션에서 이제 Amazon DynamoDB 테이블에 대해 마이크로초 읽기 성능을 사용할 수 있습니다. 자세한 내용은 DAX SDK for Go 섹션을 참조하세요. | 2018년 6월 26일 |
| DynamoDB에서 SLA 발표 | DynamoDB에서 공공 가용성 SLA를 발표했습니다. 자세한 내용은 Amazon DynamoDB 서비스 수준 계약을 참조하세요. | 2018년 6월 19일 |
| DynamoDB 연속 백업과 특정 시점으로 복구(PITR) | 특정 시점으로 복구(PITR)를 사용하면 우발적인 쓰기 또는 삭제 작업으로부터 Amazon DynamoDB 테이블을 보호할 수 있습니다. 특정 시점으로 복구를 설정해 두면 온디맨드 백업의 생성, 유지 관리, 예약을 걱정할 필요가 없습니다. 테스트 스크립트가 프로덕션 DynamoDB 테이블에 우발적으로 데이터를 쓴 경우를 예로 들어 보겠습니다. 특정 시점으로 복구가 설정되어 있으면 최근 35일 중 원하는 시점으로 테이블을 복원할 수 있습니다. | 2018년 25월 4일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| | DynamoDB는 테이블의 증분식 백업을 관리합니다. 자세한 내용은 DynamoDB의 시점 백업 섹션을 참조하세요. | |
| DynamoDB의 저장 데이터 암호화 | 새로운 DynamoDB 테이블에 대해 사용할 수 있는 DynamoDB 저장 데이터 암호화 기능은 AWS Key Management Service에 저장된 AWS 관리형 암호화 키를 사용하여 Amazon DynamoDB 테이블의 애플리케이션 데이터를 보호할 수 있습니다. 자세한 내용은 DynamoDB 저장 데이터 암호화 섹션을 참조하세요. | 2018년 2월 8일 |
| DynamoDB 백업 및 복원 | 온 디맨드 백업을 통해 DynamoDB 테이블 데이터의 완벽한 백업본을 만들고 이를 보관해 두면 기업은 물론 정부의 규제 요건을 준수하는 데도 도움이 됩니다. 프로덕션 애플리케이션의 성능과 가용성에 아무런 영향을 주지 않고도 몇 메가바이트에서 수백 테라바이트에 이르는 데이터 테이블을 백업할 방법이 있습니다. 자세한 내용은 DynamoDB에 대한 백업 및 복원 섹션을 참조하세요. | 2017년 11월 29일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| DynamoDB 전역 테이블 | 전역 테이블은 DynamoDB의 국제적인 입지를 기반으로 구축되어 완벽하게 관리되는 다중 리전 다중 활성 데이터베이스를 제공하며, 이 데이터베이스는 대규모로 확장되는 전역 애플리케이션에 대해 신속한 로컬 읽기 및 쓰기 성능을 지원합니다. 전역 테이블은 선택된 AWS 리전 간에 Amazon DynamoDB 테이블을 자동으로 복제합니다. 자세한 내용은 글로벌 테이블 - 다중 활성, 다중 리전 복제 섹션을 참조하세요. | 2017년 11월 29일 |
| DAX에 대한 Node.js 지원 | Node.js 개발자는 Node.js용 DAX 클라이언트를 사용하여 DynamoDB Accelerator(DAX)를 활용할 수 있습니다. 자세한 내용은 DynamoDB Accelerator(DAX)를 통한 인 메모리 가속화 섹션을 참조하세요. | 2017년 10월 5일 |
| DynamoDB용 VPC 엔드포인트 | DynamoDB 엔드포인트는 Amazon VPC의 Amazon EC2 인스턴스가 퍼블릭 인터넷에 노출되지 않고 DynamoDB에 액세스할 수 있도록 합니다. VPC와 DynamoDB 간의 네트워크 트래픽은 Amazon 네트워크를 벗어나지 않습니다. 자세한 내용은 Amazon VPC 엔드포인트를 사용하여 DynamoDB에 액세스 섹션을 참조하세요. | 2017년 8월 16일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| DynamoDB의 Auto Scaling | DynamoDB Auto Scaling 기능을 사용하면 프로비저닝된 처리량 설정을 수동으로 정의하거나 조정할 필요가 없습니다. 대신 DynamoDB Auto Scaling은 실제 트래픽 패턴에 따라 읽기 및 쓰기 용량을 동적으로 조정합니다. 따라서 테이블 또는 글로벌 보조 인덱스에 따라 할당된 읽기 및 쓰기 용량을 늘려 병목 현상 없이 갑작스러운 트래픽 증가를 처리할 수 있습니다. 워크로드가 감소하면 DynamoDB Auto Scaling 기능이 프로비저닝된 용량을 줄입니다. 자세한 내용은 DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리 섹션을 참조하세요. | 2017년 6월 14일 |
| DynamoDB Accelerator(DAX) | DynamoDB Accelerator(DAX)는 DynamoDB를 위한 완전 관리형, 고가용성, 인 메모리 캐시로, 초당 요청 수가 몇 백만 개인 경우에도 몇 밀리초에서 몇 마이크로초까지 최대 10배의 성능을 제공합니다. 자세한 내용은 DynamoDB Accelerator(DAX)를 통한 인 메모리 가속화 섹션을 참조하세요. | 2017년 4월 19일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| DynamoDB는 이제 유지 시간(TTL)으로 자동 항목 만료를 지원합니다 | Amazon DynamoDB 유지 시간(TTL)을 사용하면 추가 비용 없이 테이블에서 만료된 항목을 자동으로 삭제할 수 있습니다. 자세한 내용은 DynamoDB에서 TTL(Time To Live) 사용 섹션을 참조하세요. | 2017년 2월 27일 |
| DynamoDB는 이제 비용 할당 태그를 지원합니다 | 이제 Amazon DynamoDB 테이블에 태그를 추가해 사용 분류를 개선하고 더욱 자세한 비용 보고를 할 수 있습니다. 자세한 내용은 DynamoDB의 리소스에 태그 및 레이블 추가 섹션을 참조하세요. | 2017년 1월 19일 |
| 새로운 DynamoDB `DescribeLimits` API | `DescribeLimits` API는 한 리전의 AWS 계정에 대해 현재 프로비저닝된 용량 제한을 반환하는데, 리전 전체를 기준으로도 반환하고, 생성할 DynamoDB 테이블 하나를 기준으로도 반환합니다. 이에 따라 현재 계정 수준 제한 값을 알 수 있으므로 현재 사용하고 있는 할당된 용량과 이를 비교해볼 수 있으며, 제한에 도달하기 전에 이를 늘릴 수 있는 시간적 여유를 가질 수 있습니다. 자세한 내용은 Amazon DynamoDB API 참조에서 Amazon DynamoDB의 할당량 및 DescribeLimits를 참조하세요. | 2016년 3월 1일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| DynamoDB 콘솔 업데이트 및 기본 키 속성 새 용어 | DynamoDB 관리 콘솔이 보다 직관적이고 사용하기 쉽게 다시 설계되었습니다. 이 업데이트의 일환으로 기본 키 속성에 새 용어를 도입했습니다.<br>• 파티션 키 - 해시 속성이라고도 합니다.<br>• 정렬 키 - 범위 속성이라고도 합니다.<br><br>이름만 변경했을 뿐 기능은 동일합니다.<br><br>테이블 또는 보조 인덱스를 생성할 때 단순 기본 키(파티션 키만) 또는 복합 기본 키(파티션 키 및 정렬 키)를 선택할 수 있습니다. DynamoDB 설명서도 이러한 변경 사항을 반영하여 업데이트되었습니다. | 2015년 12월 11일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| Titan용 Amazon DynamoDB 스토리지 백엔드 | Titan용 DynamoDB 스토리지 백엔드는 Amazon DynamoDB 기반으로 구현된 Titan 그래프 데이터베이스용 스토리지 백엔드입니다. Titan용 DynamoDB 스토리지 백엔드를 사용하는 경우 데이터는 Amazon의 고가용성 데이터 센터 전체에 걸쳐 실행되는 DynamoDB 보안의 적용을 받게 됩니다. 플러그인은 Titan 버전 0.4.4(주로 기존 애플리케이션과 호환성을 위해 사용) 및 Titan 버전 0.5.4(새 애플리케이션용으로 권장)가 있습니다. 다른 Titan용 스토리지 백엔드와 마찬가지로, 이 플러그인은 Blueprints API, Gremlin 셸을 비롯해 Tinkerpop 스택(버전 2.4 및 2.5)을 지원합니다. | 2015년 8월 20일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| DynamoDB Streams 교차 리전 복제 및 강력히 일관된 읽기를 이용한 스캔 | DynamoDB Streams는 어떤 DynamoDB 테이블이든 시간 순서에 따라 항목 변경 사항이 있으면 이 정보를 수집하여 최대 24시간 동안 로그에 저장합니다. 로그와 데이터 항목은 변경 전후 거의 실시간으로 나타나므로 애플리케이션에서 이러한 로그와 데이터에 액세스할 수 있습니다. 자세한 내용은 DynamoDB Streams에 대한 변경 데이터 캡처 및 DynamoDB Streams API 참조를 참조하세요.<br><br>DynamoDB 교차 리전 복제는 여러 AWS 리전 간에 동일한 DynamoDB 테이블 사본을 거의 실시간으로 유지 관리하기 위한 클라이언트 측 솔루션입니다. 교차 리전 복제를 사용하여 DynamoDB 테이블을 백업하거나, 지리적으로 분산된 사용자가 오래 기다리지 않고 데이터에 액세스할 수 있습니다.<br><br>DynamoDB `Scan` 작업은 기본적으로 최종적으로 일관된 읽기를 사용합니다. 하지만 `ConsistentRead` 파라미터를 true로 설정하면 strongly consistent read도 사용할 수 있습니다. 자세한 내용은 Amazon DynamoDB API 참조 | 2015년 7월 16일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| | 에서 스캔에 대한 읽기 정합성 및 Scan을 참조하세요. | |
| AWS CloudTrail의 Amazon DynamoDB 지원 | 이제 DynamoDB가 CloudTrail과 통합됩니다. CloudTrail은 DynamoDB 콘솔 또는 DynamoDB API에서 API 호출을 캡처한 후 로그 파일로 이를 추적합니다. 자세한 내용은 AWS CloudTrail을 사용하여 DynamoDB 작업 로깅 및 AWS CloudTrail 사용 설명서를 참조하세요. | 2015년 5월 28일 |
| 쿼리 표현식에 대한 지원 개선 | 이번 릴리스에서는 새로운 `KeyConditionExpression` 파라미터가 `Query` API에 추가되었습니다. `Query`는 기본 키 값을 사용해 테이블이나 인덱스의 항목을 읽어옵니다. `KeyConditionExpression` 파라미터는 이러한 기본 키의 이름을 식별하는 문자열이자 키 값에 적용되는 조건입니다. 그러면 `Query`가 이 표현식을 만족하는 항목만 가져옵니다. `KeyConditionExpression` 구문은 DynamoDB의 다른 표현식 파라미터와 비슷하여 표현식의 이름과 값에 대해 치환 변수를 정의할 수 있습니다. 자세한 내용은 DynamoDB에서 테이블 쿼리 섹션을 참조하세요. | 2015년 4월 27일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 조건부 쓰기를 위한 새로운 비교 함수 | DynamoDB에서 `ConditionExpression` 파라미터는 `PutItem`, `UpdateItem` 또는 `DeleteItem` 작업의 성공 여부를 결정짓습니다. 즉, 이 조건이 true로 평가되는 경우에 한해 항목이 작성됩니다. 이번 릴리스에서는 새로운 함수로 `attribute_type` 와 `size`, 2개가 추가되어 `ConditionExpression` 과 함께 사용할 수 있습니다. 이 두 함수는 테이블 속성의 데이터 형식 또는 크기에 따라 조건부 쓰기가 가능합니다. 자세한 내용은 DynamoDB 조건 표현식 CLI 예제 섹션을 참조하세요. | 2015년 4월 27일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 보조 인덱스를 위한 스캔 API | DynamoDB에서 Scan 작업은 테이블 항목을 모두 읽어와서 사용자 정의 필터링 기준을 적용한 다음 선택된 데이터 항목을 애플리케이션에 반환합니다. 이러한 기능이 현재 보조 인덱스에도 똑같이 적용됩니다. 로컬 보조 인덱스 또는 글로벌 보조 인덱스를 스캔하려면 인덱스 이름과 상위 테이블 이름을 지정합니다. 기본적으로 인덱스 Scan은 인덱스의 데이터를 모두 반환하지만 필터 표현식을 사용하면 애플리케이션에 반환되는 결과 범위를 좁힐 수 있습니다. 자세한 내용은 DynamoDB에서 테이블 스캔 섹션을 참조하세요. | 2015년 2월 10일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 글로벌 보조 인덱스의 온라인 작업 | 온라인 인덱싱은 기존 테이블에 글로벌 보조 인덱스를 추가하거나 삭제할 수 있는 작업입니다. 특히 온라인 인덱싱 작업에서는 테이블 생성 시 테이블 인덱스를 모두 정의할 필요가 없습니다. 오히려 언제든지 새로운 인덱스를 추가할 수 있습니다. 마찬가지로 인덱스가 더 이상 필요 없다면 언제든지 삭제할 수 있습니다. 온라인 인덱싱 작업은 무중단(non-blocking) 방식으로 인덱스를 추가하거나 삭제하는 도중에도 테이블에 대한 읽기 및 쓰기 작업이 가능합니다. 자세한 내용은 DynamoDB에서 글로벌 보조 인덱스 관리 섹션을 참조하세요. | 2015년 1월 27일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| JSON을 이용한 문서 모델 지원 | DynamoDB는 문서 모델을 완벽하게 지원하여 문서를 저장하거나 가져오는 데 아무런 문제가 없습니다. 새로운 데이터 형식도 JSON 표준과 완전히 호환되어 문서 요소를 서로 중첩하는 것도 가능합니다. 문서 경로 역참조 연산자를 사용하면 전체 문서를 가져오지 않고도 개별 요소를 읽거나 쓸 수 있습니다. 그 밖에도 이번 릴리스에는 데이터 항목을 읽어오거나 작성할 때 프로젝션, 조건 및 업데이트 작업을 지정할 수 있는 새로운 표현식 파라미터가 추가되었습니다. JSON을 이용한 문서 모델 지원에 대한 자세한 내용은 데이터 타입 및 DynamoDB에서 표현식 사용 섹션을 참조하세요. | 2014년 10월 7일 |
| 유연한 확장성 | 테이블 및 글로벌 보조 인덱스의 경우 테이블 및 계정 단위 제한만 벗어나지 않는다면 할당된 읽기 및 쓰기 처리 용량을 얼마든지 늘릴 수 있습니다. 자세한 내용은 Amazon DynamoDB의 할당량 섹션을 참조하세요. | 2014년 10월 7일 |
| 항목 크기 증가 | DynamoDB의 최대 항목 크기가 64KB에서 400KB로 늘어났습니다. 자세한 내용은 Amazon DynamoDB의 할당량 섹션을 참조하세요. | 2014년 10월 7일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 조건식 향상 | DynamoDB는 조건식에 사용할 수 있는 연산자를 늘려 조건부 업로드, 업데이트 및 삭제의 유연성을 한층 높였습니다. 새로 사용 가능한 연산자는 속성이 존재하는지, 특정 값보다 큰지 작은지, 두 값의 사이인지, 특정 문자로 시작하는지 등을 검사할 수 있습니다. 또한 DynamoDB는 여러 조건을 평가하기 위해 옵션으로 OR 연산자를 제공합니다. 기본적으로 표현식의 여러 조건은 AND로 모두 이어지기 때문에 모든 조건이 true인 경우에 한해 표현식이 true로 평가됩니다. 하지만 OR를 지정할 경우 조건 중 하나 이상만 true이면 표현식이 true로 평가됩니다. 자세한 내용은 DynamoDB의 항목 및 속성 작업 섹션을 참조하세요. | 2014년 4월 24일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 쿼리 필터 | DynamoDB Query API는 새로운 QueryFilter 옵션을 지원합니다. 기본적으로 Query를 실행하면 특정 파티션 키 값 및 정렬 키 조건(선택 사항)과 일치하는 항목을 찾습니다. 하지만 Query 필터는 키 외의 다른 속성에 조건식을 적용합니다. 예를 들어 Query 필터를 추가할 경우 필터 조건과 일치하지 않는 항목은 제외하고 Query 결과가 애플리케이션에 반환됩니다. 자세한 내용은 DynamoDB에서 테이블 쿼리 섹션을 참조하세요. | 2014년 4월 24일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| AWS Management Console을 사용하여 데이터 내보내기 및 가져오기 | DynamoDB 콘솔은 DynamoDB 테이블의 데이터를 더욱 쉽게 내보내고 가져올 수 있도록 기능이 향상되었습니다. 단지 몇 번만 클릭하면 AWS Data Pipeline을 설정하여 워크플로우를 체계화할 뿐만 아니라 Amazon Elastic MapReduce 클러스터를 설정하여 DynamoDB 테이블에서 Amazon S3 버킷으로 데이터를 복사하거나 그 반대로 복사할 수 있습니다. 또한 내보내기나 가져오기를 1회로 제한하여 실행하거나 1일 내보내기 작업을 설정할 수도 있습니다. 그 밖에 교차 리전 내보내기 및 가져오기, DynamoDB 데이터를 AWS 리전 테이블에서 다른 AWS 리전 테이블로 복사하기 등도 가능합니다. | 2014년 3월 6일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 상위 수준 API 설명서의 재편성 | 이제 다음과 같은 API에 대한 정보를 더욱 쉽게 찾을 수 있습니다.<br>• Java: DynamoDBMappper<br>• .NET: 문서 모델 및 객체 지속성 모델<br><br>이러한 상위 수준 API 설명서는 현재 여기(DynamoDB에 대한 높은 수준의 프로그래밍 인터페이스)에서 확인할 수 있습니다. | 2014년 1월 20일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 글로벌 보조 인덱스 | DynamoDB는 글로벌 보조 인덱스를 지원하는 기능이 추가되었습니다. 글로벌 보조 인덱스를 정의할 때도 로컬 보조 인덱스와 마찬가지로 테이블의 대체 키를 사용해 인덱스에 대한 쿼리 요청을 실행합니다. 로컬 보조 인덱스와 다른 점은, 글로벌 보조 인덱스는 파티션 키가 테이블과 달라도 된다는 것입니다. 즉, 어떤 스칼라 테이블 속성이든 상관없습니다. 정렬 키는 선택 사항이지만 마찬가지로 어떤 스칼라 테이블 속성이든 사용할 수 있습니다. 글로벌 보조 인덱스 역시 자체적으로 할당 처리량 설정 값을 가지며, 상위 테이블의 처리량 설정 값과는 별도입니다. 자세한 내용은 DynamoDB에서 보조 인덱스를 사용하여 데이터 액세스 개선 및 DynamoDB에서 글로벌 보조 인덱스 사용 섹션을 참조하세요. | 2013년 12월 12일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 세분화된 액세스 제어 | DynamoDB는 세분화된 액세스 제어를 지원하는 기능이 추가되었습니다. 이 기능의 추가로 이제 고객이 직접 DynamoDB 테이블 또는 보조 인덱스의 개별 항목과 속성에 액세스할 수 있는 보안 주체(사용자, 그룹 또는 역할)를 지정할 수 있습니다. 또한 애플리케이션이 웹 자격 증명 연동 기능을 이용해 사용자 인증 작업을 Facebook, Google, 또는 Login with Amazon 같은 타사 자격 증명 공급자에게 분담할 수도 있습니다. 이러한 방식으로 애플리케이션(모바일 앱 포함)이 엄청난 수의 사용자까지 처리할 뿐만 아니라 권한이 없는 사용자는 DynamoDB 데이터 항목에 액세스하지 못하도록 제한할 수도 있습니다. 자세한 내용은 IAM 정책 조건을 사용하여 세분화된 액세스 제어 구현 섹션을 참조하세요. | 2013년 10월 29일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 4KB 읽기 용량 단위 크기 | 읽기 용량 단위의 크기가 1KB에서 4KB로 늘어났습니다. 이러한 크기의 증가로 많은 애플리케이션에 필요하여 할당되는 읽기 용량 단위의 수를 줄일 수 있게 되었습니다. 예를 들어 이번 릴리스 이전에는 10KB 항목을 읽어오려면 읽기 용량 단위 10개를 소비하였지만 이제는 단위 3개(10KB / 4KB, 다음 4KB 배수로 반올림)면 충분합니다. 자세한 내용은 DynamoDB 처리량 용량 섹션을 참조하세요. | 2013년 5월 14일 |
| 병렬 스캔 | DynamoDB는 병렬 스캔 작업을 지원하는 기능이 추가되었습니다. 이제 애플리케이션이 테이블 하나를 여러 논리 세그먼트로 나눠 동시에 모든 세그먼트를 스캔할 수 있습니다. 이 기능의 추가로 스캔 시간이 줄어들면서 테이블에 할당된 읽기 용량을 최대한 활용할 수 있게 되었습니다. 자세한 내용은 DynamoDB에서 테이블 스캔 섹션을 참조하세요. | 2013년 5월 14일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 로컬 보조 인덱스 | DynamoDB는 로컬 보조 인덱스를 지원하는 기능이 추가되었습니다. 이제 키가 아닌 속성으로 대체 정렬 키 인덱스를 정의한 다음 이 인덱스를 쿼리 요청에 사용하면 됩니다. 로컬 보조 인덱스를 이용하면 애플리케이션이 여러 차원의 데이터 항목을 효율적으로 가져올 수 있습니다. 자세한 내용은 DynamoDB의 로컬 보조 인덱스 섹션을 참조하세요. | 2013년 4월 18일 |
| 새로운 API 버전 | 이번 릴리스에서는 DynamoDB에 새로운 API 버전(2012-08-10)이 추가되었습니다. 이전 API 버전(2011-12-05)도 여전히 지원되어 기존 애플리케이션과 역호환도 가능합니다. 신규 애플리케이션은 새로운 API 버전(2012-08-10)을 사용해야 합니다. 기존 애플리케이션은 API 버전 2012-08-10으로 마이그레이션하는 것이 바람직합니다. 새로운 DynamoDB 기능(로컬 보조 인덱스 등)은 이전 API 버전으로 백포트(backport)가 지원되지 않기 때문입니다. API 버전 2012-08-10에 대한 자세한 내용은 Amazon DynamoDB API 참조를 참조하세요. | 2013년 4월 18일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| IAM 정책 변수 지원 | IAM 액세스 정책 언어는 이제 변수를 지원합니다. 정책이 평가되면 모든 정책 변수가 인증된 사용자의 세션에서 컨텍스트 기반 정보에 의해 제공되는 값으로 대체됩니다. 정책 변수를 사용하여 정책의 모든 구성 요소를 명시적으로 나열하지 않고 범용 정책을 정의할 수 있습니다. 정책 변수에 대한 자세한 내용은 AWS Identity and Access Management IAM 사용 가이드에서 정책 변수 섹션을 참조하세요.<br><br>DynamoDB의 정책 변수 예제는 Amazon DynamoDB의 Identity and Access Management 섹션을 참조하세요. | 2013년 4월 4일 |
| PHP 코드 예제가 AWS SDK for PHP 버전 2용으로 업데이트됨 | AWS SDK for PHP의 버전 2가 출시되었습니다. 이와 함께 신규 SDK를 사용할 수 있도록 Amazon DynamoDB 개발자 안내서의 PHP 코드 예제 역시 업데이트되었습니다. SDK 버전 2에 대한 자세한 내용은 AWS SDK for PHP 섹션을 참조하세요. | 2013년 1월 23일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 새로운 엔드포인트 | DynamoDB가 AWS GovCloud(미국 서부) 리전으로 확장됩니다. 현재 서비스 엔드포인트 및 프로토콜 목록은 Regions and Endpoints를 참조하세요. | 2012년 12월 3일 |
| 새로운 엔드포인트 | DynamoDB가 남아메리카(상파울루) 리전으로 확장됩니다. 현재 지원되는 엔드포인트 목록은 Regions and Endpoints를 참조하세요. | 2012년 12월 3일 |
| 새로운 엔드포인트 | DynamoDB가 아시아 태평양(시드니) 리전으로 확장됩니다. 현재 지원되는 엔드포인트 목록은 Regions and Endpoints를 참조하세요. | 2012년 11월 13일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| DynamoDB는 CRC32 체크섬과 강력한 읽기 일관성의 일괄 가져오기를 지원할 뿐만 아니라 동시에 가능한 테이블 업데이트 수의 제한도 제거하였습니다. | • DynamoDB는 HTTP 페이로드에서 CRC32 체크섬을 계산하여 새로운 헤더인 `x-amz-crc32` 로 반환합니다. 자세한 내용은 DynamoDB 하위 수준 API 섹션을 참조하세요.<br>• 기본적으로 `BatchGetItem` API의 읽기 작업은 최종 일관성을 사용합니다. 하지만 `BatchGetItem` 의 새로운 파라미터인 `ConsistentRead` 는 모든 요청 테이블에 강력한 읽기 일관성을 선택할 수 있습니다. 자세한 내용은 설명 섹션을 참조하세요.<br>• 이번 릴리스에서는 동시에 업데이트할 수 있는 테이블 수에 대한 제한이 일부 사라졌습니다. 한 번에 업데이트할 수 있는 전체 테이블 수는 여전히 10개이지만 `CREATING`, `UPDATING` 또는 `DELETING` 등 이 테이블들의 상태 조합은 제한이 없습니다. 또한 테이블 하나당 ReadCapacityUnits 또는 WriteCapacityUnits를 늘리거나 줄일 수 있는 최소 수도 더 이상 없습니다. 자세한 내용은 Amazon DynamoDB의 할당량 섹션을 참조하세요. | 2012년 11월 2일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 모범 사례 설명서 | Amazon DynamoDB 개발자 안내서는 쿼리 및 스캔 작업을 위한 권장 방안과 함께 테이블 및 항목 작업에 유용하게 사용할 수 있는 모범 사례를 소개합니다. | 2012년 9월 28일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 이진수 데이터 형식 지원 | DynamoDB는 숫자 및 문자열 형식은 물론이고 이제 이진수 데이터 형식도 지원합니다.<br><br>이번 릴리스 이전에는 이진수 데이터를 저장하려면 먼저 문자열 형식으로 변환하여 DynamoDB에 저장해야 했습니다. 이러한 클라이언트 측 변환 작업 외에도 종종 변환 도중 데이터 항목의 크기가 커져서 스토리지가 더 필요하거나 프로비저닝 처리 용량을 추가해야 하는 잠재적 가능성도 발생하였습니다.<br><br>하지만 이제는 이진수 형식 속성을 지원하여 압축 데이터, 암호화 데이터 및 이미지 등 이진수 데이터를 얼마든지 저장할 수 있게 되었습니다. 자세한 내용은 데이터 타입 섹션을 참조하세요. 그리고 실제로 AWS SDK를 사용하여 이진수 형식 데이터를 처리하는 예제는 다음 섹션을 참조하세요.<br><br>• 예: AWS SDK for Java 문서 API를 사용하여 이진 형식 속성 처리<br>• 예: AWS SDK for .NET 하위 수준 API를 사용하여 이진 형식 속성 처리 | 2012년 8월 21일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| | AWS SDK에 이진수 데이터 형식 지원이 추가되면서 최신 SDK를 다운로드하고 기존 애플리케이션까지 모두 업데이트해야 할 수도 있습니다. AWS SDK 다운로드에 대한 자세한 내용은 .NET 코드 예시 섹션을 참조하세요. | |
| DynamoDB 콘솔을 사용하여 DynamoDB 테이블 항목을 업데이트 및 복사할 수 있음 | 이제 DynamoDB 사용자는 DynamoDB 콘솔을 사용해 테이블 항목의 추가 및 삭제는 물론이고 업데이트와 복사까지 가능하게 되었습니다. 이 새로운 기능의 추가로 콘솔을 통한 개별 항목의 변경도 쉬워졌습니다. | 2012년 8월 14일 |
| DynamoDB의 최소 테이블 처리량 요건 축소 | DynamoDB는 이제 최소 테이블 처리량 요건이 특히 쓰기 용량 단위 1개와 읽기 용량 단위 1개까지 축소되었습니다. 자세한 내용은 Amazon DynamoDB 개발자 안내서의 Amazon DynamoDB의 할당량 주제를 참조하세요. | 2012년 8월 9일 |
| 서명 버전 4 지원 | DynamoDB는 이제 인증 요청에 서명 버전 4를 지원합니다. | 2012년 7월 5일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| DynamoDB 콘솔의 테이블 탐색기 지원 | DynamoDB 콘솔이 이제 테이블 탐색기를 지원하여 테이블에 저장된 데이터를 찾아 쿼리를 실행할 수 있습니다. 또한 새로운 항목을 삽입하거나 기존 항목을 삭제할 수도 있습니다. 이 기능을 지원하기 위해 `SampleData` 영역과 콘솔 사용 영역이 업데이트되었습니다. | 2012년 5월 22일 |
| 새로운 엔드포인트 | DynamoDB 가용성이 미국 서부(캘리포니아 북부) 리전, 미국 서부(오레곤) 리전 및 아시아 태평양(싱가포르) 리전의 새로운 엔드포인트로 확장됩니다.<br><br>현재 지원되는 엔드포인트 목록은 Regions and Endpoints를 참조하세요. | 2012년 4월 24일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| BatchWriteItem API 지원 | DynamoDB는 이제 일괄 쓰기 API를 지원하여 단 한 번의 API 호출로 하나 이상의 테이블에 다수의 항목을 업로드하거나 삭제할 수 있습니다. DynamoDB 일괄 쓰기 API에 대한 자세한 내용은 BatchWriteItem 섹션을 참조하세요.<br><br>AWS SDK의 항목 작업 및 일괄 쓰기 기능 사용에 대한 자세한 내용은 DynamoDB의 항목 및 속성 작업 및 .NET 코드 예시 섹션을 참조하세요. | 2012년 4월 19일 |
| 오류 코드의 추가 문서화 | 자세한 내용은 DynamoDB 관련 오류 처리 섹션을 참조하세요. | 2012년 4월 5일 |
| 새로운 엔드포인트 | DynamoDB가 아시아 태평양(도쿄) 리전으로 확장됩니다. 현재 지원되는 엔드포인트 목록은 Regions and Endpoints를 참조하세요. | 2012년 29월 2일 |
| `ReturnedItemCount` 메트릭 추가 | 새로운 메트릭인 `ReturnedItemCount` 의 추가로 DynamoDB의 쿼리 또는 스캔 작업에서 응답으로 반환되었던 항목 수를 이제는 CloudWatch에서도 모니터링할 수 있습니다. | 2012년 24월 2일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 증분 값에 대한 예제 추가 | DynamoDB는 기존 숫자 값을 일정하게 늘리거나 줄이는 기능을 지원합니다. 기존 값을 늘리는 예제는 다음 웹사이트의 "항목 업데이트" 섹션을 참조하세요.<br><br>항목 작업: Java.<br><br>항목 작업: .NET. | 2012년 1월 25일 |
| 초기 제품 릴리스 | DynamoDB는 베타 버전의 신규 서비스로 출시됩니다. | 2012년 1월 18일 |

# DynamoDB의 레거시 기능

다음 주제는 DynamoDB가 여전히 지원하는 레거시 기능입니다. 이러한 기능에 대한 개발은 활발히 이루어지지 않습니다.

주제

- 글로벌 테이블 버전 2017.11.29(레거시)
- 이전 하위 수준 API 버전(2011-12-05)
- 레거시 DynamoDB 조건부 파라미터

# 글로벌 테이블 버전 2017.11.29(레거시)

> **Important**
>
> 이 설명서는 버전 2017.11.29(레거시)의 글로벌 테이블에 대한 것이므로 새 글로벌 테이블의 경우 사용하지 않아야 합니다. 가능하면 글로벌 테이블 버전 2019.11.21(현재)을 사용해야 합니다. 이는 2017.11.29(레거시)보다 유연성과 효율성이 뛰어나고 쓰기 용량을 적게 소비합니다.
>
> 사용 중인 버전을 확인하려면 글로벌 테이블 버전 확인 섹션을 참조하세요. 기존 전역 테이블을 버전 2017.11.29(레거시)에서 버전 2019.11.21(현재)로 업데이트하는 경우 DynamoDB 글로벌 테이블 버전 섹션을 참조하세요.

주제

- 전역 테이블: 작동 방식
- 전역 테이블 관리 모범 사례 및 요구 사항
- 전역 테이블 생성
- 전역 테이블 모니터링
- 전역 테이블에 IAM 사용

# 전역 테이블: 작동 방식

> **Important**
>
> 이 설명서는 버전 2017.11.29(레거시)의 글로벌 테이블에 대한 것이므로 새 글로벌 테이블의 경우 사용하지 않아야 합니다. 가능하면 글로벌 테이블 버전 2019.11.21(현재)을 사용해야 합니다. 이는 2017.11.29(레거시)보다 유연성과 효율성이 뛰어나고 쓰기 용량을 적게 소비합니다.
>
> 사용 중인 버전을 확인하려면 글로벌 테이블 버전 확인 섹션을 참조하세요. 기존 전역 테이블을 버전 2017.11.29(레거시)에서 버전 2019.11.21(현재)로 업데이트하는 경우 DynamoDB 글로벌 테이블 버전 섹션을 참조하세요.

다음 단원에서는 Amazon DynamoDB의 전역 테이블에 대한 동작과 개념에 대해 설명합니다.

## 버전 2017.11.29(레거시)의 전역 테이블 개념

전역 테이블이란 하나의 AWS 계정에서 소유한 한 개 이상의 복제 테이블 모음입니다.

복제 테이블(줄여서 복제본이라고도 함)은 전역 테이블의 일부로 기능하는 단일 DynamoDB 테이블입니다. 각 복제본에는 동일한 데이터 항목 집합이 저장됩니다. 전역 테이블은 AWS 리전당 한 개의 복제 테이블을 가질 수 있습니다.

다음은 전역 테이블을 만드는 방법을 개념적으로 간단히 설명한 것입니다.

1. AWS 리전에서 DynamoDB Streams를 활성화하여 일반 DynamoDB 테이블을 생성합니다.
2. 데이터를 복제하려는 다른 리전 모두에 대해 1단계를 반복합니다.
3. 생성한 테이블을 기반으로 DynamoDB 전역 테이블을 정의합니다.

AWS Management Console은 이러한 작업을 자동화하므로 전역 테이블을 빠르고 쉽게 만들 수 있습니다. 자세한 내용은 전역 테이블 생성 섹션을 참조하세요.

생성된 DynamoDB 전역 테이블은 리전당 한 개씩 여러 개의 복제 테이블로 구성되며, DynamoDB는 이를 단일 단위로 처리합니다. 모든 복제본은 동일한 테이블 이름과 동일한 프라이머리 키 스키마를 갖습니다. 애플리케이션이 한 리전의 복제 테이블에 데이터를 쓰면 DynamoDB가 이 쓰기를 다른 AWS 리전에 있는 다른 복제 테이블에 자동으로 전파합니다.

> **Important**
>
> 전역 테이블은 테이블 데이터를 동기화 상태로 유지하기 위해 모든 항목에 대한 다음 속성을 자동으로 만듭니다.
>
> - `aws:rep:deleting`
> - `aws:rep:updatetime`
> - `aws:rep:updateregion`
>
> 이 속성을 변경하거나 같은 이름의 속성을 생성해서는 안 됩니다.

복제본 테이블을 전역 테이블에 추가하여 추가 리전에서 사용할 수 있습니다. (이렇게 하려면 전역 테이블이 비어 있어야 합니다. 즉, 복제 테이블 중 어느 테이블에도 데이터가 없어야 합니다.)

전역 테이블에서 복제본 테이블을 제거할 수도 있습니다. 이렇게 하면 해당 테이블은 전역 테이블과 연결이 끊어집니다. 따라서 전역 테이블과 더 이상 상호 작용을 수행하지 않으며 전역 테이블과 데이터도 주고받지 않습니다.

> **Warning**
>
> 복제본을 제거하는 것은 원자 프로세스가 아님에 유의하세요. 일관된 동작과 알려진 상태인지 확인하려면 사전에 제거할 복제본에서 애플리케이션 쓰기 트래픽을 전환해 볼 수도 있습니다. 복제본을 제거한 후 모든 복제본 리전 엔드포인트에 복제본이 연결 해제된 것으로 표시될 때까지 기다렸다가 자체 격리된 리전 테이블에 추가 쓰기를 수행합니다.

## 일반적인 작업

글로벌 테이블에 대한 일반적인 작업은 다음과 같이 작동합니다.

일반 테이블과 마찬가지로 글로벌 테이블의 복제본 테이블을 삭제할 수 있습니다. 그러면 해당 리전으로의 복제가 중지되고 해당 리전에 보관된 테이블 복사본이 삭제됩니다. 복제를 분리할 수 없으며 테이블의 복사본이 독립 엔터티로 존재하도록 할 수 없습니다.

> **Note**
>
> 새 리전을 시작하는 데 사용한 후 최소 24시간이 지나야 소스 테이블을 삭제할 수 있습니다. 너무 빨리 삭제하려고 하면 오류가 발생할 수 있습니다.

충돌은 애플리케이션이 서로 다른 리전에서 동시에 동일한 항목을 업데이트할 경우 발생합니다. 최종 일관성을 보장하기 위해 DynamoDB 글로벌 테이블은 '최종 쓰기 우선' 방법을 사용하여 동시 업데이트 간에 조정합니다. 모든 복제본은 최종 업데이트에 합의하며 모두 동일한 데이터를 갖는 상태가 됩니다.

> **Note**
>
> 충돌을 방지하는 몇 가지 방법은 다음과 같습니다.
>
> - IAM 정책을 사용하여 한 리전의 테이블에 대한 쓰기만 허용합니다.
> - IAM 정책을 사용하여 사용자를 한 리전으로만 라우팅하고 다른 리전은 유휴 대기 상태로 유지하거나, 홀수 사용자를 한 리전으로, 짝수 사용자를 다른 리전으로 번갈아 라우팅합니다.
> - Bookmark = Bookmark + 1과 같은 멱등성이 없는 업데이트의 사용을 피하고 가급적 Bookmark=25와 같은 정적 업데이트를 사용합니다.

## 전역 테이블 모니터링

CloudWatch를 사용하여 `ReplicationLatency` 지표를 관찰할 수 있습니다. 이 지표는 한 복제 테이블에 대한 DynamoDB Streams에 나타나는 업데이트된 항목과 글로벌 테이블의 다른 복제본에 나타나는 항목 간의 경과 시간을 추적합니다. `ReplicationLatency`는 밀리초 단위로 표현되며, 모든 소스-리전 및 대상-리전 쌍에 대해 내보내집니다. 이 지표는 글로벌 테이블 v2에서 제공하는 유일한 CloudWatch 지표입니다.

발생하게 되는 지연 시간은 선택한 리전 간의 거리와 기타 변수에 따라 달라집니다. 리전에 대한 0.5초~2.5초 범위의 지연 시간은 동일한 지리적 영역 내에서 일반적일 수 있습니다.

## TTL(Time To Live)

TTL(Time To Live)을 사용하여 해당 값이 항목의 만료 시간을 나타내는 속성 이름을 지정할 수 있습니다. 이 값은 Unix epoch가 시작된 이후 초 단위의 숫자로 지정됩니다.

글로벌 테이블 레거시 버전에서는 TTL 삭제가 다른 복제본에 자동으로 복제되지 않습니다. TTL 규칙을 통해 항목을 삭제하면 쓰기 단위를 사용하지 않고 해당 작업이 수행됩니다.

소스 및 대상 테이블의 프로비저닝된 쓰기 용량이 매우 낮은 경우 TTL 삭제에 쓰기 용량이 필요하므로 제한이 발생할 수 있습니다.

## 글로벌 테이블을 사용한 스트림 및 트랜잭션

각 글로벌 테이블은 해당 쓰기의 시작 지점에 관계없이 모든 쓰기를 기반으로 독립적인 스트림을 생성합니다. 이 DynamoDB 스트림을 한 리전 또는 모든 리전에서 독립적으로 사용하도록 선택할 수 있습니다.

로컬 쓰기는 처리하되 복제된 쓰기는 처리하지 않으려는 경우 각 항목에 고유한 리전 속성을 추가할 수 있습니다. 그런 다음 Lambda 이벤트 필터를 사용하여 로컬 리전의 쓰기용으로만 Lambda를 호출할 수 있습니다.

트랜잭션 작업은 원래 쓰기 작업이 실행된 리전에서만 ACID(원자성, 일관성, 격리 및 내구성) 보장을 제공합니다. 글로벌 테이블에서는 트랜잭션이 리전 간에 지원되지 않습니다.

예를 들어 미국 동부(오하이오) 및 미국 서부(오레곤) 리전에 복제본이 있는 글로벌 테이블에 대해 미국 동부(오하이오)에서 TransactWriteItems 작업을 수행할 경우 변경 내용이 복제될 때 미국 서부(오레곤)에서 부분적으로 완료된 트랜잭션을 관찰할 수 있습니다. 변경 내용은 소스 리전에서 커밋된 이후에만 다른 리전에 복제됩니다.

> **Note**
>
> - 글로벌 테이블은 DynamoDB를 직접 업데이트하여 DynamoDB Accelerator에 '쓰기'합니다. 따라서 DAX는 기한이 지난 데이터를 보유하고 있다는 사실을 인식하지 못합니다. DAX 캐시는 캐시의 TTL이 만료되는 경우에만 새로 고쳐집니다.
> - 글로벌 테이블의 태그는 자동으로 전파되지 않습니다.

## 읽기 및 쓰기 처리량

글로벌 테이블은 다음과 같은 방식으로 읽기 및 쓰기 처리량을 관리합니다.

- 쓰기 용량은 리전의 모든 테이블 인스턴스에서 동일해야 합니다.
- 버전 2019.11.21(현재)에서는 테이블이 Auto Scaling을 지원하도록 설정되거나 온디맨드 모드인 경우 쓰기 용량이 자동으로 동기화된 상태로 유지됩니다. 각 리전에서 프로비저닝된 현재 쓰기 용량은 동기화된 Auto Scaling 설정 내에서 독립적으로 증가 및 감소합니다. 테이블이 온디맨드 모드로 전환되면 해당 모드가 다른 복제본과 동기화됩니다.

- 읽기가 동일하지 않을 수 있으므로 읽기 용량은 리전마다 다를 수 있습니다. 테이블에 글로벌 복제본을 추가하면 소스 리전의 용량이 전파됩니다. 생성 후 하나의 복제본에 대한 읽기 용량을 조정할 수 있으며 이 새로운 설정은 다른 쪽으로 전송되지 않습니다.

## 정합성 및 충돌 해결

복제본 테이블의 항목이 변경되면 동일한 전역 테이블에 있는 다른 모든 복제본에 복제됩니다. 전역 테이블에 새로 쓰여진 항목은 일반적으로 몇 초만에 모든 복제본 테이블에 전파됩니다.

전역 테이블에서 각 복제 테이블에는 동일한 데이터 항목 집합이 저장됩니다. DynamoDB는 일부 항목만의 부분 복제를 지원하지 않습니다.

애플리케이션은 다른 복제본 테이블에 대해 데이터를 읽고 쓸 수 있습니다. DynamoDB는 리전 전체에서 최종 읽기 일관성을 지원하지만 리전 전체에서 강력히 일관된 읽기는 지원하지 않습니다. 애플리케이션이 최종적으로 일관된 읽기만 사용하고 하나의 AWS 리전에 대해 읽기만 발행할 경우 수정 없이 수행됩니다. 하지만 애플리케이션이 강력히 일관된 읽기를 요구하면 동일 리전에서 강력히 일관된 읽기와 쓰기를 모두 수행해야 합니다. 그렇지 않고 한 리전에 쓰기를 하고 다른 리전에서 읽기를 할 경우, 다른 리전에서 최근에 수행된 결과가 반영되지 않은 기한 경과 데이터가 읽기 응답에 포함될 수 있습니다.

충돌은 애플리케이션이 서로 다른 리전에서 동시에 동일한 항목을 업데이트할 경우 발생합니다. 최종 일관성을 보장하기 위해 DynamoDB 전역 테이블은 동시 업데이트에 대해 최종 쓰기 우선 적용 조정을 사용하며, DynamoDB는 최선을 다해 최종 쓰기를 결정합니다. 이러한 충돌 해결 방법을 사용하여 모든 복제본은 최종 업데이트에 합의하며 모두 동일한 데이터를 갖는 상태가 됩니다.

## 가용성과 내구성

한 AWS 리전이 분리되거나 저하되면, 애플리케이션이 다른 리전으로 리디렉션하여 다른 복제 테이블에서 읽기 및 쓰기를 수행할 수 있습니다. 요청을 언제 다른 리전으로 재지정할지를 결정하는 사용자 지정 비즈니스 로직을 적용할 수 있습니다.

리전이 분리되거나 저하되면 DynamoDB는 수행된 쓰기 기록을 유지하지만 모든 복제 테이블로 전파하지는 않습니다. 리전이 다시 온라인 상태로 되면 DynamoDB는 해당 리전에서 보류 중인 쓰기 전파를 재개하여 다른 리전의 복제 테이블로 전파합니다. 다시 온라인 상태로 된 리전에 대한 다른 복제본 테이블의 쓰기 전파도 재개됩니다. 이전에 성공한 모든 쓰기 작업은 리전이 격리된 기간에 관계없이 결국 전파됩니다.

# 전역 테이블 관리 모범 사례 및 요구 사항

> **Important**
>
> 이 설명서는 버전 2017.11.29(레거시)의 글로벌 테이블에 대한 것이므로 새 글로벌 테이블의 경우 사용하지 않아야 합니다. 가능하면 글로벌 테이블 버전 2019.11.21(현재)을 사용해야 합니다. 이는 2017.11.29(레거시)보다 유연성과 효율성이 뛰어나고 쓰기 용량을 적게 소비합니다.
>
> 사용 중인 버전을 확인하려면 글로벌 테이블 버전 확인 섹션을 참조하세요. 기존 전역 테이블을 버전 2017.11.29(레거시)에서 버전 2019.11.21(현재)로 업데이트하는 경우 DynamoDB 글로벌 테이블 버전 섹션을 참조하세요.

Amazon DynamoDB 전역 테이블을 사용하여 AWS 리전 간에 테이블 데이터를 복제할 수 있습니다. 전역 테이블의 복제본 테이블과 보조 인덱스의 쓰기 용량을 동일하게 설정해 데이터를 적절히 복제하는 것이 중요합니다.

주제


- 글로벌 테이블 버전
- 새 복제본 테이블 추가 요구 사항
- 용량 관리 모범 사례 및 요구 사항


## 글로벌 테이블 버전

DynamoDB 글로벌 테이블에는 글로벌 테이블 버전 2019.11.21(현재)과 글로벌 테이블 버전 2017.11.29(레거시)의 두 가지 버전이 있습니다. 가능하면 글로벌 테이블 버전 2019.11.21(현재)을 사용해야 합니다. 이는 2017.11.29(레거시)보다 유연성과 효율성이 뛰어나고 쓰기 용량을 적게 소비합니다.

사용 중인 버전을 확인하려면 글로벌 테이블 버전 확인 섹션을 참조하세요. 기존 글로벌 테이블을 버전 2017.11.29(레거시)에서 버전 2019.11.21(현재)로 업데이트하는 경우 글로벌 테이블 업그레이드 섹션을 참조하세요.

## 새 복제본 테이블 추가 요구 사항

새 복제본 테이블을 전역 테이블에 추가하려면 다음 조건이 모두 충족되어야 합니다.

- 테이블은 모든 다른 복제본과 동일한 파티션 키를 갖습니다.

- 테이블에는 지정된 동일한 쓰기 용량 관리 설정이 있어야 합니다.
- 테이블은 모든 다른 복제본과 동일한 이름을 갖습니다.
- 테이블에는 DynamoDB Streams가 활성화되어야 하며, 스트림에 항목의 새 이미지와 이전 이미지가 모두 포함되어야 합니다.
- 전역 테이블의 신규 또는 기존 복제본 테이블은 데이터를 포함할 수 없습니다.

글로벌 보조 인덱스가 지정되어 있으면, 다음 조건을 충족해야 합니다.

- 글로벌 보조 인덱스의 이름이 같아야 합니다.
- 전역 보조 인덱스의 파티션 키와 정렬 키(존재하는 경우)가 같아야 합니다.

> **Important**
>
> 쓰기 용량 설정은 전역 테이블의 복제본 테이블과 해당 보조 인덱스에 대해 일관되게 설정해야 합니다. 전역 테이블에 대한 쓰기 용량 설정을 업데이트하려면 DynamoDB 콘솔이나 `UpdateGlobalTableSettings` API 작업을 사용하는 것이 좋습니다. `UpdateGlobalTableSettings`는 쓰기 용량 설정에 대한 변경 사항을 전역 테이블에 있는 모든 복제 테이블과 해당 보조 인덱스에 자동으로 적용합니다. `UpdateTable`, `RegisterScalableTarget` 및 `PutScalingPolicy` 작업을 사용할 경우 각 복제본 테이블과 해당 보조 인덱스에 개별적으로 변경 사항을 적용해야 합니다. 자세한 내용은 Amazon DynamoDB API 참조의 UpdateGlobalTableSettings를 참조하세요.
>
> 프로비저닝된 쓰기 용량 설정을 관리하기 위해 Auto Scaling을 활성화하는 것이 좋습니다. 쓰기 용량 설정을 수동으로 관리하고 싶다면, 동일하게 복제한 WCU(쓰기 용량 단위)를 모든 복제본 테이블에 프로비저닝해야 합니다. 또한 여러 전역 테이블의 일치하는 보조 인덱스에 동일하게 복제한 WCU(쓰기 용량 단위)를 프로비저닝해야 합니다.
>
> 또한 적절한 AWS Identity and Access Management(IAM) 권한이 있어야 합니다. 자세한 내용은 전역 테이블에 IAM 사용 섹션을 참조하세요.

## 용량 관리 모범 사례 및 요구 사항

DynamoDB에서 복제 테이블에 대한 용량 설정을 관리할 때 다음을 고려하세요.

### DynamoDB Auto Scaling 사용

프로비저닝된 모드를 사용하는 복제 테이블의 처리량 용량 설정을 관리하려면 DynamoDB Auto Scaling을 사용하는 것이 좋습니다. DynamoDB Auto Scaling은 실제 애플리케이션 워크로드를 기반으

로 각 복제 테이블에 대한 읽기 용량 단위(RCU) 및 쓰기 용량 단위(WCU)를 자동으로 조정합니다. 자세한 내용은 DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리 섹션을 참조하세요.

AWS Management Console을 사용하여 복제본 테이블을 생성하는 경우, 각 복제본 테이블에 대해 RCU(읽기 용량 유닛) 및 WCU(쓰기 용량 유닛)를 관리하기 위한 기본 Auto Scaling 설정과 함께 Auto Scaling이 기본적으로 활성화됩니다.

DynamoDB 콘솔이나 `UpdateGlobalTableSettings` 호출을 사용하여 복제 테이블 또는 보조 인덱스에 대한 Auto Scaling 설정을 변경할 경우, 전역 테이블에 있는 모든 복제 테이블과 해당 보조 인덱스에 변경 사항이 자동으로 적용됩니다. 이러한 변경 사항이 기존 Auto Scaling 설정을 덮어씁니다. 따라서 전역 테이블에 있는 복제본 테이블과 보조 인덱스에 대해 할당된 쓰기 용량 설정이 일관되게 유지됩니다. `UpdateTable`, `RegisterScalableTarget`, `PutScalingPolicy` 호출 등을 사용할 경우 각 복제본 테이블과 해당 보조 인덱스에 개별적으로 변경 사항을 적용해야 합니다.

> **Note**
>
> Auto Scaling에서 애플리케이션의 용량 변경이 만족스럽지 않거나(예측할 수 없는 워크로드) 설정(최소값, 최대값 또는 사용률 임계값)을 구성하지 않으려면 온디맨드 모드를 사용하여 전역 테이블에 대한 용량을 관리할 수 있습니다. 자세한 내용은 온디맨드 모드 섹션을 참조하세요.
>
> 전역 테이블에 대한 온디맨드 모드를 활성화한 경우 복제된 쓰기 요청 단위(rWCUs) 소비량이 rWCUs 프로비저닝 방법과 일치하게 됩니다. 예를 들어 두 추가 리전에 복제되는 로컬 테이블에 10번 쓸 경우 쓰기 요청 단위를 60회 사용하게 됩니다(10 + 10 + 10 = 30, 30 x 2 = 60). 사용된 60개의 쓰기 요청 단위에는 글로벌 테이블 버전 2017.11.29(레거시)에서 `aws:rep:deleting`, `aws:rep:updatetime` 및 `aws:rep:updateregion` 속성을 업데이트하는 데 사용된 추가 쓰기가 포함됩니다.

### 수동으로 용량 관리

DynamoDB Auto Scaling을 사용하지 않기로 결정하는 경우, 각 복제 테이블과 보조 인덱스에서 읽기 용량 설정과 쓰기 용량 설정을 수동으로 설정해야 합니다.

모든 복제본 테이블에 대해 각 리전에서 프로비저닝된 rWCU(복제된 쓰기 용량 단위)는 모든 리전에서 애플리케이션 쓰기에 필요한 총 rWCU 수에 2를 곱해야 합니다. 이는 로컬 리전에서 발생하는 애플리케이션 쓰기, 다른 리전에서 발생하는 복제된 애플리케이션 쓰기를 수용합니다. 예를 들어, 오하이오의 복제본 테이블의 초당 쓰기를 5건으로, 버지니아 북부 복제본 테이블의 초당 쓰기를 5건으로 설정하고 싶다면, 각 복제본 테이블에 20개 rWCU를 프로비저닝해야 합니다(5 + 5 = 10, 10 x 2 = 20).

전역 테이블에 대한 쓰기 용량 설정을 업데이트하려면 DynamoDB 콘솔이
나 `UpdateGlobalTableSettings` API 작업을 사용하는 것이 좋습니다.
`UpdateGlobalTableSettings`는 쓰기 용량 설정에 대한 변경 사항을 전역 테이블에 있는 모든 복제 테이블과 해당 보조 인덱스에 자동으로 적용합니다. `UpdateTable`, `RegisterScalableTarget` 및 `PutScalingPolicy` 작업을 사용할 경우 각 복제본 테이블과 해당 보조 인덱스에 개별적으로 변경 사항을 적용해야 합니다. 자세한 내용은 Amazon DynamoDB API 참조를 확인하세요.

> **Note**
>
> DynamoDB의 전역 테이블에 대한 설정(`UpdateGlobalTableSettings`)을 업데이트하려면 `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` 및 `application-autoscaling:DeregisterScalableTarget` 권한이 있어야 합니다. 자세한 내용은 전역 테이블에 IAM 사용 섹션을 참조하세요.

## 전역 테이블 생성

> **Important**
>
> 이 설명서는 버전 2017.11.29(레거시)의 글로벌 테이블에 대한 것이므로 새 글로벌 테이블의 경우 사용하지 않아야 합니다. 가능하면 글로벌 테이블 버전 2019.11.21(현재)을 사용해야 합니다. 이는 2017.11.29(레거시)보다 유연성과 효율성이 뛰어나고 쓰기 용량을 적게 소비합니다.
>
> 사용 중인 버전을 확인하려면 글로벌 테이블 버전 확인 섹션을 참조하세요. 기존 전역 테이블을 버전 2017.11.29(레거시)에서 버전 2019.11.21(현재)로 업데이트하는 경우 DynamoDB 글로벌 테이블 버전 섹션을 참조하세요.

이 단원에서는 Amazon DynamoDB 콘솔 또는 AWS Command Line Interface(AWS CLI)를 사용하여 전역 테이블을 생성하는 방법을 설명합니다.

주제

- 전역 테이블 생성(콘솔)
- 전역 테이블 생성(AWS CLI)

## 전역 테이블 생성(콘솔)

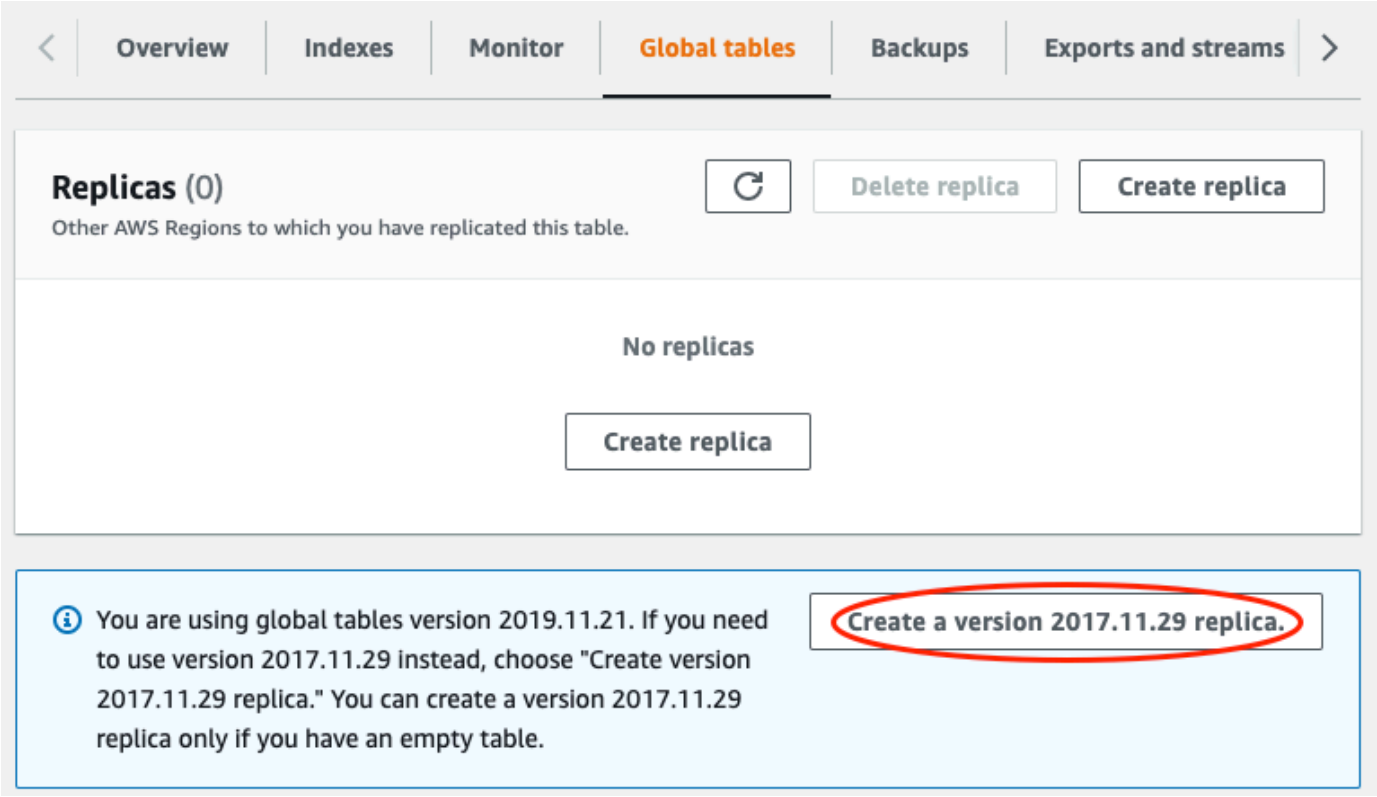
다음 단계에 따라 콘솔을 사용하여 전역 테이블을 생성합니다. 다음 예제에서는 미국 및 유럽의 복제본 테이블로 전역 테이블을 생성합니다.

1. https://console.aws.amazon.com/dynamodb/home에서 DynamoDB 콘솔을 엽니다. 이 예제의 경우, us-east-2(미국 동부 오하이오) 리전을 선택합니다.
2. 콘솔 왼쪽의 탐색 창에서 테이블을 선택합니다.
3. Create Table(테이블 생성)을 선택합니다.

   테이블 이름에 **`Music`**을(를) 입력합니다.

   기본 키에 **`Artist`**를 입력합니다. 정렬 키 추가를 선택하고 **`SongTitle`**를 입력합니다(**`Artist`**와 **`SongTitle`**은 모두 문자열이어야 함).

   테이블을 생성하려면 [Create]를 선택합니다. 이 테이블은 새로운 전역 테이블에서 첫 번째 복제본 테이블 역할을 합니다. 이는 나중에 추가하는 다른 복제본 테이블의 프로토타입입니다.
4. 글로벌 테이블 탭을 선택한 다음 버전 2017.11.29(레거시) 복제본 생성을 선택합니다.




5. 사용 가능한 복제 리전(Available replication Regions) 드롭다운에서 미국 서부(오레곤)(US West (Oregon)를 선택합니다.

콘솔에서 선택한 리전에 이름이 동일한 테이블이 없는지 확인합니다. 이름이 동일한 테이블이 있는 경우 해당 리전에서 새 복제본 테이블을 생성하려면 먼저 기존 테이블을 삭제해야 합니다.

6. 복제본 생성(Create Replica)을 선택합니다. 그러면 미국 서부(오레곤)에서 테이블 생성 프로세스가 시작됩니다.

   선택한 테이블(그리고 다른 복제본 테이블)의 전역 테이블 탭에 해당 테이블이 여러 리전에 복제되었다는 메시지가 표시됩니다.

7. 이제 다른 리전을 추가하여 전역 테이블이 미국과 유럽에 걸쳐 복제되고 동기화되도록 합니다. 이렇게 하려면 5단계를 반복하되, 이번에는 US West (Oregon)(미국 서부(오레곤)) 대신 EU(프랑크푸르트)(EU (Frankfurt))를 지정합니다.

8. 미국 동부(오하이오) 리전에서 AWS Management Console을 계속 사용해야 합니다. 왼쪽 탐색 메뉴의 항목(Items)에서 음악(Music) 테이블을 선택한 다음 항목 만들기(Create Item)를 선택합니다.

   a. [Artist]에 **`item_1`**를 입력합니다.

   b. [SongTitle]에 **`Song Value 1`**를 입력합니다.

   c. 해당 항목을 쓰려면 항목 만들기(Create item)를 선택합니다.

9. 잠시 후에 이 항목이 전역 테이블의 세 리전 모두에 복제됩니다. 제대로 되었는지 확인하려면 콘솔에서 오른쪽 상단 모서리에 있는 리전 선택기로 이동하고 Europe (Frankfurt)(유럽(프랑크푸르트))를 선택합니다. 유럽(프랑크푸르트)의 `Music` 테이블에 새 항목이 포함되어 있어야 합니다.

10. 9단계를 반복하고 미국 서부(오레곤)(US West (Oregon))을 선택하여 해당 리전의 복제를 확인합니다.

## 전역 테이블 생성(AWS CLI)

다음 단계에 따라 AWS CLI를 사용하여 `Music` 전역 테이블을 생성합니다. 다음 예제에서는 미국 및 유럽의 복제본 테이블로 전역 테이블을 생성합니다.

1. 미국 동부(오하이오)에서 DynamoDB Streams를 활성화하고(`NEW_AND_OLD_IMAGES`) 새 테이블(`Music`)을 생성합니다.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
```

```
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2
```

2. 동일한 `Music` 테이블을 미국 동부(버지니아 북부)에서 생성합니다.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-1
```

3. `us-east-2` 및 `us-east-1` 리전의 복제본 테이블로 구성된 전역 테이블(`Music`)을 생성합니다.

```
aws dynamodb create-global-table \
    --global-table-name Music \
    --replication-group RegionName=us-east-2 RegionName=us-east-1 \
    --region us-east-2
```

> **Note**
>
> 전역 테이블 이름(`Music`)은 각 복제본 테이블의 이름(`Music`)과 일치해야 합니다. 자세한 내용은 글로벌 테이블에 대한 모범 사례 섹션을 참조하세요.

4. 1단계 및 2단계에서 생성한 테이블과 동일한 설정을 사용하여 유럽(아일랜드)에 다른 테이블을 생성합니다.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
```

```
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region eu-west-1
```

이 단계를 수행한 후에는 `Music` 전역 테이블에 새 테이블을 추가합니다.

```
aws dynamodb update-global-table \
    --global-table-name Music \
    --replica-updates 'Create={RegionName=eu-west-1}' \
    --region us-east-2
```

5. 복제가 작동하는지 확인하려면 미국 동부(오하이오)의 `Music` 테이블에 새 항목을 추가합니다.

```
aws dynamodb put-item \
    --table-name Music \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

6. 몇 초 기다린 후 항목이 미국 동부(버지니아 북부) 및 유럽(아일랜드)에 성공적으로 복제되었는지 확인합니다.

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1
```

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

# 전역 테이블 모니터링

> **Important**
>
> 이 설명서는 버전 2017.11.29(레거시)의 글로벌 테이블에 대한 것이므로 새 글로벌 테이블의 경우 사용하지 않아야 합니다. 가능하면 글로벌 테이블 버전 2019.11.21(현재)을 사용해야 합니다. 이는 2017.11.29(레거시)보다 유연성과 효율성이 뛰어나고 쓰기 용량을 적게 소비합니다.
>
> 사용 중인 버전을 확인하려면 글로벌 테이블 버전 확인 섹션을 참조하세요. 기존 전역 테이블을 버전 2017.11.29(레거시)에서 버전 2019.11.21(현재)로 업데이트하는 경우 DynamoDB 글로벌 테이블 버전 섹션을 참조하세요.

Amazon CloudWatch를 사용하여 전역 테이블의 동작과 성능을 모니터링할 수 있습니다. Amazon DynamoDB는 전역 테이블의 각 복제본에 대해 `ReplicationLatency` 및 `PendingReplicationCount` 지표를 게시합니다.

- **`ReplicationLatency`** - 한 복제 테이블에 대한 DynamoDB Streams에 나타나는 업데이트된 항목과 전역 테이블의 다른 복제본에 나타나는 항목 간의 경과 시간입니다. `ReplicationLatency`는 밀리초 단위로 표현되며, 모든 원본 및 대상-리전 쌍에 대해 내보내집니다.

  정상 작동 중에는 `ReplicationLatency`가 상당히 일정해야 합니다. `ReplicationLatency` 값이 상승하면 한 복제본의 업데이트 내용이 다른 복제본 테이블로 시기 적절하게 전파되지 않는다는 것을 나타낼 수 있습니다. 시간이 지날수록 다른 복제본 테이블이 더 이상 지속적으로 업데이트 내용을 받지 않기 때문에 뒤처질 수 있습니다. 이 경우에는 각 복제본 테이블에 대해 읽기 용량 단위(RCU)와 쓰기 용량 단위(WCU)가 동일한지 확인해야 합니다. 또한 WCU 설정을 선택할 때 글로벌 테이블 버전의 권장 사항을 따라야 합니다.

  `ReplicationLatency`는 AWS 리전의 성능이 저하되고 해당 리전에 복제 테이블이 있는 경우에 증가할 수 있습니다. 이 경우 애플리케이션의 읽기 및 쓰기 작업을 다른 AWS 리전으로 일시적으로 리디렉션할 수 있습니다.

- **`PendingReplicationCount`** - 한 복제 테이블에 쓰여졌지만 전역 테이블의 다른 복제본에는 아직 쓰여지지 않은 항목 업데이트 수입니다. `PendingReplicationCount`는 항목 수로 표현되며, 모든 원본 및 대상-리전 쌍에 대해 내보내집니다.

  정상 작동 중에는 `PendingReplicationCount`가 매우 작아야 합니다. `PendingReplicationCount`가 오랜 시간 동안 증가하면 복제본 테이블의 프로비저닝된 쓰기 용량 설정이 현재 워크로드에 충분한지 여부를 조사해야 합니다.

`PendingReplicationCount`는 AWS 리전의 성능이 저하되고 해당 리전에 복제 테이블이 있는 경우에 증가할 수 있습니다. 이 경우 애플리케이션의 읽기 및 쓰기 작업을 다른 AWS 리전으로 일시적으로 리디렉션할 수 있습니다.

자세한 내용은 DynamoDB 지표 및 차원 섹션을 참조하세요.

## 전역 테이블에 IAM 사용

> **Important**
>
> 이 설명서는 버전 2017.11.29(레거시)의 글로벌 테이블에 대한 것이므로 새 글로벌 테이블의 경우 사용하지 않아야 합니다. 가능하면 글로벌 테이블 버전 2019.11.21(현재)을 사용해야 합니다. 이는 2017.11.29(레거시)보다 유연성과 효율성이 뛰어나고 쓰기 용량을 적게 소비합니다.
>
> 사용 중인 버전을 확인하려면 글로벌 테이블 버전 확인 섹션을 참조하세요. 기존 전역 테이블을 버전 2017.11.29(레거시)에서 버전 2019.11.21(현재)로 업데이트하는 경우 DynamoDB 글로벌 테이블 버전 섹션을 참조하세요.

전역 테이블을 처음으로 생성하는 경우, Amazon DynamoDB는 사용자를 위한 AWS Identity and Access Management(IAM) 서비스 연결 역할을 자동으로 생성합니다. 이름이 인 이 역할은 DynamoDB가 사용자를 대신하여 전역 테이블에 대한 교차 리전 복제를 관리하도록 허용합니다. 이 서비스 연결 역할을 삭제하지 마세요. 삭제하면 모든 전역 테이블이 더 이상 작동하지 않습니다.

서비스 연결 역할에 대한 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 사용을 참조하세요

DynamoDB에서 전역 테이블을 생성하고 관리하려면 다음의 각 항목에 액세스할 수 있는 `dynamodb:CreateGlobalTable` 권한이 있어야 합니다.

- 추가하려는 복제본 테이블.
- 이미 전역 테이블의 요소인 각각의 기존 복제본.
- 전역 테이블 자체.

DynamoDB의 전역 테이블에 대한 설정(`UpdateGlobalTableSettings`)을 업데이트하려면 `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`,

application-autoscaling:DeleteScalingPolicy 및 application-autoscaling:DeregisterScalableTarget 권한이 있어야 합니다.

기존 조정 정책을 업데이트할 때 application-autoscaling:DeleteScalingPolicy 및 application-autoscaling:DeregisterScalableTarget 권한이 필요합니다. 이렇게 하면 새로운 정책을 테이블이나 보조 인덱스에 연결하기 전에 전역 테이블 서비스가 기존의 조정 정책을 제거할 수 있습니다.

IAM 정책을 사용하여 하나의 복제 테이블에 대한 액세스 권한을 관리하는 경우, 해당 전역 테이블의 모든 다른 복제본에 동일한 정책을 적용해야 합니다. 이렇게 하면 모든 복제본 테이블에서 일관된 권한 모델을 유지할 수 있습니다.

전역 테이블의 모든 복제본에 동일한 IAM 정책을 사용함으로써, 전역 테이블 데이터에 대한 읽기 및 쓰기 액세스 권한을 의도치 않게 부여하는 것을 방지할 수도 있습니다. 예를 들어, 전역 테이블의 한 복제본에만 액세스할 수 있는 사용자를 고려합니다. 해당 사용자가 이 복제본에 쓸 수 있는 경우, DynamoDB는 다른 모든 복제 테이블에 쓰기를 전파합니다. 실제로 사용자는 전역 테이블의 모든 다른 복제본에 (간접적으로) 쓸 수 있습니다. 이 시나리오는 모든 복제본 테이블에 일관된 IAM 정책을 사용하여 방지할 수 있습니다.

### 예: CreateGlobalTable 작업 허용

전역 테이블에 복제본을 추가할 수 있으려면 전역 테이블과 해당하는 각 복제본 테이블에 대한 dynamodb:CreateGlobalTable 권한이 있어야 합니다.

다음 IAM 정책은 모든 테이블에서의 CreateGlobalTable 작업을 허용할 수 있는 권한을 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateGlobalTable"],
            "Resource": "*"
        }
    ]
}
```

## 예: UpdateGlobalTable, DescribeLimits, application-autoscaling:DeleteScalingPolicy 및 application-autoscaling:DeregisterScalableTarget 작업 허용

DynamoDB의 전역 테이블에 대한 설정(UpdateGlobalTableSettings)을 업데이트하려면 dynamodb:UpdateGlobalTable, dynamodb:DescribeLimits, application-autoscaling:DeleteScalingPolicy 및 application-autoscaling:DeregisterScalableTarget 권한이 있어야 합니다.

다음 IAM 정책은 모든 테이블에서의 UpdateGlobalTableSettings 작업을 허용할 수 있는 권한을 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateGlobalTable",
                "dynamodb:DescribeLimits",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget"
            ],
            "Resource": "*"
        }
    ]
}
```

## 예: 특정 리전에만 허용된 복제본이 있는 특정 전역 테이블 이름에 대한 CreateGlobalTable 작업 허용

다음 IAM 정책은 2개의 리전에 복제본이 있는 Customers라는 글로벌 테이블을 생성하는 CreateGlobalTable 작업을 허용하기 위해 권한을 부여합니다.

JSON

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:CreateGlobalTable",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Customers",
                "arn:aws:dynamodb:us-east-1:123456789012:table/Customers",
                "arn:aws:dynamodb:us-west-1:123456789012:table/Customers"
            ]
        }
    ]
}
```

# 이전 하위 수준 API 버전(2011-12-05)

이번 단원에서는 이전 DynamoDB 하위 수준 API 버전(2011-12-05)에서 사용할 수 있는 작업에 대해 살펴봅니다. 이 하위 수준 API 버전을 유지하는 이유는 기존 애플리케이션과의 역호환성을 지원하기 위해서입니다.

신규 애플리케이션은 현재 API 버전(2012-08-10)을 사용해야 합니다. 자세한 내용은 DynamoDB API 참조 섹션을 참조하세요.

> **Note**
>
> 애플리케이션은 최신 API 버전(2012-08-10)으로 마이그레이션하는 것이 바람직합니다. 새로운 DynamoDB 기능은 이전 API 버전으로 백포트(backport)가 지원되지 않기 때문입니다.

주제

- BatchGetItem
- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTables
- GetItem

- ListTables
- PutItem
- Query
- 스캔
- UpdateItem
- UpdateTable

## BatchGetItem

> **Important**
>
> **# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.**
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

### 설명

`BatchGetItem` 작업은 기본 키를 사용하여 여러 테이블에서 다수의 항목에 대한 속성을 반환합니다. 단일 작업에서 가져올 수 있는 최대 항목 수는 100개입니다. 또한 가져오는 항목 크기도 1MB로 제한됩니다. 응답 크기 제한을 벗어났거나, 테이블의 프로비저닝된 처리량을 초과하거나 내부 처리 오류가 발생하여 일부 결과만 반환되는 경우에는 DynamoDB가 `UnprocessedKeys` 값을 반환합니다. 그러면 다음 항목을 가져올 때 작업을 다시 시도할 수 있습니다. DynamoDB는 이러한 제한을 유지하기 위해 페이지당 반환되는 항목 수를 자동으로 조정합니다. 예를 들어 항목 100개를 가져오려고 해도 각 항목 크기가 50KB라면 시스템은 20개와 해당하는 `UnprocessedKeys` 값만 반환합니다. 따라서 다음 페이지에서 결과를 확인할 수 있습니다. 원한다면 애플리케이션에 자체 로직을 추가하여 결과 페이지를 단일 집합으로 어셈블할 수 있습니다.

요청에 포함된 각 테이블마다 프로비저닝 처리량이 부족하여 처리할 수 있는 항목이 없을 경우에는 DynamoDB가 `ProvisionedThroughputExceededException` 오류를 반환합니다.

> **Note**
>
> 기본적으로 `BatchGetItem`는 요청에 속한 모든 테이블에 최종적 일관된 읽기(Eventually Consistent Read)를 실행합니다. 그렇지 않고 consistent read를 원할 때는 테이블마다 `ConsistentRead` 파라미터를 `true`로 설정할 수 있습니다.
> `BatchGetItem` 은 항목을 병렬 방식으로 가져오기 때문에 응답 지연 시간을 최소화합니다.

애플리케이션을 설계할 때 DynamoDB는 반환되는 응답에서 속성의 순서 방식을 보장하지 않습니다. 따라서 항목 별로 응답을 구문 분석하려면 요청 항목마다 `AttributesToGet`에 기본 키 값을 추가해야 합니다.
요청된 항목이 존재하지 않으면 해당 항목 응답에서는 아무것도 반환되지 않습니다. 존재하지 않는 항목에 대한 요청은 읽기 형식에 따라 최소 읽기 용량 단위를 사용합니다. 자세한 내용은 DynamoDB 항목 크기 및 형식 단원을 참조하십시오.

## 요청

### 구문

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## ##
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{"RequestItems":
    {"Table1":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
    "Table2":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}}],
        "AttributesToGet": ["AttributeName4", "AttributeName5", "AttributeName6"]
        }
    }
}
```

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| `RequestItems` | 기본 키로 가져올 테이블 이름과 해당 항목의 컨테이너. 항목 | 예 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | 요청 시 각 테이블 이름은 작업 당 한 번만 불러올 수 있습니다.<br><br>유형: 문자열<br><br>기본값: None | |
| `Table` | 가져올 항목이 저장되어 있는 테이블 이름. 이 엔트리는 레이블이 없는 기존 테이블을 지정하는 단순 문자열입니다.<br><br>유형: 문자열<br><br>기본값: None | 예 |
| `Table:Keys` | 지정 테이블의 항목을 정의하는 기본 키 값. 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요.<br><br>형식: 키 | 예 |
| `Table:AttributesToGet` | 지정 테이블에 속한 속성 이름 배열. 속성 이름을 지정하지 않으면 모든 속성이 반환됩니다. 일부 속성을 찾을 수 없는 경우 결과에 표시되지 않습니다.<br><br>유형: 배열 | 아니요 |
| `Table:ConsistentRead` | `true`로 설정하면 consistent read가 발생하고, 그 밖의 경우에는 eventually consistent가 사용됩니다.<br><br>타입: 부울 | 아니요 |

# 응답

## 구문

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 855

{"Responses":
    {"Table1":
        {"Items":
        [{"AttributeName1": {"S":"AttributeValue"},
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue",
 "AttributeValue"]}
        }],
    "ConsumedCapacityUnits":1},
    "Table2":
        {"Items":
        [{"AttributeName1": {"S":"AttributeValue"},
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue","AttributeValue"]}
        }],
    "ConsumedCapacityUnits":1}
    },
    "UnprocessedKeys":
        {"Table3":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]}
        }
```

```
}
```

| 명칭 | 설명 |
| --- | --- |
| Responses | 테이블 이름과 테이블의 각 항목 속성.<br><br>유형: 맵 |
| Table | 항목이 저장되어 있는 테이블 이름. 이 엔트리는 레이블이 없는 테이블을 지정하는 단순 문자열입니다.<br><br>유형: 문자열 |
| Items | 작업 파라미터를 만족하는 속성 이름과 값이 저장되는 컨테이너.<br><br>형식: 속성 이름 맵과 속성의 데이터 형식 및 값. |
| ConsumedCapacityUnits | 각 테이블마다 사용되는 읽기 용량 단위 수. 이 값은 할당 처리량에 적용되는 수를 나타냅니다. 존재하지 않는 항목에 대한 요청은 읽기 형식에 따라 최소 읽기 용량 단위를 사용합니다. 자세한 정보는 DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요.<br><br>형식: 숫자 |
| UnprocessedKeys | 응답 크기 제한에 걸려 현재 응답에서 처리되지 않은 테이블 배열과 각 테이블 키가 저장됩니다. UnprocessedKeys 값은 RequestItems 파라미터와 동일한 형태가 됩니다(따라서 이 값을 이어지는 BatchGetItem 작업에 직접 입력해도 됩니다). 자세한 내용은 위의 RequestItems 파라미터를 참조하세요.<br><br>유형: 배열 |

| 명칭 | 설명 |
| --- | --- |
| `UnprocessedKeys : Table: Keys` | 항목, 그리고 항목과 연동된 속성을 정의하는 기본 키 속성 값. 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요.<br><br>형식: 속성 이름-값 페어의 배열. |
| `UnprocessedKeys : Table: AttributesToGet` | 지정 테이블에 속한 속성 이름. 속성 이름을 지정하지 않으면 모든 속성이 반환됩니다. 일부 속성을 찾을 수 없는 경우 결과에 표시되지 않습니다.<br><br>형식: 속성 이름 배열. |
| `UnprocessedKeys : Table: ConsistentRead` | `true`로 설정하면 지정 테이블에 consistent read가 사용되고, 그렇지 않으면 최종적 일관된 읽기(Eventually Consistent Read)가 사용됩니다.<br><br>유형: 부울. |

## 특수 오류

| Error | 설명 |
| --- | --- |
| `ProvisionedThroughputExceededException` | 할당이 허용되는 최대 처리량을 초과하였습니다. |

## 예시

다음은 BatchGetItem 작업을 사용해 HTTP POST 요청 및 응답을 나타낸 예제입니다. AWS SDK를 사용하는 예는 DynamoDB의 항목 및 속성 작업 단원을 참조하세요.

### 샘플 요청

다음은 두 가지 다른 테이블에서 속성을 요청하는 예제입니다.

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0
content-length: 409

{"RequestItems":
    {"comp1":
        {"Keys":
            [{"HashKeyElement":{"S":"Casey"},"RangeKeyElement":{"N":"1319509152"}},
            {"HashKeyElement":{"S":"Dave"},"RangeKeyElement":{"N":"1319509155"}},
            {"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"1319509158"}}],
        "AttributesToGet":["user","status"]},
    "comp2":
        {"Keys":
            [{"HashKeyElement":{"S":"Julie"}},{"HashKeyElement":{"S":"Mingus"}}],
        "AttributesToGet":["user","friends"]}
    }
}
```

### 샘플 응답

다음은 응답 예제입니다.

```
HTTP/1.1 200 OK
x-amzn-RequestId: GTPQVRM4VJS792J1UFJTKUBVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 373
Date: Fri, 02 Sep 2011 23:07:39 GMT

{"Responses":
    {"comp1":
        {"Items":
            [{"status":{"S":"online"},"user":{"S":"Casey"}},
            {"status":{"S":"working"},"user":{"S":"Riley"}},
            {"status":{"S":"running"},"user":{"S":"Dave"}}],
        "ConsumedCapacityUnits":1.5},
    "comp2":
        {"Items":
            [{"friends":{"SS":["Elisabeth", "Peter"]},"user":{"S":"Mingus"}},
            {"friends":{"SS":["Dave", "Peter"]},"user":{"S":"Julie"}}],
        "ConsumedCapacityUnits":1}
```

```
    },
    "UnprocessedKeys":{}
}
```

# BatchWriteItem

> **Important**
>
> **`# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.`**
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

## 설명

이 작업을 사용하면 단일 호출로 여러 테이블에서 여러 항목을 추가하거나 삭제할 수 있습니다.

하나의 항목을 업로드하려면 `PutItem`을 사용하고, 하나의 항목을 삭제하려면 `DeleteItem`을 사용할 수 있습니다. 그러나 대용량의 데이터를 업로드하거나 삭제하려는 경우(예: Amazon EMR에서 대용량의 데이터를 업로드하거나 다른 데이터베이스의 데이터를 DynamoDB로 마이그레이션) `BatchWriteItem`은 효과적인 대안을 제공합니다.

Java와 같은 언어를 사용하는 경우 스레드를 사용하여 항목을 동시에 업로드할 수 있습니다. 이는 애플리케이션의 스레드 처리를 복잡하게 합니다. 다른 언어는 스레딩을 지원하지 않습니다. 예를 들어 PHP를 사용하는 경우 한 번에 하나씩 항목을 업로드하거나 삭제해야 합니다. 두 경우 모두, `BatchWriteItem`은 지정된 추가 및 삭제 작업이 동시에 처리되는 대안을 제공하여 스레드 풀 접근 방식을 제공하므로 애플리케이션을 복잡하게 하지 않습니다.

`BatchWriteItem` 작업에서 지정된 각각의 개별적 추가 및 삭제는 소비되는 용량 단위 기준으로는 동일하다는 점에 유의하세요. 다만 `BatchWriteItem`이 지정된 작업을 동시에 수행하기 때문에 지연 시간은 줄어듭니다. 존재하지 않는 항목의 삭제 작업은 1 쓰기 용량 단위를 사용합니다. 사용되는 용량 단위에 대한 자세한 내용은 DynamoDB의 테이블 및 데이터 작업 단원을 참조하세요.

`BatchWriteItem`을 사용하는 경우 다음 제한에 유의하세요.

- 단일 요청에서 최대 작업 - 총 최대 25개의 추가 또는 삭제 작업을 지정할 수 있습니다. 그러나 총 요청 크기는 1MB(HTTP 페이로드)를 초과할 수 없습니다.
- 항목을 추가 및 삭제하는 경우에만 `BatchWriteItem` 작업을 사용할 수 있습니다. 이 작업을 사용하여 기존 항목을 업데이트할 수 없습니다.

- 원자성 작업이 아님 - BatchWriteItem에 지정된 개별 작업은 원자성입니다. 그러나 BatchWriteItem은 전체적으로 '최상의 노력' 작업이며 원자성 작업이 아닙니다. 다시 말해 BatchWriteItem 요청에서 일부 작업만 성공하고 다른 작업은 실패할 수 있습니다. 실패한 작업은 응답에서 UnprocessedItems 필드로 반환됩니다. 실패 원인 중 일부는 테이블에 대해 구성된 할당된 처리량을 초과했거나 네트워크 오류와 같은 일시적 실패 때문일 수 있습니다. 요청을 조사하거나 선택적으로 재전송할 수 있습니다. 일반적으로 BatchWriteItem을 반복적으로 호출하고 반복될 때마다 처리되지 않은 항목을 확인한 다음, 그러한 항목을 포함시킨 새 BatchWriteItem 요청을 제출합니다.
- 항목을 반환하지 않음 - BatchWriteItem은 대용량의 데이터를 효율적으로 업로드하기 위한 것입니다. 그러나 PutItem 및 DeleteItem에서 지원하는 일부 고급 작업을 제공하지 않습니다. 예를 들어 DeleteItem은 요청 본문의 ReturnValues 필드를 지원하여 응답에서 삭제된 항목을 요청합니다. BatchWriteItem 작업은 응답에 항목을 반환하지 않습니다.
- PutItem 및 DeleteItem과는 달리 BatchWriteItem은 작업의 개별 쓰기 요청에 조건을 지정하도록 허용하지 않습니다.
- 속성 값은 null이 될 수 없습니다. 문자열과 이진 형식 속성 길이는 0보다 커야 합니다. 설정 유형 속성은 비어 있으면 안 됩니다. 값이 비어 있는 요청은 ValidationException으로 거부됩니다.

DynamoDB는 다음 중 하나가 true일 경우 전체 일괄 쓰기 작업을 거부합니다.

- BatchWriteItem 요청에 지정된 하나 이상의 테이블이 존재하지 않는 경우.
- 요청의 항목에 지정된 기본 키 속성이 해당하는 테이블의 기본 키 스키마와 일치하지 않는 경우.
- 동일한 BatchWriteItem 요청에서 동일한 항목에 대해 여러 작업을 수행하려는 경우. 예를 들어 동일한 BatchWriteItem 요청에서 동일한 항목을 추가 및 삭제할 수 없습니다.
- 총 요청 크기가 1MB 요청 크기(HTTP 페이로드) 제한을 초과하는 경우.
- 일괄의 개별 항목이 64KB 항목 크기 제한을 초과하는 경우.

## 요청

### 구문

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## ##
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0
```

```
{
  "RequestItems" :  RequestItems
}

RequestItems
{
     "TableName1" :  [ Request, Request, ... ],
     "TableName2" :  [ Request, Request, ... ],
     ...
}

Request ::=
  PutRequest | DeleteRequest

PutRequest ::=
{
  "PutRequest" : {
     "Item" : {
        "Attribute-Name1" : Attribute-Value,
        "Attribute-Name2" : Attribute-Value,
         ...
     }
  }
}

DeleteRequest ::=
{
   "DeleteRequest" : {
      "Key" : PrimaryKey-Value
   }
}


PrimaryKey-Value ::= HashTypePK | HashAndRangeTypePK

HashTypePK ::=
{
   "HashKeyElement" : Attribute-Value
}

HashAndRangeTypePK
{
   "HashKeyElement" : Attribute-Value,
   "RangeKeyElement" : Attribute-Value,
```

```
}

Attribute-Value ::= String | Numeric| Binary | StringSet | NumericSet | BinarySet

Numeric ::=
{
   "N": "Number"
}

String ::=
{
   "S": "String"
}

Binary ::=
{
    "B": "Base64 encoded binary data"
}

StringSet ::=
{
   "SS": [ "String1", "String2", ... ]
}

NumberSet ::=
{
   "NS": [ "Number1", "Number2", ... ]
}

BinarySet ::=
{
   "BS": [ "Binary1", "Binary2", ... ]
}
```

요청 본문에서 RequestItems JSON 객체는 수행할 작업을 설명합니다. 작업은 테이블별로 그룹화됩니다. BatchWriteItem을 사용하여 여러 테이블에서 여러 항목을 업데이트하거나 삭제할 수 있습니다. 각 쓰기 요청별로 작업에 대한 세부 정보 앞에 있는 요청 유형(PutItem, DeleteItem)을 확인해야 합니다.

- PutRequest의 경우, 항목, 즉 속성 및 해당 값 목록을 제공합니다.
- DeleteRequest의 경우, 기본 키 이름과 값을 제공합니다.

## 응답

### 구문

다음은 응답에서 반환한 JSON 본문의 구문입니다.

```
{
  "Responses" :         ConsumedCapacityUnitsByTable
  "UnprocessedItems" :  RequestItems
}

ConsumedCapacityUnitsByTable
{
    "TableName1" : { "ConsumedCapacityUnits", : NumericValue },
    "TableName2" : { "ConsumedCapacityUnits", : NumericValue },
     ...
}

RequestItems
This syntax is identical to the one described in the JSON syntax in the request.
```

### 특수 오류

이 작업에는 특정 오류가 없습니다.

### 예시

다음 예에서는 HTTP POST 요청 및 `BatchWriteItem` 작업의 응답을 보여 줍니다. 이 요청은 Reply 및 Thread 테이블에서 다음 작업을 지정합니다.

- Reply 테이블의 항목 추가 및 삭제
- Thread 테이블로 항목 추가

AWS SDK를 사용하는 예는 DynamoDB의 항목 및 속성 작업 단원을 참조하세요.

### 샘플 요청

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## ##
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0
```

```
{
  "RequestItems":{
    "Reply":[
      {
        "PutRequest":{
          "Item":{
            "ReplyDateTime":{
              "S":"2012-04-03T11:04:47.034Z"
            },
            "Id":{
              "S":"DynamoDB#DynamoDB Thread 5"
            }
          }
        }
      },
      {
        "DeleteRequest":{
          "Key":{
            "HashKeyElement":{
              "S":"DynamoDB#DynamoDB Thread 4"
            },
            "RangeKeyElement":{
              "S":"oops - accidental row"
            }
          }
        }
      }
    ],
    "Thread":[
      {
        "PutRequest":{
          "Item":{
            "ForumName":{
              "S":"DynamoDB"
            },
            "Subject":{
              "S":"DynamoDB Thread 5"
            }
          }
        }
      }
    ]
  }
```

```
}
```

샘플 응답

다음 응답 예에서는 Thread 및 Reply 테이블에서 성공한 추가 작업 및 Reply 테이블에서 (테이블에서 할당된 처리량을 초과하는 경우 발생하는 병목 현상 등의 이유로) 실패한 삭제 작업을 보여 줍니다. JSON 응답에서 다음을 참고하세요.

- Responses 객체는 Thread 및 Reply 테이블에서 성공적인 추가 작업의 결과로 이 두 테이블에서 1 용량 단위가 사용되었다는 것을 보여 줍니다.
- UnprocessedItems 객체는 Reply 테이블에서 성공하지 못한 삭제 작업을 보여 줍니다. 새 BatchWriteItem 호출을 발행하여 처리되지 않은 이러한 요청을 처리할 수 있습니다.

```
HTTP/1.1 200 OK
x-amzn-RequestId: G8M9ANLOE5QA26AEUHJKJE0ASBVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 536
Date: Thu, 05 Apr 2012 18:22:09 GMT

{
   "Responses":{
      "Thread":{
         "ConsumedCapacityUnits":1.0
      },
      "Reply":{
         "ConsumedCapacityUnits":1.0
      }
   },
   "UnprocessedItems":{
      "Reply":[
         {
            "DeleteRequest":{
               "Key":{
                  "HashKeyElement":{
                     "S":"DynamoDB#DynamoDB Thread 4"
                  },
                  "RangeKeyElement":{
                     "S":"oops - accidental row"
                  }
               }
            }
```

```
        }
      ]
   }
}
```

# CreateTable

> **Important**
>
> ***# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.***
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

## 설명

`CreateTable` 작업은 사용자 계정에 새 테이블을 추가합니다.

이 테이블 이름은 요청을 실행하는 AWS 계정 및 요청을 수신하는 AWS 리전(예: dynamodb.us-west-2.amazonaws.com)과 연동된 테이블 이름과 달라야 합니다. 각 DynamoDB 엔드포인트는 완전히 독립적입니다. 예를 들어, dynamodb.us-west-2.amazonaws.com 및 dynamodb.us-west-1.amazonaws.com에 "MyTable"이라는 테이블이 각각 있는 경우 해당 테이블은 완전히 독립적이며 데이터를 공유하지 않습니다.

`CreateTable` 작업은 비동기 워크플로를 트리거하여 테이블 만들기를 시작합니다. DynamoDB는 테이블이 `ACTIVE` 상태가 될 때까지 테이블 상태(`CREATING`)를 즉시 반환합니다. 테이블이 `ACTIVE` 상태인 경우 데이터 플레인 작업을 수행할 수 있습니다.

DescribeTables 작업을 사용하여 테이블 상태를 확인합니다.

## 요청

### 구문

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "KeySchema":
```

```
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| TableName | 생성할 테이블 이름.<br><br>허용되는 문자는 a~z, A~Z, 0~9, '_'(밑줄), '-'(대시) 및 '.'(점)입니다. 이름에 포함되는 문자 길이는 3~255자입니다.<br><br>유형: 문자열 | 예 |
| KeySchema | 테이블의 기본 키(단순 또는 복합) 구조. HashKeyElement 에서는 이름-값 페어가 필수이지만 RangeKeyElement 에서는 선택 사항입니다(복합 기본 키의 경우에만 필요함). 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요.<br><br>기본 키 요소 이름은 1~255자로 지정할 수 있으며 문자에는 제한이 없습니다.<br><br>AttributeType에 대해 가능한 값은 "S"(문자열), "N"(숫자) 또는 "B"(이진수)입니다.<br><br>형식: HashKeyElement 맵, 또는 복합 기본 키의 HashKeyElement 및 RangeKeyElement | 예 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| `ProvisionedThroughput` | 지정 테이블의 새로운 처리량으로서 `ReadCapacityUnits` 값과 `WriteCapacityUnits` 값으로 구성됩니다. 세부 정보는 DynamoDB 프로비저닝된 용량 모드을 참조하세요.<br><br>**Note**<br>현재 최대값/최소값은 Amazon DynamoDB의 할당량를 참조하세요.<br><br>유형: 배열 | 예 |
| `ProvisionedThroughput : ReadCapacityUnits` | DynamoDB가 다른 작업과 로드 밸런스를 맞출 때까지 지정 테이블에서 사용되는 consistent `ReadCapacityUnits` 의 초당 최소 수를 설정합니다.<br><br>Eventually consistent read 작업은 consistent read 작업에 비해 필요한 부하가 적습니다. 따라서 초당 consistent `ReadCapacityUnits` 를 50으로 설정하면 초당 eventually consistent `ReadCapacityUnits` 는 100이 됩니다.<br><br>형식: 숫자 | 예 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| ProvisionedThroughput : WriteCapacityUnits | DynamoDB가 다른 작업과 로드 밸런스를 맞출 때까지 지정 테이블에서 사용되는 WriteCapacityUnits 의 초당 최소 수를 설정합니다.<br><br>형식: 숫자 | 예 |

## 응답

### 구문

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"CREATING"
    }
}
```

| 명칭 | 설명 |
| --- | --- |
| TableDescription | 테이블 속성에 대한 컨테이너 |
| CreationDateTime | UNIX epoch 시간 형식의 테이블 생성 날짜<br><br>형식: 숫자 |

| 명칭 | 설명 |
| --- | --- |
| KeySchema | 테이블의 기본 키(단순 또는 복합) 구조. HashKeyElement 에서는 이름-값 페어가 필수이지만 RangeKeyElement 에서는 선택 사항입니다(복합 기본 키의 경우에만 필요함). 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요.<br><br>형식: HashKeyElement 맵, 또는 복합 기본 키의 HashKeyElement 및 RangeKeyElement |
| ProvisionedThroughput | 지정 테이블의 처리량으로서 ReadCapacityUnits 값과 WriteCapacityUnits 값으로 구성됩니다. DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요.<br><br>유형: 배열 |
| ProvisionedThroughput :ReadCapacityUnits | DynamoDB가 다른 작업과 로드 밸런스를 맞출 때까지 사용되는 ReadCapacityUnits 의 초당 최소 수입니다.<br><br>형식: 숫자 |
| ProvisionedThroughput :WriteCapacityUnits | WriteCapacityUnits 가 다른 작업과 로드 밸런스를 맞출 때까지 사용되는 ReadCapacityUnits 의 초당 최소 수입니다.<br><br>형식: 숫자 |
| TableName | 생성된 테이블 이름<br><br>유형: 문자열 |

| 명칭 | 설명 |
| --- | --- |
| TableStatus | 현재 테이블 상태(CREATING) 테이블이 ACTIVE 상태일 때만 데이터를 입력할 수 있습니다.<br><br>DescribeTables API를 사용하여 테이블 상태를 확인합니다.<br><br>유형: 문자열 |

## 특수 오류

| Error | 설명 |
| --- | --- |
| ResourceInUseException | 기존 테이블을 다시 만들려고 시도합니다. |
| LimitExceededException | 동시 테이블 요청 수(CREATING, DELETING 또는 UPDATING 상태인 테이블의 누적 수)가 허용된 최대값을 초과합니다.<br><br>Note<br>현재 최대값/최소값은 Amazon DynamoDB의 할당량를 참조하세요.<br><br>. |

## 예시

다음 예에서는 문자열과 숫자를 포함하는 복합 기본 키가 있는 테이블을 만듭니다. AWS SDK를 사용하는 예는 DynamoDB의 테이블 및 데이터 작업 단원을 참조하세요.

### 샘플 요청

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
```

```
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0


{"TableName":"comp-table",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

### 샘플 응답

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"comp-table",
    "TableStatus":"CREATING"
    }
}
```

### 관련 작업

- DescribeTables
- DeleteTable

## DeleteItem

> **Important**
>
> **# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.**

현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

## 설명

기본 키로 테이블의 단일 항목을 삭제합니다. 항목이 있거나 항목에 예상 속성 값이 있을 경우 삭제하는 조건부 삭제 작업을 수행할 수 있습니다.

> **Note**
>
> 속성이나 값 없이 `DeleteItem`을 지정하면 항목의 모든 속성이 삭제됩니다.
> 조건을 지정하지 않으면 `DeleteItem`은 idempotent 작업이 됩니다. 동일한 항목이나 속성에서 여러 번 실행할 경우 오류 응답이 발생하지 않습니다.
> 특정 조건이 충족될 경우 항목과 속성을 삭제할 때만 조건부 삭제가 유용합니다. 조건이 충족되면 DynamoDB에서 삭제를 수행합니다. 그렇지 않으면 항목이 삭제되지 않습니다.
> 작업마다 한 속성에 대해 예상되는 조건부 검사를 수행할 수 있습니다.

## 요청

### 구문

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},"RangeKeyElement":
{"N":"AttributeValue2"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3"}}},
    "ReturnValues":"ALL_OLD"}
}
```

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| `TableName` | 삭제할 항목이 포함된 테이블 이름입니다. | 예 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | 유형: 문자열 | |
| Key | 항목을 정의하는 기본 키입니다. 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요.<br><br>형식: 해당 값에 대한 HashKeyElement 및 해당 값에 대한 RangeKeyElement 의 맵 | 예 |
| Expected | 조건부 삭제의 속성을 지정합니다. Expected 파라미터를 사용하면 속성 이름을 제공하고, DynamoDB에서 속성을 삭제하기 전에 속성에 특정 값이 있는지 확인할지 여부를 지정할 수 있습니다.<br><br>형식: 속성 이름의 맵 | 아니요 |
| Expected:AttributeName | 조건부 입력에 대한 속성 이름입니다.<br><br>유형: 문자열 | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| `Expected:AttributeName: ExpectedAttributeValue` | 이 파라미터를 사용하여 속성 이름-값 페어의 값이 존재할지 여부를 지정합니다.<br><br>다음 JSON 표기법은 항목의 "Color" 속성이 없을 경우 해당 항목을 삭제합니다.<br><br>`"Expected" : {"Color":{"Exists":false}}`<br><br>다음 JSON 표기법은 항목을 삭제하기 전에 이름이 "Color"인 속성의 기존 값이 "Yellow"인지 여부를 확인합니다.<br><br>`"Expected" : {"Color":{"Exists":true},{"Value":{"S":"Yellow"}}}`<br><br>기본적으로 `Expected` 파라미터를 사용하고 `Value`를 제공하는 경우 DynamoDB는 속성이 존재하고 바꿀 현재 값이 있는 것으로 가정합니다. 따라서 `{"Exists":true}` 가 내재되어 있으므로 지정하지 않아도 됩니다. 요청을 다음과 같이 줄일 수 있습니다.<br><br>`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | **Note**<br>확인할 속성 값이 없을 때 `{"Exists":true}` 를 지정하면 DynamoDB는 오류를 반환합니다. | |
| `ReturnValues` | 속성 이름-값 페어가 삭제되기 전에 이를 가져오려면 이 파라미터를 사용합니다. 가능한 파라미터 값은 `NONE`(기본) 또는 `ALL_OLD`입니다. `ALL_OLD`가 지정되면 이전 항목의 내용이 반환됩니다. 이 파라미터가 제공되지 않거나 `NONE`인 경우, 아무 것도 반환되지 않습니다.<br><br>유형: 문자열 | 아니요 |

## 응답

### 구문

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 353
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"Attributes":
    {"AttributeName3":{"SS":["AttributeValue3","AttributeValue4","AttributeValue5"]},
    "AttributeName2":{"S":"AttributeValue2"},
    "AttributeName1":{"N":"AttributeValue1"}
    },
"ConsumedCapacityUnits":1
```

```
}
```

| 명칭 | 설명 |
| --- | --- |
| `Attributes` | `ReturnValues` 파라미터가 `ALL_OLD`로 요청에 제공되면 DynamoDB가 속성 이름-값 페어의 배열을 반환합니다(특히 삭제된 항목). 그렇지 않으면 응답에 빈 세트가 포함됩니다.<br><br>형식: 속성 이름-값 페어의 배열. |
| `ConsumedCapacityUnits` | 작업에서 사용한 쓰기 용량 단위의 수입니다. 이 값은 할당 처리량에 적용되는 수를 나타냅니다. 존재하지 않는 항목의 삭제 요청은 1 쓰기 용량 단위를 사용합니다. 자세한 정보는 DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요.<br><br>형식: 숫자 |

## 특수 오류

| Error | 설명 |
| --- | --- |
| `ConditionalCheckFailedException` | 조건부 확인이 실패했습니다. 예상 속성 값이 검색되지 않았습니다. |

## 예

### 샘플 요청

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"comp-table",
```

```
    "Key":
        {"HashKeyElement":{"S":"Mingus"},"RangeKeyElement":{"N":"200"}},
    "Expected":
        {"status":{"Value":{"S":"shopping"}}},
    "ReturnValues":"ALL_OLD"
}
```

### 샘플 응답

```
HTTP/1.1 200 OK
x-amzn-RequestId: U9809LI6BBFJA5N2R0TB0P017JVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 353
Date: Tue, 12 Jul 2011 22:31:23 GMT

{"Attributes":
    {"friends":{"SS":["Dooley","Ben","Daisy"]},
    "status":{"S":"shopping"},
    "time":{"N":"200"},
    "user":{"S":"Mingus"}
    },
"ConsumedCapacityUnits":1
}
```

### 관련 작업

- PutItem

## DeleteTable

> **Important**
>
> ***# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.***
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

### 설명

DeleteTable 작업은 테이블과 테이블에 속한 모든 항목을 삭제합니다. DeleteTable 요청 이후 DynamoDB가 삭제를 완료할 때까지 지정한 테이블은 DELETING 상태가 됩니다. 테이블을 삭제하

려면 해당 테이블 상태가 ACTIVE이어야 합니다. 테이블이 CREATING 또는 UPDATING 상태인 경우에는 DynamoDB가 ResourceInUseException 오류를 반환합니다. 그리고 지정한 테이블이 존재하지 않는 경우에는 DynamoDB가 ResourceNotFoundException을 반환합니다. 테이블이 이미 DELETING 상태라면 어떠한 오류도 반환되지 않습니다.

> **Note**
>
> DynamoDB는 테이블 삭제가 완료될 때까지는 DELETING 상태인 테이블에서 GetItem, PutItem 등의 데이터 플레인 작업 요청을 계속 허용할 수 있습니다.

테이블은 요청을 실행하는 AWS 계정 및 요청을 수신하는 AWS 리전(예: dynamodb.us-west-1.amazonaws.com)과 관련된 테이블에서 고유해야 합니다. 각 DynamoDB 엔드포인트는 완전히 독립적입니다. 예를 들어, dynamodb.us-west-2.amazonaws.com 및 dynamodb.us-west-1.amazonaws.com에 "MyTable"이라는 테이블이 각각 있는 경우 해당 테이블은 완전히 독립적이며 데이터를 공유하지 않습니다. 즉, 한 테이블을 삭제해도 다른 테이블이 삭제되지 않습니다.

DescribeTables 작업을 사용하여 테이블 상태를 확인합니다.

## 요청

### 구문

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| TableName | 삭제할 테이블 이름<br><br>유형: 문자열 | 예 |

# 응답

## 구문

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Sun, 14 Aug 2011 22:56:22 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"DELETING"
    }
}
```

| 명칭 | 설명 |
| --- | --- |
| TableDescription | 테이블 속성에 대한 컨테이너 |
| CreationDateTime | 테이블 생성 날짜<br><br>형식: 숫자 |
| KeySchema | 테이블의 기본 키(단순 또는 복합) 구조. HashKeyElement 에서는 이름-값 페어가 필수이지만 RangeKeyElement 에서는 선택 사항입니다(복합 기본 키의 경우에만 필요함). 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요.<br><br>형식: HashKeyElement 맵, 또는 복합 기본 키의 HashKeyElement 및 RangeKeyElement |

| 명칭 | 설명 |
| --- | --- |
| ProvisionedThroughput | 지정 테이블의 처리량으로서 ReadCapacityUnits 값과 WriteCapacityUnits 값으로 구성됩니다. DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요. |
| ProvisionedThroughput : ReadCapacityUnits | DynamoDB가 다른 작업과 로드 밸런스를 맞출 때까지 지정 테이블에서 사용되는 ReadCapacityUnits 의 초당 최소 수입니다.<br><br>형식: 숫자 |
| ProvisionedThroughput : WriteCapacityUnits | DynamoDB가 다른 작업과 로드 밸런스를 맞출 때까지 지정 테이블에서 사용되는 WriteCapacityUnits 의 초당 최소 수입니다.<br><br>형식: 숫자 |
| TableName | 삭제된 테이블 이름<br><br>유형: 문자열 |
| TableStatus | 현재 테이블 상태(DELETING) 이후 삭제된 테이블에 대해 요청을 실행하면 resource not found가 반환됩니다.<br><br>DescribeTables 작업을 사용하여 테이블 상태를 확인합니다.<br><br>유형: 문자열 |

## 특수 오류

| Error | 설명 |
| --- | --- |
| ResourceInUseException | 테이블이 CREATING 또는 UPDATING 상태이면 삭제할 수 없습니다. |

## 예

### 샘플 요청

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## ##
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0
content-length: 40

{"TableName":"favorite-movies-table"}
```

### 샘플 응답

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 160
Date: Sun, 14 Aug 2011 17:20:03 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"name","AttributeType":"S"}},
    "TableName":"favorite-movies-table",
    "TableStatus":"DELETING"
}
```

### 관련 작업

- CreateTable
- DescribeTables

# DescribeTables

> **Important**
>
> ***# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.***
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

## 설명

현재 테이블 상태, 기본 키 스키마, 그리고 테이블 생성 날짜 등 테이블 관련 정보를 반환합니다. DescribeTable 결과는 최종 일관성을 따릅니다. 테이블 생성 프로세스에서 너무 일찍 DescribeTable을 사용하면 DynamoDB가 `ResourceNotFoundException`을 반환합니다. 그리고, 테이블 업데이트 프로세스에서 너무 일찍 DescribeTable을 사용할 때도 새로운 값이 바로 적용되지 않을 수도 있습니다.

## 요청

### 구문

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| `TableName` | 설명할 테이블 이름<br><br>유형: 문자열 | 예 |

## 응답

### 구문

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
Content-Length: 543


{"Table":
    {"CreationDateTime":1.309988345372E9,
    ItemCount:1,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
```

```
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": Date, "LastDecreaseDateTime":
 Date, "ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableSizeBytes":1,
    "TableStatus":"ACTIVE"
    }
}
```

| 명칭 | 설명 |
| --- | --- |
| Table | 설명할 테이블이 저장되는 컨테이너<br><br>유형: 문자열 |
| CreationDateTime | UNIX epoch 시간 형식의 테이블 생성 날짜 |
| ItemCount | 지정된 테이블의 항목 수. DynamoDB는 약 6시간마다 이 값을 업데이트합니다. 최근 변경 사항이 이 값에 반영되지 않기도 합니다.<br><br>형식: 숫자 |
| KeySchema | 테이블의 기본 키(단순 또는 복합) 구조. HashKeyElement 에서는 이름-값 페어가 필수이지만 RangeKeyElement 에서는 선택 사항입니다(복합 기본 키의 경우에만 필요함). 최대 해시 키 크기는 2,048byte입니다. 최대 범위 키 크기는 1,024byte입니다. 두 키의 크기 제한은 별도로 적용됩니다(해시 + 범위 키 2,048 + 1,024 합산). 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요. |
| ProvisionedThroughput | 지정 테이블의 처리량으로 LastIncreaseDateTime 값(있는 경우), LastDecreaseDateTime 값(있는 경우), ReadCapacityUnits 값 및 WriteCapacityUnits 값으로 구성됩니다. 테이블의 처리량이 늘거나 줄지 않으면 DynamoDB도 해당 요소의 값을 반 |

| 명칭 | 설명 |
| --- | --- |
| | 환하지 않습니다. DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요.<br><br>유형: 배열 |
| TableName | 요청한 테이블 이름<br><br>유형: 문자열 |
| TableSizeBytes | 지정된 테이블의 총 크기(바이트). DynamoDB는 약 6시간마다 이 값을 업데이트합니다. 최근 변경 사항이 이 값에 반영되지 않기도 합니다.<br><br>형식: 숫자 |
| TableStatus | 현재 테이블 상태(CREATING, ACTIVE, DELETING 또는 UPDATING). 테이블이 ACTIVE 상태일 때만 데이터를 추가할 수 있습니다. |

## 특수 오류

이 작업에는 특정 오류가 없습니다.

## 예시

다음은 "comp-table"이라는 이름의 테이블에 대해 DescribeTable 작업을 사용해 HTTP POST 요청 및 응답을 나타낸 예제입니다. 테이블은 복합 기본 키를 갖고 있습니다.

샘플 요청

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"users"}
```

### 샘플 응답

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 543

{"Table":
    {"CreationDateTime":1.309988345372E9,
    "ItemCount":23,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": 1.309988345384E9,
 "ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"users",
    "TableSizeBytes":949,
    "TableStatus":"ACTIVE"
    }
}
```

### 관련 작업

- CreateTable
- DeleteTable
- ListTables

## GetItem

> **Important**
>
> ***# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.***
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

### 설명

`GetItem` 작업은 기본 키와 일치하는 항목에 대한 `Attributes` 세트를 반환합니다. 일치하는 항목이 없으면 `GetItem`이 데이터를 반환하지 않습니다.

GetItem 작업은 기본적으로 최종적 일관된 읽기(Eventually Consistent Read)를 제공합니다. 애플리케이션에서 최종적 일관된 읽기(Eventually Consistent Read)를 사용할 수 없는 경우 ConsistentRead를 사용하세요. 이 작업은 표준 읽기보다 오래 걸릴 수 있지만 항상 마지막으로 업데이트된 값을 반환합니다. 자세한 내용은 DynamoDB 읽기 일관성 단원을 참조하십시오.

## 요청

### 구문

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
  "Key":
  {"HashKeyElement": {"S":"AttributeValue1"},
  "RangeKeyElement": {"N":"AttributeValue2"}
 },
 "AttributesToGet":["AttributeName3","AttributeName4"],
 "ConsistentRead":Boolean
}
```

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| TableName | 요청된 항목을 포함하는 테이블의 이름입니다.<br><br>유형: 문자열 | 예 |
| Key | 항목을 정의하는 기본 키 값입니다. 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요.<br><br>형식: 해당 값에 대한 HashKeyElement 및 해당 값에 대한 RangeKeyElement 의 맵 | 예 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| AttributesToGet | 속성 이름의 배열입니다. 속성 이름을 지정하지 않으면 모든 속성이 반환됩니다. 일부 속성을 찾을 수 없는 경우 결과에 표시되지 않습니다.<br><br>유형: 배열 | 아니요 |
| ConsistentRead | true로 설정하면 consistent read가 발생하고, 그 밖의 경우에는 eventually consistent가 사용됩니다.<br><br>타입: 부울 | 아니요 |

## 응답

### 구문

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 144

{"Item":{
 "AttributeName3":{"S":"AttributeValue3"},
 "AttributeName4":{"N":"AttributeValue4"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits": 0.5
}
```

| 명칭 | 설명 |
| --- | --- |
| Item | 요청된 속성을 포함합니다.<br><br>형식: 속성 이름-값 페어의 맵 |

| 명칭 | 설명 |
| --- | --- |
| ConsumedCapacityUnits | 작업에 사용된 읽기 용량 단위의 수. 이 값은 할당 처리량에 적용되는 수를 나타냅니다. 존재하지 않는 항목에 대한 요청은 읽기 형식에 따라 최소 읽기 용량 단위를 사용합니다. 자세한 정보는 DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요.<br><br>형식: 숫자 |

## 특수 오류

이 작업에는 특정 오류가 없습니다.

## 예시

AWS SDK를 사용하는 예는 DynamoDB의 항목 및 속성 작업 단원을 참조하세요.

### 샘플 요청

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"comptable",
 "Key":
  {"HashKeyElement":{"S":"Julie"},
  "RangeKeyElement":{"N":"1307654345"}},
 "AttributesToGet":["status","friends"],
 "ConsistentRead":true
}
```

### 샘플 응답

선택적 파라미터 ConsistentRead가 true로 설정되었으므로 ConsumedCapacityUnits 값은 1입니다. 같은 요청에 대해 ConsistentRead가 false로 설정되거나 지정되지 않으면 응답이 eventually consistent이고 ConsumedCapacityUnits 값은 0.5입니다.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 72

{"Item":
 {"friends":{"SS":["Lynda, Aaron"]},
 "status":{"S":"online"}
 },
"ConsumedCapacityUnits": 1
}
```

# ListTables

> **Important**
>
> ***# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.***
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

## 설명

현재 계정 및 엔드포인트에 연동되어 있는 테이블을 모두 배열 형식으로 반환합니다.

각 DynamoDB 엔드포인트는 완전히 독립적입니다. 예를 들어, dynamodb.us-west-2.amazonaws.com 및 dynamodb.us-east-1.amazonaws.com에 "MyTable"이라는 테이블이 각각 있는 경우 해당 테이블은 완전히 독립적이며 데이터를 공유하지 않습니다. ListTables 작업은 요청 계정과 연동되어 있는 테이블 이름을 비롯해 요청을 받는 엔드포인트와 연동되어 있는 테이블 이름까지 모두 반환합니다.

## 요청

### 구문

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"Table1","Limit":3}
```

ListTables 작업은 기본적으로 요청 계정과 연동되어 있는 테이블 이름을 비롯해 요청을 받는 엔드포인트와 연동되어 있는 테이블 이름까지 모두 요청합니다.

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| Limit | 반환할 수 있는 테이블 이름의 최대 수<br><br>유형: 정수 | 아니요 |
| ExclusiveStartTableName | 목록에 나열되는 첫 번째 테이블 이름. 이미 앞서서 ListTables 작업을 실행하여 응답으로 LastEvaluatedTableName 값을 가져온 경우에는 여기 값을 사용하여 목록을 계속 나열하면 됩니다.<br><br>유형: 문자열 | 아니요 |

## 응답

### 구문

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"TableNames":["Table1","Table2","Table3"], "LastEvaluatedTableName":"Table3"}
```

| 명칭 | 설명 |
| --- | --- |
| TableNames | 현재 엔드포인트에서 현재 계정과 연동되어 있는 테이블 이름<br><br>유형: 배열 |

| 명칭 | 설명 |
| --- | --- |
| LastEvaluatedTableName | 현재 목록에서 마지막 테이블 이름. 계정 및 엔드포인트에서 일부 테이블이 반환되지 않은 경우에 한함. 이미 모든 테이블 이름이 반환된 경우에는 이 값이 응답에 표시되지 않습니다. 이 값은 모든 테이블 이름이 반환되지 않아서 목록을 계속 나열할 때 새로운 요청의 ExclusiveStartTableName 으로 사용됩니다.<br><br>유형: 문자열 |

## 특수 오류

이 작업에는 특정 오류가 없습니다.

## 예시

다음은 ListTables 작업을 사용해 HTTP POST 요청 및 응답을 나타낸 예제입니다.

### 샘플 요청

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"comp2","Limit":3}
```

### 샘플 응답

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"LastEvaluatedTableName":"comp5","TableNames":["comp3","comp4","comp5"]}
```

### 관련 작업

- DescribeTables
- CreateTable
- DeleteTable

## PutItem

> **Important**
>
> ***# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.***
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

### 설명

새 항목을 만들거나 이전 항목을 새 항목으로 바꿉니다(모든 속성 포함). 지정된 테이블에 동일한 기본 키를 지닌 항목이 이미 존재하는 경우 기존 항목이 새 항목으로 완전히 바뀝니다. 조건부 입력을 수행하거나(기본 키가 지정된 항목이 없는 경우 새 항목을 삽입합니다) 기존 항목에 특정 속성 값이 있는 경우 해당 항목을 교체할 수 있습니다.

속성 값은 null이 될 수 없습니다. 문자열과 이진 형식 속성 길이는 0보다 커야 합니다. 설정 유형 속성은 비어 있으면 안 됩니다. 값이 비어 있는 요청은 `ValidationException`으로 거부됩니다.

> **Note**
>
> 새 항목이 기존의 항목을 대체하지 않도록 하려면 기본 키 속성 또는 속성에 대해 `Exists`로 설정된 `false`을 사용하여 조건부 입력 작업을 수행합니다.

`PutItem` 사용에 관한 자세한 내용은 DynamoDB의 항목 및 속성 작업 단원을 참조하세요.

### 요청

#### 구문

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
```

```
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Item":{
        "AttributeName1":{"S":"AttributeValue1"},
        "AttributeName2":{"N":"AttributeValue2"},
        "AttributeName5":{"B":"dmFsdWU="}
    },
    "Expected":{"AttributeName3":{"Value": {"S":"AttributeValue"}, "Exists":Boolean}},
    "ReturnValues":"ReturnValuesConstant"}
```

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| TableName | 항목을 저장하기 위한 테이블의 이름.<br><br>유형: 문자열 | 예 |
| Item | 항목에 대한 속성 맵으로서 항목을 정의하는 기본 키 값이 반드시 포함되어야 합니다. 항목에 대한 다른 속성 이름-값 페어가 제공될 수 있습니다. 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요.<br><br>형식: 속성 값에 대한 속성 이름의 맵. | 예 |
| Expected | 조건부 입력의 속성을 지정합니다. Expected 파라미터를 사용하면 속성 이름뿐만 아니라, DynamoDB에서 속성 값이 이미 존재하는지 또는 속성 값이 존재하며 이를 변경하기 전에 특정 값이 있는지 확인해야 하는지 여부를 제공할 수 있습니다. | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | 형식: 존재하는 경우, 속성 값에 대한 속성 이름의 맵. | |
| `Expected:AttributeName` | 조건부 입력에 대한 속성 이름입니다.<br><br>유형: 문자열 | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| `Expected:AttributeName: ExpectedAttributeValue` | 이 파라미터를 사용하여 속성 이름-값 페어의 값이 존재할지 여부를 지정합니다.<br><br>다음 JSON 표기법은 항목의 "Color" 속성이 없을 경우 해당 항목을 바꿉니다.<br><br>`"Expected" : {"Color":{"Exists":false}}`<br><br>다음 JSON 표기법은 항목을 바꾸기 전에 이름이 "Color"인 속성의 기존 값이 "Yellow"인지 여부를 확인합니다.<br><br>`"Expected" : {"Color":{"Exists":true,{"Value":{"S":"Yellow"}}}`<br><br>기본적으로 `Expected` 파라미터를 사용하고 `Value`를 제공하는 경우 DynamoDB는 속성이 존재하고 바꿀 현재 값이 있는 것으로 가정합니다. 따라서 `{"Exists":true}` 가 내재되어 있으므로 지정하지 않아도 됩니다. 요청을 다음과 같이 줄일 수 있습니다.<br><br>`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | **Note**<br>확인할 속성 값이 없을 때 `{"Exists":true}` 를 지정하면 DynamoDB는 오류를 반환합니다. | |
| `ReturnValues` | 속성 이름-값 페어가 `PutItem` 요청으로 업데이트되기 전에 이를 가져오려면 이 파라미터를 사용합니다. 가능한 파라미터 값은 `NONE`(기본) 또는 `ALL_OLD`입니다. `ALL_OLD`가 지정되어 있으며 `PutItem`이 속성 이름-값 페어를 덮어쓴 경우, 이전 항목의 내용이 반환됩니다. 이 파라미터가 제공되지 않거나 `NONE`인 경우, 아무 것도 반환되지 않습니다.<br><br>유형: 문자열 | 아니요 |

## 응답

### 구문

다음 구문 예는 요청에 `ALL_OLD`의 `ReturnValues` 파라미터가 지정되어 있다고 가정합니다. 그렇지 않을 경우 응답에는 `ConsumedCapacityUnits` 요소만 있습니다.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 85

{"Attributes":
```

```
 {"AttributeName3":{"S":"AttributeValue3"},
 "AttributeName2":{"SS":"AttributeValue2"},
 "AttributeName1":{"SS":"AttributeValue1"},
 },
"ConsumedCapacityUnits":1
}
```

| 명칭 | 설명 |
| --- | --- |
| Attributes | 입력 작업 전 속성 값입니다. 단 ReturnValues 파라미터가 요청에서 ALL_OLD 값으로 지정된 경우만 해당됩니다.<br><br>형식: 속성 이름-값 페어의 맵 |
| ConsumedCapacityUnits | 작업에서 사용한 쓰기 용량 단위의 수입니다. 이 값은 할당 처리량에 적용되는 수를 나타냅니다. 자세한 정보는 DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요.<br><br>형식: 숫자 |

## 특수 오류

| Error | 설명 |
| --- | --- |
| ConditionalCheckFailedException | 조건부 확인이 실패했습니다. 예상 속성 값이 검색되지 않았습니다. |
| ResourceNotFoundException | 지정된 항목 또는 속성을 찾을 수 없습니다. |

## 예시

AWS SDK를 사용하는 예는 DynamoDB의 항목 및 속성 작업 단원을 참조하세요.

### 샘플 요청

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## ##
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Item":
  {"time":{"N":"300"},
  "feeling":{"S":"not surprised"},
  "user":{"S":"Riley"}
  },
 "Expected":
  {"feeling":{"Value":{"S":"surprised"},"Exists":true}}
 "ReturnValues":"ALL_OLD"
}
```

### 샘플 응답

```
HTTP/1.1 200
x-amzn-RequestId: 8952fa74-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 84

{"Attributes":
 {"feeling":{"S":"surprised"},
 "time":{"N":"300"},
 "user":{"S":"Riley"}},
"ConsumedCapacityUnits":1
}
```

## 관련 작업

- UpdateItem
- DeleteItem
- GetItem
- BatchGetItem

# Query

> **Important**
>
> ***# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.***
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

## 설명

Query는 기본 키를 기준으로 하나 이상의 항목 값과 그 속성을 가져오는 작업입니다(Query는 해시 및 범위 기본 키 테이블에서만 가능합니다). 특정 HashKeyValue를 입력해야 하며, 기본 키의 RangeKeyValue에 비교 연산자를 사용하여 쿼리의 범위를 좁힐 수 있습니다. ScanIndexForward 파라미터는 범위 키를 기준으로 순방향 또는 역방향 순서로 결과를 가져오는 데 사용됩니다.

결과를 반환하지 않는 쿼리는 읽기 형식에 따라 최소 읽기 용량 단위를 사용합니다.

> **Note**
>
> 쿼리 파라미터를 만족하는 전체 항목 수의 크기가 1MB 제한을 초과하면 쿼리가 중단되고 LastEvaluatedKey와 함께 사용자에게 결과가 반환된 후 이어지는 작업에서 쿼리가 계속 됩니다. 스캔 작업과 달리 쿼리 작업은 절대로 빈 결과 집합 및 LastEvaluatedKey를 반환하지 않습니다. LastEvaluatedKey는 결과가 1MB를 초과하거나, 혹은 Limit 파라미터를 사용한 경우에만 반환됩니다.
> ConsistentRead 파라미터를 사용하면 결과를 consistent read로 설정할 수 있습니다.

## 요청

### 구문

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "Limit":2,
 "ConsistentRead":true,
```

```
 "HashKeyValue":{"S":"AttributeValue1":},
 "RangeKeyCondition": {"AttributeValueList":
[{"N":"AttributeValue2"}],"ComparisonOperator":"GT"}
 "ScanIndexForward":true,
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"AttributeName1"},
  "RangeKeyElement":{"N":"AttributeName2"}
 },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| TableName | 요청된 항목을 포함하는 테이블의 이름입니다.<br><br>유형: 문자열 | 예 |
| AttributesToGet | 속성 이름의 배열입니다. 속성 이름을 지정하지 않으면 모든 속성이 반환됩니다. 일부 속성을 찾을 수 없는 경우 결과에 표시되지 않습니다.<br><br>유형: 배열 | 아니요 |
| Limit | 반환할 최대 항목 수(대상 항목 수와 반드시 일치할 필요 없음). DynamoDB가 테이블에 대한 쿼리 중 처리할 수 있는 항목 제한 수에 이르면 쿼리를 중단하고 해당 지점까지만 대상 값을 반환한 다음 이어지는 작업에 LastEvaluatedKey 가 적용되어 쿼리를 계속합니다. 또한 DynamoDB가 이 한계에 도달하기 전에 결과 집합 크기가 1MB를 초과하는 경우에도 쿼리를 중단하고 대상 값을 | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | 반환한 다음 이어지는 작업에 `LastEvaluatedKey` 가 적용되어 쿼리를 계속합니다.<br><br>형식: 숫자 | |
| `ConsistentRead` | `true`로 설정하면 consistent read가 발생하고, 그 밖의 경우에는 eventually consistent가 사용됩니다.<br><br>타입: 부울 | 아니요 |
| `Count` | `true`로 설정할 경우DynamoDB 가 일치하는 항목 및 해당 속성의 목록이 아닌 쿼리 파라미터와 일치하는 전체 항목 수를 반환합니다. `Limit` 파라미터는 항목 수 계산 쿼리에만 적용됩니다.<br><br>`AttributesToGet` 목록을 제공하는 동안 `Count`를 `true`로 설정하지 마세요. 그렇지 않으면 DynamoDB가 유효성 검사 오류를 반환합니다. 자세한 내용은 결과 내 항목 수 계산 단원을 참조하십시오.<br><br>타입: 부울 | 아니요 |
| `HashKeyValue` | 복합 기본 키에서 해시 구성 요소의 속성 값<br><br>형식: 문자열, 숫자 또는 이진수 | 예 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| RangeKeyCondition | 쿼리에 사용되는 속성 값과 비교 연산자가 저장되는 컨테이너. 쿼리를 요청할 때는 RangeKeyCondition 이 필요하지 않습니다. HashKeyValue 만 입력하면 DynamoDB 가 지정한 해시 키 요소 값을 갖는 항목을 모두 반환합니다.<br><br>유형: 맵 | 아니요 |
| RangeKeyCondition : AttributeValueList | 쿼리 파라미터를 위해 평가할 속성 값. BETWEEN 비교를 지정하지 않으면 AttributeValueList 에 속성 값이 하나 저장됩니다. 그리고, BETWEEN 비교를 지정하면 AttributeValueList 에 속성 값이 2개 저장됩니다.<br><br>형식: AttributeValue 에 대한 ComparisonOperator 의 맵. | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| `RangeKeyCondition : ComparisonOperator` | 같다, 크다 등 제공한 속성을 평가하는 기준입니다. 다음은 쿼리 작업 시 유효한 비교 연산자입니다.<br><br>**Note**<br>크다, 같음 또는 작다 등의 문자열 값 비교는 ASCII 문자 코드 값을 기준으로 합니다. 예를 들어 `a`는 `A`보다 크고 `aa`는 `B`보다 큽니다. 코드 값 목록은 http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters를 참조하세요.<br>이진수의 경우에는, 예를 들어 쿼리 표현식을 평가할 때처럼 DynamoDB가 이진수 값을 비교하면서 이진수 데이터의 각 바이트를 부호가 없는 것으로 처리합니다.<br><br>형식: 문자열 또는 이진수 | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | EQ : 같음.<br><br>EQ의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 같지 않습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]} 과 같지 않습니다. | |
| | LE : 작거나 같음.<br><br>LE의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 같지 않습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]} 과 비교할 수 없습니다. | |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | LT : 작음.<br><br>LT의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 같지 않습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]} 과 비교할 수 없습니다. | |
| | GE : 크거나 같음.<br><br>GE의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 같지 않습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]} 과 비교할 수 없습니다. | |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | GT : 큼.<br><br>GT의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 같지 않습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]} 과 비교할 수 없습니다. | |
| | BEGINS_WITH : 접두사 여부 확인.<br><br>BEGINS_WITH 의 경우, AttributeValueList 에 문자열 또는 이진수(숫자 또는 집합 제외) 중 한 가지 형식의 AttributeValue 만 저장됩니다. 비교 대상의 속성은 문자열 또는 이진수가 되어야 합니다(숫자 또는 집합 제외). | |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | BETWEEN : 첫 번째 값보다 크거나 같음 및 두 번째 값보다 작거나 같음.<br><br>BETWEEN의 경우, 문자열, 숫자 또는 이진수(집합 제외) 중에서 동일한 형식으로 2개의 `AttributeValueList` 요소가 `AttributeValue` 에 저장되어야 합니다. 대상 값이 첫 번째 요소보다 크거나 같을 때, 그리고 두 번째 요소보다 작거나 같을 때 대상 속성이 일치합니다. 요청에서 지정한 형식과 다른 형식의 `AttributeValue` 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 `{"S":"6"}` 은 `{"N":"6"}` 과 비교할 수 없습니다. 또한 `{"N":"6"}` 은 `{"NS":["6", "2", "1"]}`과 비교할 수 없습니다. | |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| ScanIndexForward | 인덱스의 오름차순 또는 내림차순을 지정합니다. DynamoDB는 범위 키에서 지정한 요청 순서를 반영하여 결과를 반환합니다. 데이터 형식이 숫자라면 결과는 숫자의 순서대로 반환됩니다. 그렇지 않으면 ASCII 문자 코드 값에 따라 순서가 결정됩니다.<br><br>타입: 부울<br><br>기본값은 true입니다(오름차순). | 아니요 |
| ExclusiveStartKey | 이전 쿼리를 계속할 항목의 기본 키. 결과 집합 크기나 Limit 파라미터 등으로 인해 쿼리가 끝나기도 전에 중단된 경우에는 이전 쿼리가 이 값을 LastEvaluatedKey 로 입력할 수도 있습니다. 그러면 LastEvaluatedKey 가 새로운 쿼리 요청으로 다시 전달되어 중단된 지점부터 쿼리 작업을 계속합니다.<br><br>형식: 복합 기본 키의 HashKeyElement , 또는 HashKeyElement 및 RangeKeyElement . | 아니요 |

## 응답

### 구문

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
    "AttributeName2":{"N":"AttributeValue2"},
    "AttributeName3":{"S":"AttributeValue3"}
    },{
    "AttributeName1":{"S":"AttributeValue3"},
    "AttributeName2":{"N":"AttributeValue4"},
    "AttributeName3":{"S":"AttributeValue3"},
    "AttributeName5":{"B":"dmFsdWU="}
}],
    "LastEvaluatedKey":{"HashKeyElement":{"AttributeValue3":"S"},
                        "RangeKeyElement":{"AttributeValue4":"N"}
     },
     "ConsumedCapacityUnits":1
}
```

| 명칭 | 설명 |
| --- | --- |
| Items | 쿼리 파라미터를 만족하는 항목 속성<br><br>형식: 속성 이름 맵과 속성의 데이터 형식 및 값. |
| Count | 응답의 항목 수. 자세한 내용은 결과 내 항목 수 계산 단원을 참조하십시오.<br><br>형식: 숫자 |
| LastEvaluatedKey | 쿼리 작업이 중단된 항목의 기본 키(이전 결과 집합 포함). 이 값은 새로운 요청에서는 이 값을 제외하고 새 작업을 시작할 때 사용됩니다. |

| 명칭 | 설명 |
| --- | --- |
| | 전체 쿼리 결과 집합이 완료되면("마지막 페이지"까지 처리) `LastEvaluatedKey` 는 `null` 값을 갖습니다.<br><br>형식: 복합 기본 키의 `HashKeyElement` , 또는 `HashKeyElement` 및 `RangeKeyElement` . |
| `ConsumedCapacityUnits` | 작업에 사용된 읽기 용량 단위의 수. 이 값은 할당 처리량에 적용되는 수를 나타냅니다. 자세한 정보는 DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요.<br><br>형식: 숫자 |

## 특수 오류

| Error | 설명 |
| --- | --- |
| `ResourceNotFoundException` | 지정된 테이블을 찾을 수 없습니다. |

## 예시

AWS SDK를 사용하는 예는 DynamoDB에서 테이블 쿼리 단원을 참조하세요.

### 샘플 요청

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## ##
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"John"},
 "ScanIndexForward":false,
```

```
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"John"},
  "RangeKeyElement":{"S":"The Matrix"}
 }
}
```

### 샘플 응답

```
HTTP/1.1 200
x-amzn-RequestId: 3647e778-71eb-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The End"}
 },{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The Beatles"}
 }],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"John"},"RangeKeyElement":{"S":"The
 Beatles"}},
"ConsumedCapacityUnits":1
}
```

### 샘플 요청

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## ##
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"Airplane"},
 "RangeKeyCondition":{"AttributeValueList":[{"N":"1980"}],"ComparisonOperator":"EQ"},
```

```
 "ScanIndexForward":false}
```

### 샘플 응답

```
HTTP/1.1 200
x-amzn-RequestId: 8b9ee1ad-774c-11e0-9172-d954e38f553a
content-type: application/x-amz-json-1.0
content-length: 119

{"Count":1,"Items":[{
 "fans":{"SS":["Dave","Aaron"]},
 "name":{"S":"Airplane"},
 "rating":{"S":"***"},
 "year":{"N":"1980"}
 }],
"ConsumedCapacityUnits":1
}
```

### 관련 작업

- 스캔

## 스캔

> **Important**
>
> ***# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.***
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

### 설명

Scan 작업은 테이블에 대한 전체 스캔을 수행하여 하나 이상의 항목 및 해당 속성을 반환합니다. ScanFilter를 제공하여 더 많은 특정 결과를 가져옵니다.

> **Note**
>
> 스캔한 총 항목의 수가 1MB 제한을 초과하면 스캔이 중지되고 LastEvaluatedKey과 함께 사용자에게 결과가 반환되며 후속 작업에서 스캔을 계속합니다. 또한 결과에는 제한을 초과하

는 항목 수가 포함되어 있습니다. 스캔에는 필터 조건을 충족하는 테이블 데이터가 없을 수 있습니다.
결과 세트는 최종적으로 일관적입니다.

## 요청

### 구문

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Limit": 2,
    "ScanFilter":{
        "AttributeName1":{"AttributeValueList":
[{"S":"AttributeValue"}],"ComparisonOperator":"EQ"}
    },
    "ExclusiveStartKey":{
        "HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"}
    },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| TableName | 요청된 항목을 포함하는 테이블의 이름입니다.<br><br>유형: 문자열 | 예 |
| AttributesToGet | 속성 이름의 배열입니다. 속성 이름을 지정하지 않으면 모든 속성이 반환됩니다. 일부 속성을 찾을 수 없는 경우 결과에 표시되지 않습니다. | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | 유형: 배열 | |
| `Limit` | 평가할 최대 항목 수입니다(반드시 일치하는 항목 수는 아님). DynamoDB가 결과를 처리하는 중에 한도까지 항목 수를 처리하면, 이를 중지하고 해당 지점까지 일치하는 값을 반환하며, 후속 작업에 `LastEvaluatedKey` 를 적용하여 항목 가져오기를 계속합니다. 또한 DynamoDB가 1MB에 도달하기 전에 스캔한 데이터 설정 크기가 이 한도를 초과하면, 스캔을 중지하고 이 한도까지 일치하는 값을 반환하며, 후속 작업에 `LastEvaluatedKey` 를 적용하여 스캔을 계속합니다.<br><br>형식: 숫자 | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| Count | true로 설정한 경우, DynamoDB는 스캔 작업의 총 항목 수를 반환합니다. 작업에 할당된 필터에 대해 일치하는 항목이 없는 경우도 마찬가지입니다. 수 계산 전용 스캔에 Limit 파라미터를 적용할 수 있습니다.<br><br>AttributesToGet 목록을 제공하는 동안 Count를 true로 설정하지 마세요. 그렇지 않으면 DynamoDB가 유효성 검사 오류를 반환합니다. 자세한 내용은 결과 내 항목 수 계산 단원을 참조하십시오.<br><br>타입: 부울 | 아니요 |
| ScanFilter | 스캔 결과를 평가하고 원하는 값만 반환합니다. 여러 조건이 "AND" 작업으로 처리됩니다. 모든 조건이 결과에 포함되어야 합니다.<br><br>형식: 비교 연산자가 포함된 값에 대한 속성 이름의 맵. | 아니요 |
| ScanFilter :AttributeValueList | 필터의 스캔 결과를 평가하는 값과 조건입니다.<br><br>형식: AttributeValue 에 대한 Condition 의 맵. | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| `ScanFilter : ComparisonOperator` | 같다, 크다 등 제공한 속성을 평가하는 기준입니다. 다음은 스캔 작업에 대해 유효한 비교 연산자입니다.<br><br>**Note**<br>크다, 같음 또는 작다 등의 문자열 값 비교는 ASCII 문자 코드 값을 기준으로 합니다. 예를 들어 `a`는 `A`보다 크고 `aa`는 `B`보다 큽니다. 코드 값 목록은 http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters를 참조하세요.<br>이진수의 경우에는, 예를 들어 쿼리 표현식을 평가할 때처럼 DynamoDB가 이진수 값을 비교하면서 이진수 데이터의 각 바이트를 부호가 없는 것으로 처리합니다.<br><br>형식: 문자열 또는 이진수 | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | EQ : 같음.<br><br>EQ의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 같지 않습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]} 과 같지 않습니다. | |
| | NE : 같지 않음.<br><br>NE의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 같지 않습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]} 과 같지 않습니다. | |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | LE : 작거나 같음.<br><br>LE의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 같지 않습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]} 과 비교할 수 없습니다. | |
| | LT : 작음.<br><br>LT의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 같지 않습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]} 과 비교할 수 없습니다. | |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | GE : 크거나 같음.<br><br>GE의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 같지 않습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]} 과 비교할 수 없습니다. | |
| | GT : 큼.<br><br>GT의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 같지 않습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]} 과 비교할 수 없습니다. | |
| | NOT_NULL : 속성 있음. | |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | NULL : 속성 없음. | |
| | CONTAINS : 세트의 하위 시퀀스 또는 값 확인.<br><br>CONTAINS의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 비교의 대상 속성이 문자열이면 작업은 하위 문자열 일치를 확인합니다. 비교의 대상 속성이 이진수이면 작업은 입력과 일치하는 대상의 하위 시퀀스를 찾습니다. 비교의 대상 속성이 세트("SS", "NS" 또는 "BS")이면 작업은 해당 세트의 구성 요소(하위 문자열 아님)를 확인합니다. | |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | NOT_CONTAINS : 세트의 하위 시퀀스 부재 또는 값의 부재 확인.<br><br>NOT_CONTAINS 의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 하나의 AttributeValue 만 포함할 수 있습니다(집합 아님). 비교의 대상 속성이 문자열이면 작업은 하위 문자열이 일치하지 않는지 확인합니다. 비교의 대상 속성이 이진수이면 입력과 일치하는 대상의 하위 시퀀스가 없는지 확인합니다. 비교의 대상 속성이 세트("SS", "NS" 또는 "BS")이면 작업은 해당 세트의 구성 요소(하위 문자열 아님)의 부재를 확인합니다. | |
| | BEGINS_WITH : 접두사 여부 확인.<br><br>BEGINS_WITH 의 경우, AttributeValueList 에 문자열 또는 이진수(숫자 또는 집합 제외) 중 한 가지 형식의 AttributeValue 만 저장됩니다. 비교 대상의 속성은 문자열 또는 이진수가 되어야 합니다(숫자 또는 집합 제외). | |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | IN : 정확한 일치 확인.<br><br>IN의 경우, AttributeValueList 는 문자열, 숫자 또는 이진수 유형 중 두 개 이상의 AttributeValue 를 포함할 수 있습니다(집합 아님). 비교 대상 속성이 일치하려면 형식이 동일하고 값도 정확해야 합니다. 문자열은 문자열 세트와 일치하지 않습니다. | |
| | BETWEEN : 첫 번째 값보다 크거나 같음 및 두 번째 값보다 작거나 같음.<br><br>BETWEEN의 경우, 문자열, 숫자 또는 이진수(집합 제외) 중에서 동일한 형식으로 2개의 AttributeValueList 요소가 AttributeValue 에 저장되어야 합니다. 대상 값이 첫 번째 요소보다 크거나 같을 때, 그리고 두 번째 요소보다 작거나 같을 때 대상 속성이 일치합니다. 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 {"S":"6"} 은 {"N":"6"} 과 비교할 수 없습니다. 또한 {"N":"6"} 은 {"NS":["6", "2", "1"]}과 비교할 수 없습니다. | |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| ExclusiveStartKey | 이전 스캔을 계속할 항목의 기본 키입니다. 결과 세트 크기 또는 Limit 파라미터 중 하나로 인해 이전 스캔 작업이 전체 테이블을 스캔하기 전에 중단된 경우 해당 스캔은 이 값을 제공할 수 있습니다. LastEvaluatedKey 는 새 스캔 요청으로 다시 전달되어 해당 지점에서 작업을 계속할 수 있습니다.<br><br>형식: 복합 기본 키의 HashKeyElement , 또는 HashKeyElement 및 RangeKeyElement . | 아니요 |

## 응답

### 구문

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 229

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
    "AttributeName2":{"S":"AttributeValue2"},
    "AttributeName3":{"S":"AttributeValue3"}
    },{
    "AttributeName1":{"S":"AttributeValue4"},
    "AttributeName2":{"S":"AttributeValue5"},
    "AttributeName3":{"S":"AttributeValue6"},
    "AttributeName5":{"B":"dmFsdWU="}
    }],
    "LastEvaluatedKey":
        {"HashKeyElement":{"S":"AttributeName1"},
```

```
        "RangeKeyElement":{"N":"AttributeName2"},
    "ConsumedCapacityUnits":1,
    "ScannedCount":2}
}
```

| 명칭 | 설명 |
| --- | --- |
| Items | 작업 파라미터를 충족하는 속성의 컨테이너입니다.<br><br>형식: 속성 이름 맵과 속성의 데이터 형식 및 값. |
| Count | 응답의 항목 수. 자세한 내용은 결과 내 항목 수 계산 단원을 참조하십시오.<br><br>형식: 숫자 |
| ScannedCount | 필터를 적용하기 전 전체 스캔의 항목 수입니다. ScannedCount 값이 높지만 Count 결과가 거의 없거나 전혀 없는 경우 스캔 작업이 비효율적이라는 것을 나타냅니다. 자세한 내용은 결과 내 항목 수 계산 단원을 참조하십시오.<br><br>형식: 숫자 |
| LastEvaluatedKey | 스캔 작업이 중지된 항목의 기본 키입니다. 후속 스캔 작업에 이 값을 제공하여 해당 지점부터 작업을 계속합니다.<br><br>전체 스캔 결과 설정이 완료되면(예: 작업이 "마지막 페이지"를 처리)LastEvaluatedKey 는 null입니다. |
| ConsumedCapacityUnits | 작업에 사용된 읽기 용량 단위의 수. 이 값은 할당 처리량에 적용되는 수를 나타냅니다. 자세한 정보는 DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요.<br><br>형식: 숫자 |

## 특수 오류

| Error | 설명 |
| --- | --- |
| `ResourceNotFoundException` | 지정된 테이블을 찾을 수 없습니다. |

## 예시

AWS SDK를 사용하는 예는 DynamoDB에서 테이블 스캔 단원을 참조하세요.

### 샘플 요청

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## ##
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable","ScanFilter":{}}
```

### 샘플 응답

```
HTTP/1.1 200
x-amzn-RequestId: 4e8a5fa9-71e7-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 465

{"Count":4,"Items":[{
 "date":{"S":"1980"},
 "fans":{"SS":["Dave","Aaron"]},
 "name":{"S":"Airplane"},
 "rating":{"S":"***"}
 },{
 "date":{"S":"1999"},
 "fans":{"SS":["Ziggy","Laura","Dean"]},
 "name":{"S":"Matrix"},
 "rating":{"S":"*****"}
 },{
 "date":{"S":"1976"},
 "fans":{"SS":["Riley"]},"
 name":{"S":"The Shaggy D.A."},
```

```
 "rating":{"S":"**"}
 },{
 "date":{"S":"1985"},
 "fans":{"SS":["Fox","Lloyd"]},
 "name":{"S":"Back To The Future"},
 "rating":{"S":"****"}
 }],
    "ConsumedCapacityUnits":0.5
 "ScannedCount":4}
```

## 샘플 요청

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## ##
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0
content-length: 125

{"TableName":"comp5",
 "ScanFilter":
  {"time":
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 }
}
```

## 샘플 응답

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 262
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":2,
 "Items":[
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
  "time":{"N":"2000"},
  "user":{"S":"Casey"}},
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
```

```
  "time":{"N":"2000"},
  "user":{"S":"Fredy"}
  }],
"ConsumedCapacityUnits":0.5
"ScannedCount":4
}
```

### 샘플 요청

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## ##
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Limit":2,
 "ScanFilter":
  {"time":
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 },
 "ExclusiveStartKey":
  {"HashKeyElement":{"S":"Fredy"},"RangeKeyElement":{"N":"2000"}}
}
```

### 샘플 응답

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 232
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":1,
 "Items":[
  {"friends":{"SS":["Jane","James","John"]},
  "status":{"S":"exercising"},
  "time":{"N":"2200"},
  "user":{"S":"Roger"}}
 ],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"250"}},
"ConsumedCapacityUnits":0.5
```

```
"ScannedCount":2
}
```

### 관련 작업

- Query
- BatchGetItem

## UpdateItem

> **Important**
>
> **`# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.`**
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

### 설명

기존 항목의 속성을 편집합니다. 조건부 업데이트를 수행할 수 있습니다(속성 이름-값 페어가 없는 경우 새로 삽입하거나 특정 예상 속성 값이 있는 경우 기존 이름-값 페어를 바꿉니다).

> **Note**
>
> UpdateItem을 사용하여 기본 키 속성을 업데이트할 수 없습니다. 대신 해당 항목을 삭제하고 PutItem을 사용하여 새 속성이 포함된 새 항목을 만듭니다.

UpdateItem 작업에는 업데이트를 수행하는 방법을 정의하는 `Action` 파라미터가 포함되어 있습니다. 속성 값을 입력, 삭제 또는 추가할 수 있습니다.

속성 값은 null이 될 수 없습니다. 문자열과 이진 형식 속성 길이는 0보다 커야 합니다. 설정 유형 속성은 비어 있으면 안 됩니다. 값이 비어 있는 요청은 `ValidationException`으로 거부됩니다.

기존 항목에 지정된 기본 키가 있는 경우:

- PUT - 지정된 속성을 추가합니다. 속성이 이미 있는 경우 새 값으로 바꿉니다.
- DELETE - 지정된 값이 없는 경우 속성과 해당 값을 제거합니다. 값 세트가 지정된 경우 지정된 세트의 값이 이전 세트에서 제거됩니다. 따라서 속성 값에 [a,b,c]가 포함되어 있고 삭제 작업에 [a,c]가 포

함되어 있으면 최종 속성 값은 [b]가 됩니다. 지정된 값 유형은 기존 값 유형과 일치해야 합니다. 빈 세트를 지정하는 것은 유효하지 않습니다.

- ADD - 숫자에 대해 또는 대상 속성이 세트(문자열 세트 포함)인 경우에만 추가 작업을 사용합니다. ADD는 대상 속성이 단일 문자열 값이거나 스칼라 이진 값인 경우에는 작동하지 않습니다. 지정된 값은 숫자 값에 추가(기존 숫자 값 증가 또는 감소)되거나 문자열 세트에 추가 값으로 추가됩니다. 값 세트가 지정된 경우 해당 값이 기존 세트에 추가됩니다. 예를 들어 원래 값이 [1,2]이고 제공한 값이 [3]이면 추가 작업 후 세트는 [4,5]가 아니라 [1,2,3]이 됩니다. 세트 속성에 대한 Add 작업이 지정되어 있으며 지정된 속성 유형이 기존 세트 유형과 일치하지 않는 경우 오류가 발생합니다.

  존재하지 않는 속성에 대해 ADD를 사용하면 해당 속성과 그 값이 항목에 추가됩니다.

항목이 지정된 기본 키와 일치하지 않는 경우:

- PUT - 지정된 기본 키가 포함된 새 항목을 만듭니다. 그런 다음 지정된 속성을 추가합니다.
- DELETE - 아무 것도 발생하지 않습니다.
- ADD - 속성 값에 대해 제공된 기본 키와 숫자(또는 숫자 세트)가 포함된 항목을 만듭니다. 문자열 또는 이진 형식에는 유효하지 않습니다.

> **Note**
>
> ADD를 사용하여 업데이트 전에는 존재하지 않은 항목의 숫자 값을 증가 또는 감소시키는 경우, DynamoDB는 초기 값으로 0을 사용합니다. 또한 ADD를 사용하여 항목을 업데이트함으로써 업데이트 전에는 존재하지 않은 속성의 숫자 값을 증가 또는 감소시키는 경우, DynamoDB는 초기 값으로 0을 사용합니다. 예를 들어, ADD를 사용하여 업데이트 전에는 존재하지 않은 속성에 +3을 추가합니다. DynamoDB는 0을 초기 값으로 사용하고 업데이트 후의 값은3입니다.

이 작업의 사용에 대한 자세한 내용은 DynamoDB의 항목 및 속성 작업 단원을 참조하세요.

## 요청

### 구문

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0
```

```
{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},
        "RangeKeyElement":{"N":"AttributeValue2"}},
    "AttributeUpdates":{"AttributeName3":{"Value":
{"S":"AttributeValue3_New"},"Action":"PUT"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3_Current"}}},
    "ReturnValues":"ReturnValuesConstant"
}
```

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| TableName | 업데이트할 항목이 포함된 테이블 이름입니다.<br><br>유형: 문자열 | 예 |
| Key | 항목을 정의하는 기본 키입니다. 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요.<br><br>형식: 해당 값에 대한 HashKeyElement 및 해당 값에 대한 RangeKeyElement 의 맵 | 예 |
| AttributeUpdates | 업데이트의 새 값 및 작업에 대한 속성 이름의 맵입니다. 속성 이름은 수정할 속성을 지정하며, 기본 키 속성은 포함할 수 없습니다.<br><br>형식: 속성 업데이트에 대한 속성 이름, 값 및 작업의 맵. | |
| AttributeUpdates :Action | 업데이트 수행 방법을 지정합니다. 가능한 값은 PUT(기본), ADD 또는 DELETE입니다. 의미 | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | 는 UpdateItem 설명에 있습니다.<br><br>유형: 문자열<br><br>기본값: PUT | |
| `Expected` | 조건부 업데이트의 속성을 지정합니다. `Expected` 파라미터를 사용하면 속성 이름뿐만 아니라, DynamoDB에서 속성 값이 이미 존재하는지 또는 속성 값이 존재하며 이를 변경하기 전에 특정 값이 있는지 확인해야 하는지 여부를 제공할 수 있습니다.<br><br>형식: 속성 이름의 맵 | 아니요 |
| `Expected:AttributeName` | 조건부 입력에 대한 속성 이름입니다.<br><br>유형: 문자열 | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| `Expected:AttributeName: ExpectedAttributeValue` | 이 파라미터를 사용하여 속성 이름-값 페어의 값이 존재할지 여부를 지정합니다.<br><br>다음 JSON 표기법은 항목의 "Color" 속성이 없을 경우 해당 항목을 업데이트합니다.<br><br>`"Expected" : {"Color":{"Exists":false}}`<br><br>다음 JSON 표기법은 항목을 업데이트하기 전에 이름이 "Color"인 속성의 기존 값이 "Yellow"인지 여부를 확인합니다.<br><br>`"Expected" : {"Color":{"Exists":true},{"Value":{"S":"Yellow"}}}`<br><br>기본적으로 `Expected` 파라미터를 사용하고 `Value`를 제공하는 경우 DynamoDB는 속성이 존재하고 바꿀 현재 값이 있는 것으로 가정합니다. 따라서 `{"Exists":true}` 가 내재되어 있으므로 지정하지 않아도 됩니다. 요청을 다음과 같이 줄일 수 있습니다.<br><br>`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` | 아니요 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | **Note**<br>확인할 속성 값이 없을 때 `{"Exists" :true}` 를 지정하면 DynamoDB는 오류를 반환합니다. | |
| `ReturnValues` | 속성 이름-값 페어가 `UpdateItem` 요청으로 업데이트되기 전에 이를 가져오려면 이 파라미터를 사용합니다. 가능한 파라미터 값은 `NONE`(기본) 또는 `ALL_OLD`, `UPDATED_OLD` , `ALL_NEW` 또는 `UPDATED_NEW` 입니다. `ALL_OLD`가 지정되어 있으며 `UpdateItem` 이 속성 이름-값 페어를 덮어쓴 경우, 이전 항목의 내용이 반환됩니다. 이 파라미터가 제공되지 않거나 `NONE`인 경우, 아무 것도 반환되지 않습니다. `ALL_NEW`가 지정된 경우, 해당 항목에 대한 새 버전의 모든 속성이 반환됩니다. `UPDATED_NEW` 가 지정된 경우, 업데이트된 속성의 새 버전만 반환됩니다.<br><br>유형: 문자열 | 아니요 |

# 응답

## 구문

다음 구문 예는 요청에 ALL_OLD의 ReturnValues 파라미터가 지정되어 있다고 가정합니다. 그렇지 않을 경우 응답에는 ConsumedCapacityUnits 요소만 있습니다.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 140

{"Attributes":{
 "AttributeName1":{"S":"AttributeValue1"},
 "AttributeName2":{"S":"AttributeValue2"},
 "AttributeName3":{"S":"AttributeValue3"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits":1
}
```

| 명칭 | 설명 |
| --- | --- |
| Attributes | 속성 이름-값 페어의 맵입니다. 단 ReturnValues 파라미터가 요청에서 NONE 이외의 값으로 지정된 경우만 해당됩니다.<br><br>형식: 속성 이름-값 페어의 맵 |
| ConsumedCapacityUnits | 작업에서 사용한 쓰기 용량 단위의 수입니다. 이 값은 할당 처리량에 적용되는 수를 나타냅니다. 자세한 정보는 DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요.<br><br>형식: 숫자 |

## 특수 오류

| Error | 설명 |
| --- | --- |
| ConditionalCheckFailedException | 조건부 확인이 실패했습니다. 속성("+ name +") 값은 ("+ value +")이지만 ("+ expValue +")로 예상되었습니다. |
| ResourceNotFoundExceptions | 지정된 항목 또는 속성을 찾을 수 없습니다. |

## 예시

AWS SDK를 사용하는 예는 DynamoDB의 항목 및 속성 작업 단원을 참조하세요.

### 샘플 요청

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## ##
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
    "Key":
        {"HashKeyElement":{"S":"Julie"},"RangeKeyElement":{"N":"1307654350"}},
    "AttributeUpdates":
        {"status":{"Value":{"S":"online"},
        "Action":"PUT"}},
    "Expected":{"status":{"Value":{"S":"offline"}}},
    "ReturnValues":"ALL_NEW"
}
```

### 샘플 응답

```
HTTP/1.1 200 OK
x-amzn-RequestId: 5IMHO7F01Q9P7Q6QMKMMI3R3QRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 121
Date: Fri, 26 Aug 2011 21:05:00 GMT

{"Attributes":
```

```
    {"friends":{"SS":["Lynda, Aaron"]},
    "status":{"S":"online"},
    "time":{"N":"1307654350"},
    "user":{"S":"Julie"}},
"ConsumedCapacityUnits":1
}
```

### 관련 작업

- PutItem
- DeleteItem

## UpdateTable

> **Important**
>
> ***# #### #### API ## 2011-12-05# ## ###### # ####### ##### # ###.***
> 현재 하위 수준 API에 대한 설명서는 Amazon DynamoDB API 참조 섹션을 참조하세요.

### 설명

주어진 테이블의 할당 처리량을 업데이트합니다. 테이블 처리량을 설정하면 성능을 관리하는 데 효과적이기 때문에 DynamoDB에서 프로비저닝 처리량 기능으로 지원되고 있습니다. 자세한 내용은 DynamoDB 프로비저닝된 용량 모드 단원을 참조하십시오.

프로비저닝 처리량 값은 Amazon DynamoDB의 할당량의 상한값과 하한값에 따라 높이거나 낮출 수 있습니다.

업데이트 작업이 성공하려면 테이블이 `ACTIVE` 상태이어야 합니다. UpdateTable은 비동기식 작업입니다. 즉, 작업 중에는 테이블이 `UPDATING` 상태가 됩니다. 테이블이 `UPDATING` 상태이더라도 할당 처리량은 호출 전과 같습니다. 새로운 할당 처리량 설정은 UpdateTable 작업이 끝나고 테이블이 `ACTIVE` 상태로 돌아온 후부터 적용됩니다.

### 요청

#### 구문

```
// This header is abbreviated.
```

```
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| TableName | 업데이트할 테이블 이름<br><br>유형: 문자열 | 예 |
| ProvisionedThroughput | 지정 테이블의 새로운 처리량으로서 ReadCapacityUnits 값과 WriteCapacityUnits 값으로 구성됩니다. DynamoDB 프로비저닝된 용량 모드 섹션을 참조하세요.<br><br>유형: 배열 | 예 |
| ProvisionedThroughput :ReadCapacityUnits | DynamoDB가 다른 작업과 로드 밸런스를 맞출 때까지 지정 테이블에서 사용되는 consistent ReadCapacityUnits 의 초당 최소 수를 설정합니다.<br><br>Eventually consistent read 작업은 consistent read 작업에 비해 필요한 부하가 적습니다. 따라서 초당 consistent ReadCapacityUnits 를 50으로 설정하면 초당 eventuall | 예 |

| 명칭 | 설명 | 필수 |
| --- | --- | --- |
| | y consistent ReadCapacityUnits 는 100이 됩니다.<br><br>형식: 숫자 | |
| ProvisionedThroughput :WriteCapacityUnits | DynamoDB가 다른 작업과 로드 밸런스를 맞출 때까지 지정 테이블에서 사용되는 WriteCapacityUnits 의 초당 최소 수를 설정합니다.<br><br>형식: 숫자 | 예 |

## 응답

### 구문

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/json
Content-Length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeValue1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeValue2","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
        "WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"UPDATING"}}
```

| 명칭 | 설명 |
| --- | --- |
| CreationDateTime | 테이블 생성 날짜<br><br>형식: 숫자 |
| KeySchema | 테이블의 기본 키(단순 또는 복합) 구조. HashKeyElement 에서는 이름-값 페어가 필수이지만 RangeKeyElement 에서는 선택 사항입니다(복합 기본 키의 경우에만 필요함). 최대 해시 키 크기는 2,048byte입니다. 최대 범위 키 크기는 1,024byte입니다. 두 키의 크기 제한은 별도로 적용됩니다(해시 + 범위 키 2,048 + 1,024 합산). 기본 키에 대한 자세한 내용은 프라이머리 키 단원을 참조하세요.<br><br>형식: HashKeyElement 맵, 또는 복합 기본 키의 HashKeyElement 및 RangeKeyElement |
| ProvisionedThroughput | LastIncreaseDateTime 값(해당되는 경우)과 LastDecreaseDateTime 값(해당되는 경우)을 포함하여 지정 테이블의 현재 처리량 설정,<br><br>유형: 배열 |
| TableName | 업데이트된 테이블 이름<br><br>유형: 문자열 |
| TableStatus | 현재 테이블 상태(CREATING, ACTIVE, DELETING 또는 UPDATING). UPDATING이 되어야 합니다.<br><br>DescribeTables 작업을 사용하여 테이블 상태를 확인합니다.<br><br>유형: 문자열 |

## 특수 오류

| Error | 설명 |
| --- | --- |
| ResourceNotFoundException | 지정된 테이블을 찾을 수 없습니다. |
| ResourceInUseException | 테이블이 ACTIVE 상태가 아닙니다. |

## 예

### 샘플 요청

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"comp1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

### 샘플 응답

```
HTTP/1.1 200 OK
content-type: application/x-amz-json-1.0
content-length: 390
Date: Sat, 19 Nov 2011 00:46:47 GMT

{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
        "WriteCapacityUnits":10},
    "TableName":"comp1",
```

```
    "TableStatus":"UPDATING"}
}
```

### 관련 작업

- CreateTable
- DescribeTables
- DeleteTable

## 레거시 DynamoDB 조건부 파라미터

이 문서에서는 DynamoDB의 레거시 조건부 파라미터에 대한 개요를 제공하며 이 대신 새 표현식 파라미터를 사용할 것을 권장합니다. AttributesToGet, AttributeUpdates, ConditionalOperator, Expected, KeyConditions, QueryFilter, and ScanFilter 등의 파라미터에 대해 자세히 설명하고, 새 표현식 파라미터를 대신 사용하는 방법에 대한 예를 보여줍니다.

> **Important**
>
> 가능하면 이러한 레거시 파라미터 대신 새 표현식 파라미터를 사용하는 것이 좋습니다. 자세한 내용은 DynamoDB에서 표현식 사용 단원을 참조하십시오.
> 또한 DynamoDB는 단일 호출에서 기존 조건부 파라미터와 표현식 파라미터의 혼용을 허용하지 않습니다. 예를 들어 `AttributesToGet`과 `ConditionExpression`을 사용해 `Query` 작업을 호출하면 오류가 발생합니다.

다음 표는 이러한 레거시 파라미터를 여전히 지원하는 DynamoDB API 작업과, 대신 사용할 수 있는 표현식 파라미터를 나타낸 것입니다. 이 표는 애플리케이션에서 표현식 파라미터를 대신 사용하도록 업데이트하려는 경우 유용합니다.

| 이 API 작업을 사용하는 경우 | 함께 사용하는 기존 파라미터 | 대신 사용할 수 있는 표현식 파라미터 |
|---|---|---|
| `BatchGetItem` | `AttributesToGet` | `ProjectionExpression` |
| `DeleteItem` | `Expected` | `ConditionExpression` |
| `GetItem` | `AttributesToGet` | `ProjectionExpression` |

| 이 API 작업을 사용하는 경우 | 함께 사용하는 기존 파라미터 | 대신 사용할 수 있는 표현식 파라미터 |
|---|---|---|
| PutItem | Expected | ConditionExpression |
| Query | AttributesToGet | ProjectionExpression |
| | KeyConditions | KeyConditionExpression |
| | QueryFilter | FilterExpression |
| Scan | AttributesToGet | ProjectionExpression |
| | ScanFilter | FilterExpression |
| UpdateItem | AttributeUpdates | UpdateExpression |
| | Expected | ConditionExpression |

다음 단원에서는 기존 조건부 파라미터에 대한 자세한 내용을 제공합니다.

주제

- AttributesToGet(레거시)
- AttributeUpdates(레거시)
- ConditionalOperator(레거시)
- Expected(레거시)
- KeyConditions(레거시)
- QueryFilter(레거시)
- ScanFilter(레거시)
- 기존 파라미터를 이용한 조건 작성

# AttributesToGet(레거시)

> **Note**
>
> 가능하면 이러한 레거시 파라미터 대신 새 표현식 파라미터를 사용하는 것이 좋습니다. 자세한 내용은 DynamoDB에서 표현식 사용 단원을 참조하십시오. 이 파라미터를 대체하는 새 파라미터에 대한 자세한 내용은 대신 ProjectionExpression 사용 섹션을 참조하세요.

레거시 조건 파라미터 `AttributesToGet`은 DynamoDB에서 검색되는 하나 이상의 속성 어레이입니다. 속성 이름을 제공하지 않으면 모든 속성이 반환됩니다. 요청한 속성을 찾을 수 없는 경우 결과에 표시되지 않습니다.

`AttributesToGet`을 사용하면 목록이나 맵 유형의 속성을 검색할 수 있지만 목록이나 맵 안의 개별 요소는 검색할 수 없습니다.

`AttributesToGet`은 할당 처리량에 영향을 주지 않습니다. DynamoDB는 애플리케이션에 반환되는 데이터 크기가 아닌 항목 크기를 기준으로 소비된 용량 단위를 결정합니다.

## 대신 ProjectionExpression 사용 - 예

Music 테이블에서 항목을 검색하고 싶은데, 속성 중 일부만 반환되길 원한다고 가정합니다. 다음 AWS CLI 예제에서와 같이 `GetItem` 요청을 `AttributesToGet` 파라미터와 함께 사용할 수 있습니다.

```
aws dynamodb get-item \
    --table-name Music \
    --attributes-to-get '["Artist", "Genre"]' \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

대신에 `ProjectionExpression`을 사용할 수 있습니다.

```
aws dynamodb get-item \
    --table-name Music \
    --projection-expression "Artist, Genre" \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
```

```
    }'
```

## AttributeUpdates(레거시)

> **Note**
>
> 가능하면 이러한 레거시 파라미터 대신 새 표현식 파라미터를 사용하는 것이 좋습니다. 자세한 내용은 DynamoDB에서 표현식 사용 단원을 참조하십시오. 이 파라미터를 대체하는 새 파라미터에 대한 자세한 내용은 대신 UpdateExpression을 사용합니다. 섹션을 참조하세요.

`UpdateItem` 작업에서, 레거시 조건 파라미터 `AttributeUpdates`에는 수정할 속성 이름, 각각에서 수행할 작업, 각각의 새 값이 포함되어 있습니다. 해당 테이블의 인덱스에 대한 인덱스 키 속성인 속성을 업데이트하는 경우, 속성 유형이 표 설명의 `AttributesDefinition`에 정의되어 있는 인덱스 키 유형과 일치해야 합니다. `UpdateItem`을 사용하여 다른 키가 아닌 속성을 업데이트할 수 있습니다.

속성 값은 null이 될 수 없습니다. 문자열과 이진 형식 속성 길이는 0보다 커야 합니다. 설정 유형 속성은 비어 있으면 안 됩니다. 값이 비어 있는 요청은 `ValidationException` 예외로 거부됩니다.

각 `AttributeUpdates` 요소는 수정할 속성 이름 및 다음 파라미터로 구성됩니다.

- `Value` - 이 속성의 새 값(해당되는 경우)입니다.
- `Action` - 업데이트 수행 방법을 지정하는 값입니다. 이 작업은 데이터 유형이 숫자이거나 집합인 기존 속성에 대해서만 유효하므로, 다른 데이터 유형에는 `ADD`를 사용하지 마십시오.

  지정된 기본 키를 가진 항목이 테이블에 없으면, 다음 값이 다음 작업을 수행합니다.

  - `PUT` - 지정된 속성을 항목에 추가합니다. 속성이 이미 있는 경우 새 값으로 바꿉니다.
  - `DELETE` - `DELETE`에 대해 지정된 값이 없는 경우, 속성과 해당 값을 제거합니다. 지정된 값의 데이터 유형은 기존 값의 데이터 유형과 일치해야 합니다.

    값 세트가 지정된 경우 해당 값이 이전 세트에서 제거됩니다. 예를 들어, 속성 값이 세트 `[a,b,c]`이고 `DELETE` 작업이 `[a,c]`를 지정하면, 최종 속성 값은 `[b]`가 됩니다. 빈 세트를 지정하면 오류가 발생합니다.
  - `ADD` - 속성이 아직 없으면 지정된 값을 항목에 추가합니다. 속성이 있으면 `ADD`의 동작은 속성의 데이터 형식에 따라 달라집니다.
    - 기존 속성이 숫자이고 `Value`도 숫자이면 `Value`가 산술적으로 기존 속성에 추가됩니다. `Value`가 음수이면 기존 속성에서 차감됩니다.

> **Note**
>
> ADD를 사용하여 업데이트 전에는 존재하지 않은 항목의 숫자 값을 증가 또는 감소시키는 경우, DynamoDB는 초기 값으로 0을 사용합니다.
> 마찬가지로, ADD를 사용하여 업데이트 전에는 존재하지 않은 속성 값을 증가 또는 감소시키는 경우, DynamoDB는 초기 값으로 0을 사용합니다. 예를 들어, 업데이트하려는 항목에 itemcount라는 속성이 없는데, 그래도 이 속성에 숫자 3을 ADD하려고 한다고 가정합니다. DynamoDB는 itemcount 속성을 만들고, 초기 값을 0으로 설정하고, 마지막으로 3을 더합니다. 결과는 새로운 itemcount 속성이 되며, 그 값은 3이 됩니다.

- 기존 데이터 형식이 세트이고 Value도 세트이면 Value가 기존 세트에 추가됩니다. 예를 들어, 속성 값이 세트 [1,2]이고 ADD 작업이 [3]을 지정하면, 최종 속성 값은 [1,2,3]이 됩니다. 세트 속성에 대한 ADD 작업이 지정되어 있으며 지정된 속성 유형이 기존 세트 유형과 일치하지 않는 경우 오류가 발생합니다.

  두 세트의 프리미티브 데이터 형식이 동일해야 합니다. 예를 들어, 기존 데이터 형식이 문자열 세트인 경우 Value 도 문자열 세트여야 합니다.

지정된 키를 가진 항목이 테이블에 없으면, 다음 값이 다음 작업을 수행합니다.

- PUT - DynamoDB가 지정된 기본 키를 사용하여 새 항목을 만든 다음, 속성을 추가합니다.
- DELETE - 존재하지 않는 항목에서는 속성을 삭제할 수 없으므로 아무 변화가 없습니다. 작업은 성공하지만 DynamoDB가 새 항목을 만들지 않습니다.
- ADD - DynamoDB가 속성 값에 대해 제공된 기본 키와 숫자(또는 숫자 세트)로 항목을 만듭니다. 허용되는 유일한 데이터 형식은 숫자 및 숫자 집합입니다.

인덱스 키의 일부인 속성을 제공하는 경우, 해당 속성의 데이터 형식이 표 속성 설명에 있는 스키마의 데이터 형식과 일치해야 합니다.

## 대신 UpdateExpression 사용 - 예

Music 테이블의 항목을 수정하려고 한다고 가정합니다. 다음 AWS CLI 예제에서와 같이 UpdateItem 요청을 AttributeUpdates 파라미터와 함께 사용할 수 있습니다.

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "SongTitle": {"S":"Call Me Today"},
```

```
        "Artist": {"S":"No One You Know"}
    }' \
    --attribute-updates '{
        "Genre": {
            "Action": "PUT",
            "Value": {"S":"Rock"}
        }
    }'
```

대신에 UpdateExpression을 사용할 수 있습니다.

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "SongTitle": {"S":"Call Me Today"},
        "Artist": {"S":"No One You Know"}
    }' \
    --update-expression 'SET Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

## ConditionalOperator(레거시)

> **Note**
>
> 가능하면 이러한 레거시 파라미터 대신 새 표현식 파라미터를 사용하는 것이 좋습니다. 자세한 내용은 DynamoDB에서 표현식 사용 단원을 참조하십시오.

레거시 조건 파라미터 ConditionalOperator는 Expected, QueryFilter 또는 ScanFilter 맵의 조건에 적용하는 데 사용되는 논리 연산자입니다.

- AND - 모든 조건이 true로 평가되면 전체 맵이 true로 평가됩니다.
- OR - 조건 중 하나라도 true로 평가되면 전체 맵이 true로 평가됩니다.

ConditionalOperator를 생략하는 경우 AND가 기본값입니다.

전체 맵이 true로 평가되는 경우에만 작업이 성공합니다.

> **Note**
>
> 이 파라미터는 목록 또는 맵 유형의 속성을 지원하지 않습니다.

## Expected(레거시)

> **Note**
>
> 가능하면 이러한 레거시 파라미터 대신 새 표현식 파라미터를 사용하는 것이 좋습니다. 자세한 내용은 DynamoDB에서 표현식 사용 단원을 참조하십시오. 이 파라미터를 대체하는 새 파라미터에 대한 자세한 내용은 대신 ConditionExpression을 사용합니다. 섹션을 참조하세요.

레거시 조건 파라미터 `Expected`는 `UpdateItem` 작업의 조건부 블록입니다. `Expected`는 속성/조건 페어의 맵입니다. 각 속성은 속성 이름, 비교 연산자 및 하나 이상의 값으로 구성됩니다. DynamoDB는 비교 연산자를 사용하여 속성을 제공된 값과 비교합니다. 각 `Expected` 요소에 대한 평가 결과는 true 또는 false입니다.

`Expected` 맵의 요소를 다수 지정하는 경우, 기본적으로 모든 조건이 true로 평가되어야 합니다. 즉, 조건은 `AND` 연산자를 사용하여 결합됩니다. (대신 `ConditionalOperator` 파라미터를 사용하여 조건을 OR로 연결할 수 있습니다. 이렇게 하는 경우 모든 조건이 아니라 조건 중 하나 이상이 true로 평가되어야 합니다.)

`Expected` 맵이 true로 평가되면 조건부 작업이 성공하지만, 그렇지 않으면 실패합니다.

`Expected`는 다음을 포함합니다.

- `AttributeValueList` - 제공된 속성에 대해 평가되는 하나 이상의 값. 목록에 있는 값의 개수는 사용되는 `ComparisonOperator`에 따라 달라집니다.

  숫자 유형의 경우 값 비교가 숫자입니다.

  크다, 같다 또는 작다 등의 문자열 값 비교는 UTF-8 이진 인코딩을 사용하는 유니코드를 기준으로 합니다. 예를 들어 `a`는 `A`보다 크고 `a`는 `B`보다 큽니다.

  이진수 유형의 경우, DynamoDB가 이진수 값을 비교할 때 이진수 데이터의 각 바이트를 부호가 없는 것으로 처리합니다.

- ComparisonOperator - AttributeValueList의 속성을 평가하는 비교기입니다. 비교를 수행하면 DynamoDB가 강력한 일관된 읽기(Strongly Consistent Read)를 사용합니다.

  다음 비교 연산자를 사용할 수 있습니다.

  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN

  다음은 각 비교 연산자에 대한 설명입니다.

  - EQ : 같음. EQ는 모든 데이터 형식(목록 및 맵 포함)에 대해 지원됩니다.

    AttributeValueList에는 문자열, 숫자, 이진수, 문자열 집합, 숫자 집합 또는 이진수 집합 유형의 AttributeValue 요소 하나만 포함될 수 있습니다. 요청에서 제공된 형식과 다른 형식의 AttributeValue 요소가 항목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 같지 않습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 같지 않습니다.

  - NE : 같지 않음. NE는 모든 데이터 형식(목록 및 맵 포함)에 대해 지원됩니다.

    AttributeValueList에는 문자열, 숫자, 이진수, 문자열 집합, 숫자 집합 또는 이진수 집합 유형의 AttributeValue 하나만 포함될 수 있습니다. 항목에 요청에서 제공한 형식과 다른 형식의 AttributeValue가 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 같지 않습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 같지 않습니다.

  - LE : 작거나 같음.

    AttributeValueList에는 문자열, 숫자, 이진수 유형의 AttributeValue 요소 하나만 포함될 수 있습니다(집합 유형 제외). 요청에서 제공된 형식과 다른 형식의 AttributeValue 요소가 항목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 같지 않습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 비교할 수 없습니다.

  - LT : 작음.

    AttributeValueList에는 문자열, 숫자, 이진수 유형의 AttributeValue 하나만 포함될 수 있습니다(집합 유형 제외). 요청에서 제공된 형식과 다른 형식의 AttributeValue 요소가 항목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 같지 않습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 비교할 수 없습니다.

  - GE : 크거나 같음.

AttributeValueList에는 문자열, 숫자, 이진수 유형의 AttributeValue 요소 하나만 포함될 수 있습니다(집합 유형 제외). 요청에서 제공된 형식과 다른 형식의 AttributeValue 요소가 항목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 같지 않습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 비교할 수 없습니다.

- GT : 큼.

  AttributeValueList에는 문자열, 숫자, 이진수 유형의 AttributeValue 요소 하나만 포함될 수 있습니다(집합 유형 제외). 요청에서 제공된 형식과 다른 형식의 AttributeValue 요소가 항목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 같지 않습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 비교할 수 없습니다.

- NOT_NULL : 속성 있음. NOT_NULL은 모든 데이터 형식(목록 및 맵 포함)에 대해 지원됩니다.

  > **Note**
  >
  > 이 연산자는 속성의 데이터 형식이 아닌 존재 여부를 테스트합니다. 속성 "a"의 데이터 형식이 null이고 NOT_NULL을 사용하여 평가하는 경우, 결과가 부울 true가 됩니다. 속성 "a"가 존재하기 때문에 이러한 결과가 나오는 것이며, 데이터 형식은 NOT_NULL 비교 연산자와 관련 없습니다.

- NULL : 속성 없음. NULL은 모든 데이터 형식(목록 및 맵 포함)에 대해 지원됩니다.

  > **Note**
  >
  > 이 연산자는 속성의 데이터 형식이 아닌 비존재 여부를 테스트합니다. 속성 "a"의 데이터 형식이 null이고 NULL을 사용하여 평가하는 경우, 결과가 부울 false가 됩니다. 속성 "a"가 존재하기 때문에 이러한 결과가 나오는 것이며, 데이터 형식은 NULL 비교 연산자와 관련 없습니다.

- CONTAINS: 세트의 하위 시퀀스 또는 값 확인.

  AttributeValueList에는 문자열, 숫자, 이진수 유형의 AttributeValue 요소 하나만 포함될 수 있습니다(집합 유형 제외). 비교 대상의 속성이 문자열 유형이라면 연산자가 하위 문자열이 일치하는지 확인합니다. 비교 대상의 속성이 이진수 유형이라면 연산자가 대상에서 입력 이진수와 일치하는 부분 수열을 찾습니다. 비교 대상의 속성이 집합("SS", "NS" 또는 "BS")이라면 집합을 구성하는 어떤 요소든지 정확히 일치하는 것을 찾아야만 연산자가 true로 평가됩니다.

CONTAINS는 목록의 경우 지원됩니다. "a CONTAINS b"를 평가할 때, "a"는 목록이 될 수 있지만, "b"는 집합, 맵 또는 목록이 될 수 없습니다.

- NOT_CONTAINS: 세트의 하위 시퀀스 부재 또는 값의 부재 확인.

  AttributeValueList에는 문자열, 숫자, 이진수 유형의 AttributeValue 요소 하나만 포함될 수 있습니다(집합 유형 제외). 비교 대상의 속성이 문자열이라면 연산자가 하위 문자열이 일치하지 않는지 확인합니다. 비교 대상의 속성이 이진수라면 연산자가 대상에서 입력 이진수와 일치하는 부분 수열이 없는지 확인합니다. 비교 대상의 속성이 집합("SS", "NS" 또는 "BS")이라면 집합을 구성하는 어떤 요소든지 정확히 일치하는 것을 찾지 못해야만(does not) 연산자가 true로 평가됩니다.

  NOT_CONTAINS는 목록의 경우 지원됩니다. "a NOT CONTAINS b"를 평가할 때, "a"는 목록이 될 수 있지만, "b"는 집합, 맵 또는 목록이 될 수 없습니다.

- BEGINS_WITH: 접두사 여부 확인.

  AttributeValueList에는 문자열 또는 이진수 유형의 AttributeValue 하나만 포함될 수 있습니다(숫자 또는 집합 유형 제외). 비교 대상의 속성은 문자열 또는 이진수 유형이 되어야 합니다(숫자 또는 집합 유형 제외).

- IN: 두 집합 안의 일치하는 요소 확인.

  AttributeValueList에는 문자열, 숫자 또는 이진수 유형 중 하나 이상의 AttributeValue 요소가 포함될 수 있습니다(집합 유형 제외). 이 속성들은 기존 집합 유형 항목 속성과 비교됩니다. 입력된 집합 요소 중 하나라도 항목 속성에 있으면 표현식은 true로 평가됩니다.

- BETWEEN: 첫 번째 값보다 크거나 같음 및 두 번째 값보다 작거나 같음.

  AttributeValueList에는 문자열, 숫자 또는 이진수(집합 유형 제외) 중에서 동일한 형식으로 2개의 AttributeValue 요소가 들어 있어야 합니다. 대상 값이 첫 번째 요소보다 크거나 같을 때, 그리고 두 번째 요소보다 작거나 같을 때 대상 속성이 일치합니다. 요청에서 제공된 형식과 다른 형식의 AttributeValue 요소가 항목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 비교할 수 없습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 비교할 수 없습니다.

다음 파라미터는 AttributeValueList 및 ComparisonOperator 대신 사용할 수 있습니다.

- Value - DynamoDB가 속성과 값을 비교할 값.
- Exists - DynamoDB가 조건부 작업을 시도하기 전에 값을 평가하도록 할 부울 값:

- `Exists`가 `true`이면, DynamoDB가 속성 값이 테이블에 이미 있는지를 확인합니다. 있으면, 조건이 true로 평가되고, 없으면 조건이 false로 평가됩니다.
- `Exists`가 `false`이면, DynamoDB가 속성 값이 테이블에 없다고(not) 가정합니다. 값이 실제로 없으면, 가정이 유효하고 조건이 true로 평가됩니다. 값이 있으면, 값이 없다는 가정에도 불구하고 조건이 false로 평가됩니다.

`Exists`의 기본값은 `true`입니다.

`Value` 및 `Exists` 파라미터는 `AttributeValueList` 및 `ComparisonOperator`와 호환되지 않습니다. 두 파라미터 집합을 한 번에 사용하면 DynamoDB에서 `ValidationException` 예외를 반환합니다.

> **Note**
>
> 이 파라미터는 목록 또는 맵 유형의 속성을 지원하지 않습니다.

## 대신 ConditionExpression 사용 - 예

특정 조건이 true인 경우에만 Music 테이블의 항목을 수정하길 원한다고 가정합니다. 다음 AWS CLI 예제에서와 같이 `UpdateItem` 요청을 `Expected` 파라미터와 함께 사용할 수 있습니다.

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }' \
    --attribute-updates '{
        "Price": {
            "Action": "PUT",
            "Value": {"N":"1.98"}
        }
    }' \
    --expected '{
        "Price": {
            "ComparisonOperator": "LE",
            "AttributeValueList": [ {"N":"2.00"} ]
        }
    }'
```

대신에 ConditionExpression을 사용할 수 있습니다.

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }' \
    --update-expression 'SET Price = :p1' \
    --condition-expression 'Price <= :p2' \
    --expression-attribute-values '{
        ":p1": {"N":"1.98"},
        ":p2": {"N":"2.00"}
    }'
```

## KeyConditions(레거시)

> **Note**
>
> 가능하면 이러한 레거시 파라미터 대신 새 표현식 파라미터를 사용하는 것이 좋습니다. 자세한 내용은 DynamoDB에서 표현식 사용 단원을 참조하십시오. 이 파라미터를 대체하는 새 파라미터에 대한 자세한 내용은 대신 KeyConditionExpression을 사용합니다. 섹션을 참조하세요.

레거시 조건 파라미터 KeyConditions에는 Query 작업의 선택 기준이 포함됩니다. 테이블에 있는 쿼리의 경우, 테이블 기본 키 속성에 있는 조건만 가질 수 있습니다. EQ 조건으로 파티션 키 이름 및 값을 제공해야 합니다. 정렬 키의 두 번째 조건은 옵션으로 입력할 수 있습니다.

> **Note**
>
> 정렬 키 조건을 제공하지 않으면 파티션 키가 일치하는 모든 항목이 검색됩니다.
> FilterExpression 또는 QueryFilter가 있는 경우 항목이 검색된 후 적용됩니다.

인덱스에 있는 쿼리의 경우, 인덱스 키 속성에 있는 조건만 가질 수 있습니다. EQ 조건으로 인덱스 파티션 키 이름 및 값을 제공해야 합니다. 인덱스 정렬 키의 두 번째 조건은 옵션으로 입력할 수 있습니다.

각 KeyConditions 요소는 비교할 속성 이름 및 다음 파라미터로 구성됩니다.

- AttributeValueList - 제공된 속성에 대해 평가되는 하나 이상의 값. 목록에 있는 값의 개수는 사용되는 ComparisonOperator에 따라 달라집니다.

  숫자 유형의 경우 값 비교가 숫자입니다.

  크다, 같다 또는 작다 등의 문자열 값 비교는 UTF-8 이진 인코딩을 사용하는 유니코드를 기준으로 합니다. 예를 들어 a는 A보다 크고 a는 B보다 큽니다.

  Binary의 경우, DynamoDB가 이진수 값을 비교할 때 이진수 데이터의 각 바이트를 부호가 없는 것으로 처리합니다.

- ComparisonOperator - 속성을 평가하는 비교기. 예: 같다, 크다, 작다.

  KeyConditions의 경우 다음 비교 연산자만 지원됩니다.

  EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN

  다음은 이러한 비교 연산자에 대한 설명입니다.

  - EQ : 같음.

    AttributeValueList에는 문자열, 숫자, 이진수 유형의 AttributeValue 하나만 포함될 수 있습니다(집합 유형 제외). 요청에서 지정한 형식과 다른 형식의 AttributeValue 요소가 항목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 같지 않습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 같지 않습니다.

  - LE : 작거나 같음.

    AttributeValueList에는 문자열, 숫자, 이진수 유형의 AttributeValue 요소 하나만 포함될 수 있습니다(집합 유형 제외). 요청에서 제공된 형식과 다른 형식의 AttributeValue 요소가 항목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 같지 않습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 비교할 수 없습니다.

  - LT : 작음.

    AttributeValueList에는 문자열, 숫자, 이진수 유형의 AttributeValue 하나만 포함될 수 있습니다(집합 유형 제외). 요청에서 제공된 형식과 다른 형식의 AttributeValue 요소가 항목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 같지 않습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 비교할 수 없습니다.

  - GE : 크거나 같음.

    AttributeValueList에는 문자열, 숫자, 이진수 유형의 AttributeValue 요소 하나만 포함될 수 있습니다(집합 유형 제외). 요청에서 제공된 형식과 다른 형식의 AttributeValue 요소가 항

목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 같지 않습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 비교할 수 없습니다.

- GT : 큼.

  AttributeValueList에는 문자열, 숫자, 이진수 유형의 AttributeValue 요소 하나만 포함될 수 있습니다(집합 유형 제외). 요청에서 제공된 형식과 다른 형식의 AttributeValue 요소가 항목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 같지 않습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 비교할 수 없습니다.

- BEGINS_WITH: 접두사 여부 확인.

  AttributeValueList에는 문자열 또는 이진수 유형의 AttributeValue 하나만 포함될 수 있습니다(숫자 또는 집합 유형 제외). 비교 대상의 속성은 문자열 또는 이진수 유형이 되어야 합니다(숫자 또는 집합 유형 제외).

- BETWEEN: 첫 번째 값보다 크거나 같음 및 두 번째 값보다 작거나 같음.

  AttributeValueList에는 문자열, 숫자 또는 이진수(집합 유형 제외) 중에서 동일한 형식으로 2개의 AttributeValue 요소가 들어 있어야 합니다. 대상 값이 첫 번째 요소보다 크거나 같을 때, 그리고 두 번째 요소보다 작거나 같을 때 대상 속성이 일치합니다. 요청에서 제공된 형식과 다른 형식의 AttributeValue 요소가 항목에 포함되어 있으면 값이 일치하지 않습니다. 예를 들어 {"S":"6"}은 {"N":"6"}과 비교할 수 없습니다. 또한 {"N":"6"}은 {"NS":["6", "2", "1"]}과 비교할 수 없습니다.

## 대신 KeyConditionExpression 사용 - 예

Music 테이블의 파티션 키와 동일한 파티션 키가 있는 여러 항목을 검색하길 원한다고 가정합니다. 다음 AWS CLI 예제에서와 같이 Query 요청을 KeyConditions 파라미터와 함께 사용할 수 있습니다.

```
aws dynamodb query \
    --table-name Music \
    --key-conditions '{
        "Artist":{
            "ComparisonOperator":"EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        },
        "SongTitle":{
            "ComparisonOperator":"BETWEEN",
            "AttributeValueList": [ {"S": "A"}, {"S": "M"} ]
        }
```

```
    }'
```

대신에 KeyConditionExpression을 사용할 수 있습니다.

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a AND SongTitle BETWEEN :t1 AND :t2' \
    --expression-attribute-values '{
        ":a": {"S": "No One You Know"},
        ":t1": {"S": "A"},
        ":t2": {"S": "M"}
    }'
```

## QueryFilter(레거시)

> **Note**
>
> 가능하면 이러한 레거시 파라미터 대신 새 표현식 파라미터를 사용하는 것이 좋습니다. 자세한 내용은 DynamoDB에서 표현식 사용 단원을 참조하십시오. 이 파라미터를 대체하는 새 파라미터에 대한 자세한 내용은 대신 FilterExpression을 사용합니다. 섹션을 참조하세요.

Query 작업에서 레거시 조건 파라미터 QueryFilter는 항목이 읽히고 원하는 값만 반환한 후 쿼리 결과를 평가하는 조건입니다.

이 파라미터는 목록 또는 맵 유형의 속성을 지원하지 않습니다.

> **Note**
>
> QueryFilter는 항목이 이미 읽힌 후에 적용되므로 필터링 프로세스로 인해 읽기 용량 단위를 추가로 사용하지 않습니다.

QueryFilter 맵에 조건을 다수 제공하는 경우, 기본적으로 모든 조건이 true로 평가됩니다. 즉, 조건은 AND 연산자를 사용하여 결합됩니다. (대신 ConditionalOperator(레거시) 파라미터를 사용하여 조건을 OR로 연결할 수 있습니다. 이렇게 하는 경우 모든 조건이 아니라 조건 중 하나 이상이 true로 평가되어야 합니다.)

QueryFilter에는 키 속성을 사용할 수 없습니다. 파티션 키 또는 정렬 키에는 필터 조건을 정의할 수 없습니다.

각 QueryFilter 요소는 비교할 속성 이름 및 다음 파라미터로 구성됩니다.

- AttributeValueList - 제공된 속성에 대해 평가되는 하나 이상의 값. 목록에 있는 값의 개수는 ComparisonOperator에 지정된 연산자에 따라 달라집니다.

  숫자 유형의 경우 값 비교가 숫자입니다.

  크다, 같다 또는 작다 등의 문자열 값 비교는 UTF-8 이진 인코딩을 기준으로 합니다. 예를 들어 a는 A보다 크고 a는 B보다 큽니다.

  이진수 유형의 경우, DynamoDB가 이진수 값을 비교할 때 이진수 데이터의 각 바이트를 부호가 없는 것으로 처리합니다.

  JSON에 데이터 형식을 지정하는 방법에 대한 자세한 내용은 DynamoDB 하위 수준 API 단원을 참조하십시오.

- ComparisonOperator - 속성을 평가하는 비교기. 예: 같다, 크다, 작다.

  다음 비교 연산자를 사용할 수 있습니다.

  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN

## 대신 FilterExpression 사용 - 예

Music 테이블을 쿼리하는데 일치하는 항목에 조건을 적용하고 싶다고 가정합니다. 다음 AWS CLI 예제에서와 같이 Query 요청을 QueryFilter 파라미터와 함께 사용할 수 있습니다.

```
aws dynamodb query \
    --table-name Music \
    --key-conditions '{
        "Artist": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        }
    }' \
    --query-filter '{
        "Price": {
            "ComparisonOperator": "GT",
            "AttributeValueList": [ {"N": "1.00"} ]
        }
```

```
    }'
```

대신에 `FilterExpression`을 사용할 수 있습니다.

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a' \
    --filter-expression 'Price > :p' \
    --expression-attribute-values '{
        ":p": {"N":"1.00"},
        ":a": {"S":"No One You Know"}
    }'
```

## ScanFilter(레거시)

> **Note**
>
> 가능하면 이러한 레거시 파라미터 대신 새 표현식 파라미터를 사용하는 것이 좋습니다. 자세한 내용은 DynamoDB에서 표현식 사용 단원을 참조하십시오. 이 파라미터를 대체하는 새 파라미터에 대한 자세한 내용은 대신 FilterExpression을 사용합니다. 섹션을 참조하세요.

`Scan` 작업에서 레거시 조건 파라미터 `ScanFilter`는 원하는 값만 반환한 후 쿼리 결과를 평가하는 조건입니다.

> **Note**
>
> 이 파라미터는 목록 또는 맵 유형의 속성을 지원하지 않습니다.

`ScanFilter` 맵에 조건을 다수 지정하는 경우, 기본적으로 모든 조건이 true로 평가됩니다. 즉, 조건이 모두 AND로 연결됩니다. (대신 ConditionalOperator(레거시) 파라미터를 사용하여 조건을 OR로 연결할 수 있습니다. 이렇게 하는 경우 모든 조건이 아니라 조건 중 하나 이상이 true로 평가되어야 합니다.)

각 `ScanFilter` 요소는 비교할 속성 이름 및 다음 파라미터로 구성됩니다.

- `AttributeValueList` - 제공된 속성에 대해 평가되는 하나 이상의 값. 목록에 있는 값의 개수는 `ComparisonOperator`에 지정된 연산자에 따라 달라집니다.

  숫자 유형의 경우 값 비교가 숫자입니다.

크다, 같다 또는 작다 등의 문자열 값 비교는 UTF-8 이진 인코딩을 기준으로 합니다. 예를 들어 a는 A보다 크고 a는 B보다 큽니다.

Binary의 경우, DynamoDB가 이진수 값을 비교할 때 이진수 데이터의 각 바이트를 부호가 없는 것으로 처리합니다.

JSON에 데이터 형식을 지정하는 방법에 대한 자세한 내용은 DynamoDB 하위 수준 API 단원을 참조하십시오.

- `ComparisonOperator` - 속성을 평가하는 비교기. 예: 같다, 크다, 작다.

  다음 비교 연산자를 사용할 수 있습니다.

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

## 대신 FilterExpression 사용 - 예

Music 테이블을 검색하는데 일치하는 항목에 조건을 적용하고 싶다고 가정합니다. 다음 AWS CLI 예제에서와 같이 `Scan` 요청을 `ScanFilter` 파라미터와 함께 사용할 수 있습니다.

```
aws dynamodb scan \
    --table-name Music \
    --scan-filter '{
        "Genre":{
            "AttributeValueList":[ {"S":"Rock"} ],
            "ComparisonOperator": "EQ"
        }
    }'
```

대신에 `FilterExpression`을 사용할 수 있습니다.

```
aws dynamodb scan \
    --table-name Music \
    --filter-expression 'Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

# 기존 파라미터를 이용한 조건 작성

> **Note**
>
> 가능하면 이러한 레거시 파라미터 대신 새 표현식 파라미터를 사용하는 것이 좋습니다. 자세한 내용은 DynamoDB에서 표현식 사용 단원을 참조하십시오.

다음 단원에서는 `Expected`, `QueryFilter`, `ScanFilter` 등 기존 파라미터를 사용해 조건을 작성하는 방법에 대해서 살펴봅니다.

> **Note**
>
> 그 외에 새로운 애플리케이션은 표현식 파라미터를 사용해야 합니다. 자세한 내용은 DynamoDB에서 표현식 사용 단원을 참조하십시오.

## 단순 조건

속성 값만 알고 있으면 직접 조건을 작성하여 테이블 속성과 비교할 수 있습니다. 조건의 평가 방법은 항상 true 또는 false이며, 다음과 같이 구성됩니다.

- `ComparisonOperator` - 초과, 미만, 같음 등
- `AttributeValueList`(선택 사항) - 비교할 속성 값. 사용하는 `ComparisonOperator`에 따라 `AttributeValueList`에는 값이 1개, 2개 또는 그 이상이 저장되거나, 하나도 저장되지 않을 수 있습니다.

다음 단원에서는 이러한 파라미터를 조건에 사용하는 몇 가지 예제와 함께 여러 비교 연산자에 대해서 설명합니다.

속성 값이 없는 비교 연산자

- `NOT_NULL` - 속성이 존재하는 경우 true
- `NULL` - 속성이 존재하지 않는 경우 true.

위 연산자는 속성의 존재 여부를 확인하는 데 사용됩니다. 비교할 값이 없기 때문에 `AttributeValueList`를 지정할 필요도 없습니다.

예

다음은 Dimensions 속성이 존재하는 경우 true로 평가되는 표현식입니다.

```
...
    "Dimensions": {
         ComparisonOperator: "NOT_NULL"
    }
...
```

속성 값이 하나인 비교 연산자

- EQ - 속성이 값과 같으면 true

  AttributeValueList에는 값이 하나만 저장되며, 이 값의 형식으로는 문자열, 숫자, 이진수, 문자열 집합, 숫자 집합 또는 이진수 집합이 될 수 있습니다. 요청에서 지정한 형식과 다른 형식의 값이 항목에 저장되면 값은 일치하지 않습니다. 예를 들어 문자열 "3"은 숫자 3과 다릅니다. 또한 숫자 3은 숫자 집합 [3, 2, 1]과 다릅니다.
- NE - 속성이 값과 같지 않으면 true

  AttributeValueList에는 값이 하나만 저장되며, 이 값의 형식으로는 문자열, 숫자, 이진수, 문자열 집합, 숫자 집합 또는 이진수 집합이 될 수 있습니다. 요청에서 지정한 형식과 다른 형식의 값이 항목에 저장되면 값은 일치하지 않습니다.
- LE - 속성이 값보다 작거나 같으면 true

  AttributeValueList에는 값이 하나만 저장되며, 이 값의 형식으로는 문자열, 숫자, 이진수가 될 수 있습니다(집합 제외). 요청에서 지정한 형식과 다른 형식의 AttributeValue 값이 항목에 저장되면 값은 일치하지 않습니다.
- LT - 속성이 값보다 작으면 true

  AttributeValueList에는 값이 하나만 저장되며, 이 값의 형식으로는 문자열, 숫자, 이진수가 될 수 있습니다(집합 제외). 요청에서 지정한 형식과 다른 형식의 값이 항목에 저장되면 값은 일치하지 않습니다.
- GE - 속성이 값보다 크거나 같으면 true

  AttributeValueList에는 값이 하나만 저장되며, 이 값의 형식으로는 문자열, 숫자, 이진수가 될 수 있습니다(집합 제외). 요청에서 지정한 형식과 다른 형식의 값이 항목에 저장되면 값은 일치하지 않습니다.
- GT - 속성이 값보다 크면 true

AttributeValueList에는 값이 하나만 저장되며, 이 값의 형식으로는 문자열, 숫자, 이진수가 될 수 있습니다(집합 제외). 요청에서 지정한 형식과 다른 형식의 값이 항목에 저장되면 값은 일치하지 않습니다.

- CONTAINS - 값이 집합 내에 저장되거나, 하나의 값이 다른 값에 저장되는 경우 true

  AttributeValueList에는 값이 하나만 저장되며, 이 값의 형식으로는 문자열, 숫자, 이진수가 될 수 있습니다(집합 제외). 비교 대상의 속성이 문자열이라면 연산자가 하위 문자열이 일치하는지 확인합니다. 비교 대상의 속성이 이진수라면 연산자가 대상에서 입력 이진수와 일치하는 부분 수열을 찾습니다. 그리고, 비교 대상의 속성이 집합이라면 집합을 구성하는 어떤 요소든지 정확히 일치하는 것을 찾아야만 연산자가 true로 평가됩니다.
- NOT_CONTAINS - 값이 집합 내에 저장되지 않거나, 혹은 하나의 값이 다른 값에 저장되지 않는 경우 true

  AttributeValueList에는 값이 하나만 저장되며, 이 값의 형식으로는 문자열, 숫자, 이진수가 될 수 있습니다(집합 제외). 비교 대상의 속성이 문자열이라면 연산자가 하위 문자열이 일치하지 않는지 확인합니다. 비교 대상의 속성이 이진수라면 연산자가 대상에서 입력 이진수와 일치하는 부분 수열이 없는지 확인합니다. 그리고, 비교 대상의 속성이 집합이라면 집합을 구성하는 어떤 항이든지 정확히 일치하는 것을 찾지 못할 때 연산자가 true로 평가됩니다.
- BEGINS_WITH - 속성 중 첫 번째 일부 문자가 입력한 값과 일치하는 경우 true 숫자를 비교할 때는 이 연산자를 사용할 수 없습니다.

  AttributeValueList에 문자열 또는 이진수(숫자 또는 집합 제외) 중 한 가지 형식의 값만 저장됩니다. 비교 대상의 속성은 문자열 또는 이진수가 되어야 합니다(숫자 또는 집합 제외).

위의 연산자들은 모두 속성과 값을 비교하는 데 사용됩니다. AttributeValueList는 단일 값으로 구성하여 지정해야 합니다. 대부분 연산자에서 이 값은 스칼라가 되어야 하지만 EQ와 NE 연산자는 집합도 지원합니다.

예시

다음과 같은 경우에 한해 아래 표현식들은 true로 평가됩니다.

- 제품 가격이 100보다 큰 경우

```
...
    "Price": {
        ComparisonOperator: "GT",
        AttributeValueList: [ {"N":"100"} ]
```

```
    }
...
```

- 제품 카테고리가 "Bo"로 시작되는 경우

```
...
    "ProductCategory": {
        ComparisonOperator: "BEGINS_WITH",
        AttributeValueList: [ {"S":"Bo"} ]
    }
...
```

- 제품이 빨간색, 녹색 또는 검은색으로 출시되는 경우

```
...
    "Color": {
        ComparisonOperator: "EQ",
        AttributeValueList: [
            [ {"S":"Black"}, {"S":"Red"}, {"S":"Green"} ]
        ]
    }
...
```

> **Note**
>
> 집합 데이터 형식을 비교할 때 요소의 순서는 중요하지 않습니다. DynamoDB는 요청에서 지정한 순서와 상관없이 동일한 값 집합이 포함된 항목만 반환합니다.

속성 값이 2개인 비교 연산자

- `BETWEEN` - 값이 엔드포인트를 포함해 하한선과 상한선 사이에 있는 경우 true.

  `AttributeValueList`에는 문자열, 숫자 또는 이진수(집합 제외) 중 동일한 형식의 요소 2개가 저장되어야 합니다. 대상 값이 첫 번째 요소보다 크거나 같을 때, 그리고 두 번째 요소보다 작거나 같을 때 대상 속성이 일치합니다. 요청에서 지정한 형식과 다른 형식의 값이 항목에 저장되면 값은 일치하지 않습니다.

위 연산자는 속성 값이 범위를 벗어나지 않았는지 확인할 때 사용됩니다. `AttributeValueList`에는 문자열, 숫자 또는 이진수 중 동일한 형식의 스칼라 요소 2개가 저장되어야 합니다.

예

다음은 제품 가격이 100에서 200 사이인 경우에 true로 평가되는 표현식입니다.

```
...
    "Price": {
        ComparisonOperator: "BETWEEN",
        AttributeValueList: [ {"N":"100"}, {"N":"200"} ]
    }
...
```

속성 값이 n개인 비교 연산자

- `IN` - 값이 열거 목록의 값 중 하나라도 같으면 true. 이 목록에서는 집합이 아닌 스칼라 값만 지원됩니다. 대상 속성이 일치하려면 형식이 동일하고 값도 정확해야 합니다.

  `AttributeValueList`에는 문자열, 숫자 또는 이진수(집합 제외) 중 한 가지 형식의 요소가 하나 이상 저장됩니다. 이 속성들은 집합을 제외한 형식의 기존 항목 속성과 비교됩니다. 입력된 집합 요소 중 하나라도 항목 속성에 있으면 표현식은 true로 평가됩니다.

  `AttributeValueList`에는 문자열, 숫자 또는 이진수(집합 제외) 중 한 가지 형식의 값이 하나 이상 저장됩니다. 비교 대상 속성이 일치하려면 형식이 동일하고 값도 정확해야 합니다. 문자열은 문자열 세트와 일치하지 않습니다.

위 연산자는 열거 목록 내 입력 값의 유무를 확인할 때 사용됩니다. `AttributeValueList`에서 스칼라 값은 얼마든지 지정할 수 있지만 모두 데이터 형식이 동일해야 합니다.

예

다음은 Id 값이 201, 203 또는 205인 경우 true로 평가되는 표현식입니다.

```
...
    "Id": {
        ComparisonOperator: "IN",
        AttributeValueList: [ {"N":"201"}, {"N":"203"}, {"N":"205"} ]
    }
...
```

## 다수의 조건 사용

DynamoDB에서는 다수의 조건을 결합하여 복잡한 표현식을 작성할 수 있습니다. ConditionalOperator(레거시)를 옵션으로 최소 2개 이상의 표현식을 입력하면 가능합니다.

기본적으로 조건을 다수 지정할 때 전체 표현식이 true로 평가되려면 모든 조건이 true로 평가되어야 합니다. 다시 말해서 묵시적으로 AND 연산이 발생합니다.

예

다음은 제품이 최소 600페이지가 넘는 서적인 경우에 true로 평가되는 표현식입니다. 두 조건 모두 묵시적으로 AND가 사용되기 때문에 true로 평가되어야 합니다.

```
...
    "ProductCategory": {
        ComparisonOperator: "EQ",
        AttributeValueList: [ {"S":"Book"} ]
    },
    "PageCount": {
        ComparisonOperator: "GE",
        AttributeValueList: [ {"N":600} ]
    }
...
```

ConditionalOperator(레거시)는 AND 연산의 발생 여부를 명확히 할 때 사용됩니다. 다음 예제는 위 예제와 똑같이 동작합니다.

```
...
    "ConditionalOperator" : "AND",
    "ProductCategory": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"N":"Book"} ]
    },
    "PageCount": {
        "ComparisonOperator": "GE",
        "AttributeValueList": [ {"N":600} ]
    }
...
```

그 밖에도 `ConditionalOperator`를 OR로 설정할 수 있습니다. 이 말은 조건 중 적어도 하나는 true로 평가되어야 한다는 의미입니다.

예

다음은 제품이 산악 자전거인 경우, 특정 브랜드 이름인 경우, 혹은 가격이 100보다 큰 경우에 true로 평가되는 표현식입니다.

```
...
    ConditionalOperator : "OR",
    "BicycleType": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Mountain" ]
    },
    "Brand": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Brand-Company A" ]
    },
    "Price": {
        "ComparisonOperator": "GT",
        "AttributeValueList": [ {"N":"100"} ]
    }
...
```

> **Note**
>
> 복잡한 표현식에서는 첫 번째 조건부터 마지막 조건까지 차례대로 처리됩니다.
> 단일 표현식에서는 AND와 OR를 함께 사용할 수 없습니다.

## 기타 조건부 연산자

이전 DynamoDB 버전에서는 `Expected` 파라미터의 조건부 쓰기 동작이 달랐습니다. `Expected` 맵의 각 항목은 다음 파라미터와 함께 DynamoDB가 확인하는 속성 이름을 나타냈습니다.

- `Value` - 속성과 비교할 값
- `Exists` - 작업에 앞서 값의 존재 여부를 확인

`Value` 및 `Exists` 옵션은 앞으로도 DynamoDB에서 지원되지만 등식 조건이나 속성의 존재 여부를 테스트하는 용도로만 사용할 수 있습니다. 따라서 대신에 `ComparisonOperator`와 `AttributeValueList` 사용을 권장합니다. 이 두 가지 옵션은 훨씬 광범위한 조건을 작성할 수 있기 때문입니다.

Example

DeleteItem은 서적의 출판 중단 여부를 확인하며, 이 조건이 true인 경우에 한해 해당 항목을 삭제할 수 있습니다. 다음은 기존 조건을 사용하는 AWS CLI 예입니다.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "Exists": true,
            "Value": {"BOOL":false}
        }
    }'
```

다음은 위 예제와 동일하지만 기존 조건을 사용하지 않는 예제입니다.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"BOOL":false} ]
        }
    }'
```

Example

PutItem 작업은 동일한 기본 키 속성으로 기존 항목을 덮어쓰지 못하도록 보호합니다. 아래 예제에서는 기존 조건을 사용하고 있습니다.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
```

```
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "Exists": false }
    }'
```

다음은 위 예제와 동일하지만 기존 조건을 사용하지 않는 예제입니다.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "ComparisonOperator": "NULL" }
    }'
```

> **Note**
>
> `Expected` 맵의 조건일 때는 기존 `Value` 및 `Exists` 옵션을 `ComparisonOperator` 및 `AttributeValueList`와 함께 사용할 수 없습니다. 함께 사용하면 조건부 쓰기가 오류를 일으킵니다.