



Panduan Developer

Amazon DynamoDB



Versi API 2012-08-10

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon DynamoDB: Panduan Developer

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan antara para pelanggan, atau dengan cara apa pun yang menghina atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan milik dari pemiliknya masing-masing, yang mungkin berafiliasi dengan, terhubung ke, atau disponsori oleh Amazon.

Table of Contents

Apa itu Amazon DynamoDB?	1
Ketersediaan tinggi dan daya tahan	1
Mulai menggunakan DynamoDB	2
DynamoDB tutorial	3
Cara kerjanya	3
Kisi-kisi	4
Komponen inti	9
API DynamoDB	18
Jenis data yang didukung dan aturan penamaan	22
Konsistensi baca	29
Mode kapasitas baca/tulis	30
Kelas tabel	39
Partisi dan distribusi data	40
Dari SQL ke NoSQL	44
Relasional atau NoSQL?	45
Karakteristik basis data	48
Membuat tabel	51
Mendapatkan informasi tentang tabel	54
Menulis data ke tabel	55
Membaca data dari tabel	60
Mengelola indeks	69
Memodifikasi data dalam tabel	75
Menghapus data dari tabel	79
Menghapus tabel	81
Sumber daya tambahan untuk Amazon DynamoDB	82
Alat untuk pengodean dan visualisasi	82
Panduan Preskriptif	83
Pusat Pengetahuan	84
Posting blog, repositori, dan panduan	85
Pemodelan data dan pola desain	85
Kursus pelatihan	86
Menyiapkan DynamoDB	87
Menyiapkan DynamoDB lokal (versi yang dapat diunduh)	87
Men-deploy	88

Catatan penggunaan	95
Riwayat rilis	99
Telemetri lokal DynamoDB	104
Menyiapkan DynamoDB (layanan web)	107
Mendaftar untuk AWS	107
Memberikan akses terprogram	108
Mengonfigurasi kredensial Anda	109
Integrasi dengan layanan DynamoDB lainnya	110
Mengakses DynamoDB	111
Menggunakan konsol	111
Menggunakan AWS CLI	112
Mengunduh dan mengonfigurasi AWS CLI	113
Menggunakan AWS CLI dengan DynamoDB	113
Menggunakan AWS CLI dengan DynamoDB lokal	114
Menggunakan API	115
Menggunakan NoSQL Workbench	115
Rentang alamat IP	117
Mulai menggunakan DynamoDB	118
Konsep dasar	118
Prasyarat	118
Langkah 2: Buat tabel	119
Langkah 2: Tulis data	125
Langkah 3: Baca data	131
Langkah 4: Perbarui data	135
Langkah 5: Cari data	138
Langkah 6: Buat indeks sekunder global	142
Langkah 7: Kueri indeks sekunder global	147
Langkah 4: (Opsional) hapus	151
Langkah selanjutnya	152
Memulai dengan DynamoDB dan SDK AWS	153
Membuat tabel	153
Membuat tabel DynamoDB menggunakan SDK AWS	153
Menulis item	197
Menulis item pada tabel DynamoDB menggunakan SDK AWS	197
Membaca item	221
Membaca item dari tabel DynamoDB menggunakan SDK AWS	222

Memperbarui item	244
Memperbarui item dalam tabel DynamoDB menggunakan SDK AWS	244
Menghapus item	270
Menghapus item dalam tabel DynamoDB menggunakan SDK AWS	270
Mengkueri tabel	291
Kueri tabel DynamoDB menggunakan SDK AWS	292
Memindai tabel	323
Memindai tabel DynamoDB menggunakan SDK AWS	292
Bekerja dengan AWS SDK	347
Pemrograman dengan DynamoDB	349
Ikhtisar dukungan AWS SDK untuk DynamoDB	349
Antarmuka terprogram	352
API tingkat rendah	359
Penanganan kesalahan	365
Antarmuka pemrograman tingkat tinggi	373
Java 1.x: DynamoDBMapper	374
Java 2.x: DynamoDB Ditingkatkan Klien	445
.NET: Model dokumen	445
.NET: Model persistensi objek	478
Menjalankan Contoh kode	520
Muat data sampel sampel sampel	521
Contoh kode Java	521
Contoh kode .NET	524
Pemrograman dengan Python	528
Tentang Boto	528
Dokumentasi Boto	529
Lapisan klien dan sumber daya	529
Menggunakan batch_writer	533
Contoh kode tambahan	533
Sesi dan keamanan utas	534
Config	534
Penanganan kesalahan	539
Pencatatan log	541
Kait acara	543
Pagination dan Paginator	544
Pelayan	546

Pemrograman dengan JavaScript	546
Tentang AWS SDK for JavaScript	547
AWS SDK for JavaScript V3	547
JavaScript dokumentasi	548
Lapisan abstraksi	548
Fungsi utilitas Marshall	551
Membaca item	551
Penulisan bersyarat	553
Paginasi	554
Config	556
Pelayan	559
Penanganan kesalahan	560
Pencatatan log	562
Pertimbangan	563
Pemrograman dengan Java 2.x	564
Tentang AWS SDK for Java 2.x	564
Memulai	565
Dokumentasi SDK for Java 2.x	575
Antarmuka yang didukung	575
Contoh kode tambahan	590
Sinkronisasi dan pemrograman async	590
Klien HTTP	591
Config	592
Penanganan kesalahan	600
AWS ID permintaan	601
Pencatatan log	601
Paginasi	604
Anotasi kelas data	605
Bekerja dengan DynamoDB	606
Bekerja dengan tabel	606
Operasi dasar pada tabel	607
Pertimbangan saat mengubah Mode Kapasitas baca/tulis	616
Pertimbangan saat memilih kelas tabel	617
Tabel kapasitas yang disediakan	619
Ukuran dan format item	628
Mengelola kapasitas throughput dengan penskalaan otomatis	630

Penandaan sumber daya	653
Bekerja dengan tabel: Java	659
Bekerja dengan tabel: .NET	666
Menggunakan tabel global	675
Mereplikasi data secara mulus di Wilayah dengan tabel global	677
Memberikan keamanan dan akses untuk tabel global Anda dengan AWS KMS	678
Cara kerjanya	679
Praktik terbaik dan persyaratan	684
Tutorial: Membuat tabel global	687
Memantau tabel global	693
Menggunakan IAM dengan tabel global	693
Menentukan versi	697
Meningkatkan tabel global	699
Bekerja dengan operasi baca dan tulis	709
DynamoDB API	709
Bahasa kueri PartiQL	909
Bekerja dengan indeks	956
Indeks sekunder global	961
Indeks Sekunder Lokal	1023
Bekerja dengan transactions	1077
Cara kerjanya	1078
Menggunakan IAM dengan transactions	1087
Kode contoh	1090
Bekerja dengan stream	1094
Opsi	1095
Bekerja dengan Kinesis Data Streams	1097
Bekerja dengan DynamoDB Streams	1115
Bekerja dengan pencadangan Sesuai Permintaan	1176
MenggunakanAWSCadangan	1177
Menggunakan pencadangan DynamoDB	1187
Bekerja dengan point-in-time pemulihan	1207
Cara kerjanya	1208
Sebelum Anda memulai	1211
Memulihkan tabel ke titik waktu	1211
Akselerasi dalam memori dengan DAX	1218
Kasus penggunaan untuk DAX	1219

Catatan penggunaan DAX	1220
Cara kerjanya	1221
Cara DAX Memproses Permintaan	1223
Cache Item	1224
Cache kueri	1225
Komponen DAX	1226
Simpul	1227
klaster	1227
Wilayah dan Availability Zone	1228
Grup parameter	1229
Grup keamanan	1229
klaster ARN	1230
Titik akhir klaster	1230
Titik akhir simpul	1230
Grup subnet	1231
Kejadian	1231
Jendela pemeliharaan	1231
Membuat klaster DAX	1233
Membuat peran layanan IAM untuk DAX untuk mengakses DynamoDB	1233
Menggunakan AWS CLI	1235
Menggunakan konsol	1241
Model Konsistensi	1246
Konsistensi di antara node cluster DAX	1247
perilaku cache item DAX	1247
perilaku cache kueri DAX	1251
Bacaan yang sangat konsisten dan transaksional	1252
Caching negatif	1252
Strategi untuk Penulis	1253
Mengembangkan dengan Klien DAX	1256
Tutorial: Menjalankan aplikasi sampel	1256
Memodifikasi aplikasi yang ada untuk menggunakan DAX	1305
Mengelola klaster DAX	1306
Izin IAM untuk mengelola klaster DAX	1306
Penskalaan klaster DAX	1309
Menyesuaikan Pengaturan klaster DAX	1310
Mengkonfigurasi Pengaturan TTL	1312

Support pemberian tag untuk DAX	1313
AWS CloudTrailIntegrasi	1314
Menghapus klaster DAX	1315
Pemantauan DAX	1315
Alat pemantauan	1315
Pemantauan CloudWatch dengan	1317
Logging operasi DAX menggunakan AWS CloudTrail	1343
Instans Burstable DAX T3/T2	1343
Rangkaian instans DAX T2	1344
Rangkaian instans DAX T3	1344
Kontrol akses DAX	1345
Peran layanan IAM untuk DAX	1346
Kebijakan IAM untuk mengizinkan akses klaster DAX	1348
Studi Kasus: Mengakses DynamoDB dan DAX	1349
Akses ke DynamoDB, tetapi tidak ada akses ke DAX	1350
Akses ke DynamoDB dan DAX	1352
Akses ke DynamoDB melalui DAX, tetapi tidak ada akses langsung ke DynamoDB	1358
Enkripsi DAX saat istirahat	1360
Mengaktifkan enkripsi saat istirahat menggunakanAWS Management Console	1362
Enkripsi DAX dalam transit	1363
Menggunakan peran tertaut layanan untuk DAX	1364
Izin peran tertaut layanan untuk DAX	1365
Membuat peran tertaut layanan untuk DAX	1366
Mengedit peran tertaut layanan untuk DAX	1366
Menghapus peran tertaut layanan untuk DAX	1367
Mengakses DAX di seluruh akun AWS	1368
Mengatur IAM	1369
Siapkan VPC	1372
Ubah klien DAX untuk memungkinkan akses lintas akun	1374
Panduan ukuran cluster DAX	1378
Gambaran Umum	1379
Memperkirakan lalu lintas	1379
Pengujian beban	1380
Referensi API	1382
Pemodelan data	1383
Fondasi pemodelan data	1384

Desain tabel tunggal	1385
Desain multitabel	1388
Blok bangunan pemodelan data	1389
Kunci urutan komposit	1390
Multi-penghunian	1392
Indeks jarang	1393
Waktu untuk beroperasi	1394
Pengarsipan waktu untuk beroperasi	1395
Partisi vertikal	1396
Sharding penulisan	1399
Paket desain skema pemodelan data	1400
Prasyarat	1401
Jejaring sosial	1402
Profil game	1411
Sistem manajemen keluhan	1420
Pembayaran berulang	1437
Pembaruan status perangkat	1442
Toko online	1457
Migrasi ke DynamoDB	1482
Alasan untuk bermigrasi	1482
Pertimbangan saat bermigrasi	1483
Cara kerjanya	1485
Alat Migrasi	1487
Memilih strategi migrasi	1487
Migrasi offline	1491
Migrasi hibrid	1493
Online - memigrasikan setiap tabel 1:1	1494
Online - migrasi dengan tabel penahapan khusus	1495
NoSQL Workbench	1499
Unduh	1500
Menginstal	1501
Pemodel data	1505
Membuat model baru	1506
Mengimpor model yang ada	1514
Mengekspor model	1516
Mengedit model yang ada	1518

Visualisasi data	1522
Menambahkan data	1522
Mengimpor dari CSV	1525
Faset	1526
Tampilan agregat	1529
Melakukan model data	1530
Pembangun operasi	1533
Menghubungkan ke set data	1534
Membangun operasi	1539
Tabel kloning	1563
Mengekspor ke CSV	1566
Model data Sampel	1567
Model data karyawan	1567
Model data diskusi	1568
Model data musik	1568
Model data ski	1569
Kartu kredit menawarkan model data	1569
Model data bookmark	1570
Riwayat rilis	1570
Contoh kode	1576
Tindakan	1584
Buat tabel	1585
Hapus tabel	1628
Menghapus item dalam tabel	1643
Mendapatkan batch item	1665
Mendapatkan item dari tabel	1687
Dapatkan informasi tentang tabel	1710
Mencantumkan tabel	1725
Menempatkan item dalam tabel	1742
Mengkueri Tabel	1766
Menjalankan pernyataan PartiQL	1798
Jalankan batch pernyataan PartiQL	1820
Memindai tabel	1846
Memperbarui item dalam tabel	1870
Menulis sekumpulan item	1896
Skenario	1924

Mempercepat pembacaan dengan DAX	1925
Memulai tabel, item, dan kueri	1933
Melakukan kueri pada tabel menggunakan batch pernyataan PartiQL	2083
Melakukan kueri tabel menggunakan PartiQL	2142
Menggunakan model dokumen	2195
Menggunakan model persistensi objek tingkat tinggi	2211
Contoh lintas layanan	2221
Membangun aplikasi untuk mengirimkan data ke tabel DynamoDB	2221
Membuat API REST untuk melacak data COVID-19	2223
Membuat aplikasi messenger	2224
Membuat aplikasi nirserver untuk mengelola foto	2225
Membuat aplikasi web untuk melacak data DynamoDB	2229
Membuat aplikasi obrolan websocket	2231
Mendeteksi APD dalam gambar	2232
Menginvokasi fungsi Lambda dari browser	2233
Menyimpan EXIF dan informasi gambar lainnya	2234
Menggunakan API Gateway untuk menginvokasi fungsi Lambda	2235
Menggunakan Step Functions untuk menginvokasi fungsi Lambda	2237
Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda	2238
Keamanan	2240
AWS kebijakan terkelola	2241
AWS kebijakan terkelola	2241
AmazonDynamoDB ReadOnlyAccess	2242
DynamoDB memperbarui kebijakan terkelola AWS	2243
Kebijakan berbasis sumber daya	2244
Membuat tabel	2245
Lampirkan kebijakan berbasis sumber daya	2251
Lampirkan kebijakan ke aliran	2256
Hapus kebijakan berbasis sumber daya	2259
Akses lintas akun	2259
Memblokir akses publik	2261
Tindakan IAM	2264
Otorisasi IAM	2269
Contoh-contoh	2270
Pertimbangan	2276
Praktik terbaik	2277

Perlindungan data	2279
Enkripsi saat diam	2279
Perlindungan data di DAX	2306
Privasi lalu lintas antarjaringan	2306
IAM	2308
Manajemen Identitas dan Akses	2308
Ketentuan penggunaan	2344
Manajemen identitas dan akses di DAX	2368
Validasi kepatuhan	2369
Ketahanan	2370
Keamanan infrastruktur	2371
Menggunakan titik akhir VPC	2372
AWS PrivateLink untuk DynamoDB	2381
Jenis titik akhir VPC Amazon	2382
Pertimbangan saat menggunakan AWS PrivateLink untuk Amazon DynamoDB	2383
Membuat titik akhir Amazon VPC	2384
Mengakses titik akhir antarmuka Amazon DynamoDB	2384
Mengakses tabel DynamoDB dan mengontrol operasi API dari titik akhir antarmuka DynamoDB	2384
Memperbarui konfigurasi DNS on-premise	2387
Membuat kebijakan titik akhir Amazon VPC	2389
Konfigurasi dan analisis kelemahan	2390
Praktik terbaik keamanan	2390
Praktik terbaik keamanan preventif	2390
Praktik terbaik keamanan detective	2393
Pemantauan	2397
Pencatatan dan pemantauan di DynamoDB	2397
Alat-alat pemantauan	2398
Pemantauan CloudWatch dengan Amazon	2400
Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail	2432
Pencatatan log dan pemantauan di DAX	2455
Wawasan Kontributor	2456
Cara kerjanya	2456
Memulai	2463
Menggunakan IAM	2469

Menggunakan Pemberitahuan Notifikasi Pengguna AWS Pengguna dengan Amazon DynamoDB	2474
Praktik terbaik	2476
Desain NoSQL	2476
NoSQL vs. RDBMS	2477
Dua konsep utama	2477
Pendekatan umum	2478
NoSQL Workbench	2479
Perlindungan penghapusan	2480
DynamoDB Well-Architected Lens	2480
Optimasi biaya	2480
Melakukan tinjauan Amazon DynamoDB Well-Architected Lens	2532
Pilar Amazon DynamoDB Well-Architected Lens	2532
Desain kunci partisi	2535
Kapasitas lonjakan	2535
Kapasitas adaptif	2536
Mendistribusikan beban kerja	2537
Pembagian tulis	2539
Mengunggah data secara efisien	2540
Desain kunci urutan	2542
Kontrol versi	2543
Indeks sekunder	2544
Pedoman umum	2544
Indeks jarang	2547
Agregasi	2549
Muatan berlebih GSI	2551
Pembagian GSI	2552
Membuat replika	2553
Item besar	2555
Kompresi	2555
Partisi vertikal	2555
Menggunakan Amazon S3	2556
Data deret waktu	2557
Pola desain untuk data deret waktu	2557
Contoh tabel deret waktu	2557
any-to-many Hubungan M	2558

Daftar kedekatan	2559
Grafik terwujud	2560
DynamoDB–RDBMS hibrida	2565
Tidak bermigrasi	2565
Penerapan sistem hibrida	2566
Pemodelan relasional	2567
Model basis data relasional tradisional	2567
Cara DynamoDB menghilangkan kebutuhan akan operasi JOIN	2569
Bagaimana transaksi DynamoDB menghilangkan overhead ke proses tulis	2570
Langkah pertama	2571
Contoh	2573
Mengkueri dan memindai	2577
Performa pemindaian	2577
Menghindari lonjakan	2578
Pemindaian paralel	2581
Desain tabel	2582
Desain tabel global	2582
Desain tabel global	2583
Fakta kunci	2583
Kasus penggunaan	2585
Mode tulis	2586
Meminta perutean	2594
Mengevakuasi Wilayah	2603
Kapasitas throughput untuk tabel global	2606
Daftar periksa dan FAQ untuk tabel global	2608
Bidang kontrol	2615
Laporan Penagihan dan Penggunaan	2616
Kapasitas Throughput	2619
Pengaliran	2623
Penyimpanan	2624
Pencadangan dan Pemulihan	2625
Transfer Data	2629
CloudWatch	2629
DAX	2631
Menggunakan DynamoDB dengan layanan AWS lain	2633
Mengintegrasikan dengan Amazon Cognito	2633

Mengintegrasikan dengan Amazon Redshift	2635
Mengintegrasikan dengan Amazon EMR	2637
Gambaran Umum	2637
Tutorial: Bekerja dengan Amazon DynamoDB dan Apache Hive	2638
Membuat tabel eksternal di Hive	2647
Mengolah Pernyataan HiveQL	2651
Melakukan kueri data dalam DynamoDB	2652
Menyalin data ke dan dari Amazon DynamoDB	2654
Penyempurnaan kinerja	2668
Integrasi dengan S3	2674
Mengimpor dari Amazon S3	2675
Ekspor ke Amazon S3	2696
Integrasi dengan Amazon Service OpenSearch	2720
Cara kerjanya	2721
Membuat integrasi	2722
Langkah selanjutnya	2722
Menangani perubahan yang menyebabkan gangguan	2722
Praktik terbaik integrasi	2726
Membuat snapshot	2726
Tangkapan data perubahan	2727
Integrasi nol-ETL dengan Layanan OpenSearch	2727
Kuota dan batas	2731
Throughput dan mode kapasitas baca/tulis	2731
Ukuran unit kapasitas (untuk tabel yang disediakan)	2732
Ukuran unit permintaan (untuk tabel sesuai permintaan)	2732
Kuota default throughput	2732
Meningkatkan atau mengurangi throughput (untuk tabel yang disediakan)	2734
Kapasitas Terpesan	2735
Kuota impor	2736
Wawasan Kontributor	2736
Tabel	2736
Ukuran tabel	2736
Jumlah maksimum tabel per akun per wilayah	2736
Tabel global	2737
Indeks sekunder	2738
Indeks sekunder per tabel	2738

Atribut Indeks Sekunder yang diproyeksikan per tabel	2738
Tombol partisi dan kunci urutan	2738
Panjang kunci partisi	2738
Nilai kunci partisi	2739
Panjang kunci urutan	2739
Nilai kunci urutan	2739
Peraturan penamaan	2739
Nama tabel dan nama Indeks Sekunder	2739
Nama atribut	2740
Tipe Data	2740
String	2740
Angka	2740
Biner	2741
Item	2741
Ukuran item	2741
Ukuran item untuk tabel dengan Indeks Sekunder Lokal	2741
Atribut	2741
Pasangan atribut nama-nilai per item	2741
Jumlah nilai dalam daftar, peta, atau set	2741
Nilai atribut	2742
Kedalaman atribut bertingkat	2742
Parameter ekspresi	2742
Panjang	2742
Operator dan operan	2742
Kata yang dicadangkan	2742
Transaksi DynamoDB	2743
DynamoDB Streams	2743
Pembaca serpihan secara bersamaan di DynamoDB Streams	2743
Kapasitas tulis maksimum untuk tabel dengan DynamoDB Streams diaktifkan	2743
DynamoDB Accelerator (DAX)	2744
AWS ketersediaan wilayah	2744
Simpul	2744
Grup parameter	2745
Grup subnet	2745
Batasan khusus API	2745
Enkripsi DynamoDB saat istirahat	2748

Ekspor tabel ke Amazon S3	2748
Pencadangan dan pemulihan	2748
Referensi API	2749
Pemecahan Masalah	2750
Latensi	2750
Throttling	2752
Pastikan tabel mode yang disediakan memiliki kapasitas yang memadai	2753
Pertimbangkan untuk beralih ke mode on-demand	2753
Mendistribusikan operasi baca dan tulis secara merata di antara kunci partisi	2754
Tingkatkan kuota throughput baca atau tulis tingkat tabel	2754
Lampiran	2756
Memecahkan masalah pembentukan koneksi SSL/TLS	2756
Menguji aplikasi atau layanan Anda	2756
Menguji browser klien Anda	2757
Memperbarui klien aplikasi perangkat lunak Anda	2757
Memperbarui browser klien Anda	2758
Memperbarui paketan sertifikat Anda secara manual	2758
Contoh tabel dan data	2759
File File File File File File File File File	2760
Membuat contoh tabel dan mengunggah data	2773
Membuat contoh tabel dan mengunggah data - Java	2773
Membuat contoh tabel dan mengunggah data - .NET	2783
Contoh aplikasi menggunakan AWS SDK for Python (Boto3)	2795
Langkah 1: men-deploy dan menguji secara lokal	2797
Langkah 2: Periksa model data dan detail implementasi	2802
Langkah 3: Menerapkan dalam Produksi	2812
Langkah 4: Bersihkan Sumber Daya	2822
Integrasi dengan AWS Data Pipeline	2822
Prasyarat untuk ekspor dan impor data	2825
Mengekspor data dari DynamoDB ke Amazon S3	2834
Mengimpor data dari Amazon S3 ke DynamoDB	2835
Memecahkan masalah	2837
Template yang telah ditetapkan untuk AWS Data Pipeline dan DynamoDB	2838
Backend Penyimpanan Amazon DynamoDB untuk Titan	2839
Disimpan kata-kata di DynamoDB	2839
Parameter bersyarat lama	2852

AttributesToGet	2854
AttributeUpdates	2855
ConditionalOperator	2858
Diharapkan	2859
KeyConditions	2864
QueryFilter	2868
ScanFilter	2870
Kondisi penulisan dengan parameter lama	2872
Versi API tingkat rendah sebelumnya (2011-12-05)	2880
BatchGetItem	2881
BatchWriteItem	2889
CreateTable	2896
DeleteItem	2904
DeleteTable	2911
DescribeTables	2915
GetItem	2919
ListTables	2923
PutItem	2926
Kueri	2934
Pemindaian	2952
UpdateItem	2973
UpdateTable	2983
AWS Contoh SDK for Java 1.x	2988
DAX dan Java SDK v1	2989
Memodifikasi SDK for Java 1.x yang ada agar menggunakan DAX	3001
Menanyakan indeks sekunder global dengan SDK for Java 1.x	3006
Riwayat dokumen	3010
Pembaruan sebelumnya	3033
Fitur warisan	3069
Versi tabel global 2017.11.29 (Legacy)	3069
Cara kerjanya	3069
Praktik Terbaik dan Persyaratan	3075
Membuat tabel global	3079
Memantau tabel global	3083
Menggunakan IAM dengan tabel global	3084
.....	mmmlxxxviii

Apa itu Amazon DynamoDB?

Amazon DynamoDB adalah layanan database NoSQL terkelola sepenuhnya yang menyediakan kinerja cepat dan dapat diprediksi dengan skalabilitas tanpa hambatan. DynamoDB memungkinkan Anda meringankan beban administratif pengoperasian dan penskalaan database terdistribusi sehingga Anda tidak perlu khawatir tentang penyediaan perangkat keras, penyiapan dan konfigurasi, replikasi, patching perangkat lunak, atau penskalaan klaster. DynamoDB juga menawarkan enkripsi saat istirahat, yang menghilangkan beban operasional dan kompleksitas yang terkait dengan perlindungan data sensitif. Untuk informasi selengkapnya, lihat [Enkripsi DynamoDB saat diam](#).

Dengan DynamoDB, Anda dapat membuat tabel database yang dapat menyimpan dan mengambil data dalam jumlah berapa pun dan melayani tingkat lalu lintas permintaan apa pun. Anda dapat meningkatkan atau menurunkan kapasitas throughput tabel Anda tanpa downtime atau penurunan performa. Anda dapat menggunakan AWS Management Console untuk memantau pemanfaatan sumber daya dan metrik performa.

DynamoDB menyediakan kemampuan pencadangan sesuai permintaan. Hal ini memungkinkan Anda untuk membuat pencadangan penuh dari tabel Anda untuk retensi dan pengarsipan jangka panjang untuk kebutuhan kepatuhan terhadap peraturan. Untuk informasi selengkapnya, lihat [Menggunakan cadangan Sesuai Permintaan dan DynamoDB Permintaan](#).

Anda dapat membuat cadangan sesuai permintaan dan mengaktifkan point-in-time pemulihan untuk tabel Amazon DynamoDB Anda. Point-in-time Pemulihan P membantu melindungi tabel Anda dari operasi menulis atau menghapus yang tidak disengaja. Dengan point-in-time pemulihan, Anda dapat mengembalikan tabel ke titik waktu mana pun selama 35 hari terakhir. Untuk informasi selengkapnya, lihat [Point-in-time recovery: Cara kerjanya](#).

DynamoDB memungkinkan Anda untuk menghapus item kedaluwarsa dari tabel secara otomatis untuk membantu Anda mengurangi penggunaan penyimpanan dan biaya menyimpan data yang tidak lagi relevan. Untuk informasi selengkapnya, lihat [Waktu untuk Hidup \(TTL\)](#).

Ketersediaan tinggi dan daya tahan

DynamoDB secara otomatis menyebar data dan lalu lintas untuk tabel Anda atas jumlah server yang cukup untuk menangani throughput dan persyaratan penyimpanan Anda, sambil mempertahankan performa yang konsisten dan cepat. Semua data Anda disimpan di solid-state disk (SSD) dan secara otomatis direplikasi di beberapa Availability Zone di suatu AWS Wilayah, menyediakan ketersediaan

tinggi bawaan dan daya tahan data. Anda dapat menggunakan tabel global untuk menjaga tabel DynamoDB tetap sinkron di seluruh Wilayah. AWS Untuk informasi selengkapnya, lihat [Tabel global - Replikasi multi-Wilayah untuk DynamoDB](#).

Mulai menggunakan DynamoDB

Kami menyarankan Anda untuk memulai dengan membaca bagian berikut:

- [Amazon DynamoDB: Cara kerjanya](#)—Untuk mempelajari konsep penting DynamoDB.
- [Menyiapkan DynamoDB](#)—Untuk mempelajari cara mengatur DynamoDB (versi dapat diunduh atau layanan web).
- [Mengakses DynamoDB](#)— Untuk mempelajari cara mengakses DynamoDB menggunakan konsol AWS CLI,, atau API.

Dari sana, Anda memiliki dua opsi untuk memulai dengan DynamoDB dengan cepat:

- [Mulai menggunakan DynamoDB](#)
- [Memulai dengan DynamoDB dan SDK AWS](#)

Untuk mempelajari lebih lanjut tentang pengembangan aplikasi, lihat hal berikut:

- [Pemrograman dengan DynamoDB dan SDK AWS](#)
- [Bekerja dengan tabel, item, kueri, pindaian, dan indeks](#)

Untuk menemukan rekomendasi dengan cepat untuk memaksimalkan performa dan meminimalkan biaya throughput, lihat yang berikut ini: [Praktik terbaik untuk mendesain dan merancang dengan DynamoDB](#). Untuk mempelajari cara menandai sumber daya DynamoDB, lihat [Menambahkan tag dan label ke sumber daya](#).

Untuk praktik terbaik, panduan petunjuk, dan alat, lihat [Sumber daya Amazon DynamoDB](#).

Anda dapat menggunakan AWS Database Migration Service (AWS DMS) untuk memigrasikan data dari database relasional atau MongoDB ke tabel DynamoDB. Untuk informasi selengkapnya, lihat [AWS Database Migration Service Panduan Pengguna](#).

Untuk mempelajari cara menggunakan MongoDB sebagai sumber migrasi, lihat [Menggunakan MongoDB sebagai Sumber untuk AWS Database Migration Service](#). Untuk mempelajari cara

menggunakan DynamoDB sebagai target migrasi, [lihat Menggunakan database Amazon DynamoDB sebagai target](#). AWS Database Migration Service

DynamoDB tutorial

Tutorial berikut menyajikan end-to-end prosedur lengkap untuk membiasakan diri dengan DynamoDB. Tutorial ini dapat dilengkapi dengan tingkat gratis AWS dan akan memberi Anda pengalaman praktis menggunakan DynamoDB.

- [Membangun Aplikasi Menggunakan NoSQL Key-Value Data Store](#)
- [Buat dan Kueri Tabel NoSQL dengan Amazon DynamoDB](#)

Amazon DynamoDB: Cara kerjanya

Bagian berikut memberikan gambaran umum tentang komponen layanan Amazon DynamoDB dan bagaimana mereka berinteraksi.

Setelah Anda membaca pendahuluan ini, cobalah bekerja melalui bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#), yang memandu Anda melalui proses pembuatan tabel contoh, mengunggah data, dan melakukan beberapa operasi database dasar.

Untuk tutorial bahasa khusus dengan kode sampel, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Topik

- [Kisi-kisi untuk DynamoDB](#)
- [Komponen inti dari Amazon DynamoDB](#)
- [API DynamoDB](#)
- [Jenis data dan aturan penamaan yang didukung di Amazon DynamoDB](#)
- [Konsistensi baca](#)
- [Mode kapasitas baca/tulis](#)
- [Kelas tabel](#)
- [Partisi dan distribusi data](#)

Kisi-kisi untuk DynamoDB

Lembar contekan ini memberikan referensi cepat untuk bekerja dengan Amazon DynamoDB dan berbagai SDK-nya. AWS

Pengaturan awal

1. [Mendaftar untuk AWS](#).
2. [Dapatkan kunci akses AWS](#) untuk mengakses DynamoDB secara terprogram.
3. [Konfigurasi kredensial DynamoDB](#) Anda.

Lihat juga:

- [Menyiapkan DynamoDB \(layanan web\)](#)
- [Mulai menggunakan DynamoDB](#)
- [Ikhtisar dasar komponen inti](#)

SDK atau CLI

Pilih [SDK](#) pilihan Anda, atau atur [AWS CLI](#).

Note

Saat Anda menggunakan AWS CLI on Windows, garis miring terbalik (\) yang tidak ada di dalam kutipan diperlakukan sebagai carriage return. Selain itu, Anda harus menghilangkan tanda kutip dan kurung kurawal apa pun di dalam tanda kutip lainnya. Sebagai contoh, lihat tab Windows di “Buat tabel” di bagian berikutnya.

Lihat juga:

- [AWS CLI dengan DynamoDB](#)
- [Mulai menggunakan DynamoDB - langkah 2](#)

Tindakan dasar

Bagian ini menyediakan kode untuk tugas DynamoDB dasar. Untuk informasi selengkapnya tentang tugas ini, lihat [Memulai DynamoDB dan SDK](#). AWS

Buat tabel

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 \  
  --provisioned-throughput \  
    ReadCapacityUnits=10,WriteCapacityUnits=5
```

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 ^  
  --provisioned-throughput ^  
    ReadCapacityUnits=10,WriteCapacityUnits=5
```

Tulis item ke tabel

```
aws dynamodb put-item \  
  --table-name Music \  
  --item file:///item.json
```

Baca item dari tabel

```
aws dynamodb get-item \  
  --table-name Music \  
  --item file:///item.json
```


Hapus item dari tabel

```
aws dynamodb delete-item --table-name Music --key file://key.json
```

Mengkueri Tabel

```
aws dynamodb query --table-name Music  
--key-condition-expression "ArtistName=:Artist and SongName=:Songtitle"
```

Hapus tabel

```
aws dynamodb delete-table --table-name Music
```

Daftar nama tabel

```
aws dynamodb list-tables
```

Peraturan penamaan

- Semua nama harus dikodekan menggunakan UTF-8 dan peka huruf besar-kecil.
- Nama tabel dan nama indeks panjangnya harus antara 3 hingga 255 karakter, dan hanya dapat berisi karakter berikut:
 - a-z
 - A-Z
 - 0-9
 - _(garis bawah)
 - -(dasbor)
 - .(titik)
- Nama atribut harus memiliki panjang minimal satu karakter dan berukuran kurang dari 64 KB.

Untuk informasi selengkapnya, lihat [Aturan penamaan](#).

Dasar-dasar kuota layanan

Membaca dan menulis unit

- Unit kapasitas baca (RCU) – Satu pembacaan sangat konsisten per detik, atau dua pembacaan akhirnya konsisten per detik, untuk item berukuran hingga 4 KB.
- Unit kapasitas tulis (WCU) – Satu tulis per detik, untuk item berukuran hingga 1 KB.

Batasan tabel

- Ukuran tabel — Tidak ada batas praktis pada ukuran tabel. Tabel tidak dibatasi dalam jumlah item atau jumlah byte.
- Jumlah tabel — Untuk AWS akun apa pun, ada kuota awal 2.500 tabel per AWS Wilayah.
- Batas ukuran halaman untuk kueri dan pemindaian – Ada batas 1 MB per halaman, per kueri atau pemindaian. Jika parameter kueri atau operasi pemindaian Anda pada tabel menghasilkan lebih dari 1 MB data, DynamoDB mengembalikan item awal yang cocok. Ini juga mengembalikan properti `LastEvaluatedKey` yang dapat Anda gunakan dalam permintaan baru untuk membaca halaman berikutnya.

Indeks

- Indeks sekunder lokal (LSI) – Anda dapat menentukan maksimum lima indeks sekunder lokal. LSI terutama berguna ketika indeks harus memiliki konsistensi yang kuat dengan tabel dasar.
- Indeks sekunder global (GSI) – Ada kuota default 20 indeks sekunder global per tabel.
- Atribut indeks sekunder yang diproyeksikan per tabel – Anda dapat memproyeksikan total hingga 100 atribut ke semua indeks lokal tabel dan sekunder global. Ini hanya berlaku untuk atribut proyeksi yang ditentukan pengguna.

Kunci partisi

- Panjang minimum nilai kunci partisi adalah 1 byte. Panjang maksimum adalah 2048 byte.
- Tidak ada batasan praktis untuk jumlah nilai kunci partisi yang berbeda, untuk tabel atau indeks sekunder.
- Panjang minimum nilai kunci urutan adalah 1 byte. Panjang maksimum adalah 1024 byte.
- Secara umum, tidak ada batas praktis pada jumlah nilai kunci urutan berbeda per nilai kunci partisi. Pengecualian ini adalah untuk tabel dengan indeks sekunder.

Untuk informasi selengkapnya tentang indeks sekunder, desain kunci partisi, dan desain kunci urutan, lihat [Praktik terbaik](#).

Batas untuk tipe data yang umum digunakan

- String – Panjang string dibatasi oleh ukuran item maksimum 400 KB. String adalah Unicode dengan pengodean biner UTF-8.
- Angka – Angka dapat memiliki hingga 38 digit presisi, dan dapat positif, negatif, atau nol.
- Binary – Panjang biner dibatasi oleh ukuran item maksimum 400 KB. Aplikasi yang bekerja dengan atribut biner harus mengkodekan data dalam pengkodean base64 sebelum mengirimnya ke DynamoDB.

Untuk daftar tipe daya yang didukung, lihat [Jenis data](#). Untuk informasi selengkapnya, lihat [Kuota layanan](#).

Item, atribut, dan parameter ekspresi

Ukuran item maksimum di DynamoDB adalah 400 KB, yang mencakup panjang biner nama atribut (UTF-8 panjang) dan panjang biner nilai atribut (UTF-8 panjang). Nama atribut dihitung terhadap batas ukuran.

Tidak ada batas pada jumlah nilai dalam daftar, peta, atau set, selama item yang berisi nilai-nilai cocok dalam batas ukuran item 400-KB.

Untuk parameter ekspresi, panjang maksimum string ekspresi adalah 4 KB.

Untuk informasi selengkapnya tentang ukuran item, atribut, dan parameter ekspresi, lihat [Kuota layanan](#).

Informasi lain

- [Keamanan](#)
- [Pencatatan dan pemantauan](#)
- [Bekerja dengan stream](#)
- [Cadangan dan pemulihan P oint-in-time](#)
- [Integrasi dengan layanan lain AWS](#)
- [Referensi API](#)
- [Pusat Arsitektur: Praktik Terbaik Database](#)
- [Video tutorial](#)
- [Forum DynamoDB](#)

Komponen inti dari Amazon DynamoDB

Dalam DynamoDB, tabel, item, dan atribut adalah komponen inti yang Anda kerjakan. Sebuah tabel adalah koleksi item, dan setiap item adalah koleksi atribut. DynamoDB menggunakan kunci primer untuk secara unik mengidentifikasi setiap item dalam tabel dan indeks sekunder untuk memberikan kueri yang lebih fleksibel. Anda dapat menggunakan DynamoDB Streams untuk menangkap peristiwa modifikasi data dalam tabel DynamoDB.

Ada batas dalam DynamoDB. Untuk informasi selengkapnya, lihat [Layanan, akun, dan tabel kuota di Amazon DynamoDB](#).

Video berikut ini akan memberi Anda tampilan pengantar tabel, item, dan atribut.

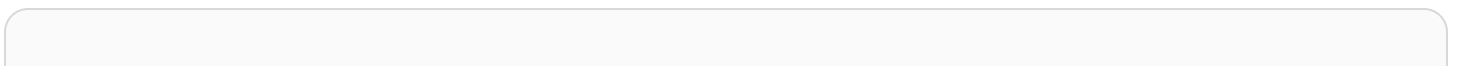
[Tabel, item, dan atribut](#)

Tabel, item, dan atribut

Berikut ini adalah komponen DynamoDB dasar:

- **Tabel** – Mirip dengan sistem database lainnya, DynamoDB menyimpan data dalam tabel. Sebuah tabel adalah koleksi data. Misalnya, lihat tabel contoh yang disebut Orang yang dapat Anda gunakan untuk menyimpan informasi kontak pribadi tentang teman, keluarga, atau orang lain yang menarik. Anda juga dapat memiliki tabel Mobil untuk menyimpan informasi tentang kendaraan yang dikendarai orang.
- **Item** — Setiap tabel berisi nol item atau lebih. Sebuah item adalah sekelompok atribut yang dapat diidentifikasi secara unik di antara semua item lainnya. Dalam tabel Orang, setiap item mewakili satu orang. Untuk tabel Mobil, setiap item mewakili satu kendaraan. Item dalam DynamoDB serupa dalam banyak hal dengan baris, catatan, atau tupel dalam sistem database lain. Dalam DynamoDB, tidak ada batasan jumlah item yang dapat Anda simpan dalam tabel.
- **Atribut** — Setiap item terdiri dari satu atau beberapa atribut. Sebuah atribut adalah elemen data fundamental, sesuatu yang tidak perlu dipecah lebih jauh. Misalnya, item dalam tabel People berisi atribut yang disebut `personId`, `lastName`, `firstName`, dan sebagainya. Untuk tabel Departemen, item mungkin memiliki atribut seperti `DepartmentID`, `Name`, `Manager`, dan sebagainya. Atribut di DynamoDB serupa dalam banyak hal dengan bidang atau kolom dalam sistem database lain.

Diagram berikut menunjukkan tabel bernama Orang dengan beberapa item dan atribut contoh.



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"
}
```

Perhatikan hal berikut tentang tabel Orang:

- Setiap item dalam tabel memiliki pengidentifikasi unik, atau kunci primer, yang membedakan item dari semua hal lain dalam tabel. Dalam tabel Orang, kunci primer terdiri dari satu atribut (PersonID).
- Selain kunci primer, tabel Orang tidak berskema, yang berarti bahwa baik atribut maupun tipe datanya perlu didefinisikan terlebih dahulu. Setiap item dapat memiliki atribut tersendiri.
- Sebagian besar atribut skalar, yang berarti bahwa mereka hanya dapat memiliki satu nilai. String dan angka adalah contoh umum dari skalar.

- Beberapa item memiliki atribut bersarang (Alamat). DynamoDB mendukung atribut bersarang hingga sedalam 32 tingkat.

Berikut ini adalah tabel contoh lain bernama Musik yang dapat Anda gunakan untuk melacak koleksi musik Anda.

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"
}
```

Perhatikan hal berikut tentang tabel Musik:

- Kunci utama untuk Musik terdiri dari dua atribut (Artis dan SongTitle). Setiap item dalam tabel harus memiliki dua atribut ini. Kombinasi Artis dan SongTitle membedakan setiap item dalam tabel dari yang lainnya.
- Selain kunci primer, tabel Musik tidak berskema, yang berarti bahwa baik atribut maupun tipe datanya perlu didefinisikan terlebih dahulu. Setiap item dapat memiliki atribut tersendiri.
- Salah satu item memiliki atribut bersarang (PromotionInfo), yang berisi atribut bersarang lainnya. DynamoDB mendukung atribut bersarang hingga sedalam 32 tingkat.

Untuk informasi selengkapnya, lihat [Bekerja dengan tabel dan data di DynamoDB](#).

Kunci primer

Ketika Anda membuat tabel, selain nama tabel, Anda harus menentukan kunci primer dari tabel. Kunci utama secara unik mengidentifikasi setiap item dalam tabel, sehingga tidak ada dua item dapat memiliki kunci yang sama.

DynamoDB mendukung dua jenis kunci primer:

- Kunci partisi — Sebuah kunci primer sederhana, terdiri dari satu atribut yang dikenal sebagai kunci partisi.

DynamoDB menggunakan nilai kunci partisi sebagai masukan untuk fungsi hash internal. Output dari fungsi hash menentukan partisi (penyimpanan fisik internal pada DynamoDB) di mana item akan disimpan.

Dalam tabel yang hanya memiliki kunci partisi, tidak ada dua item dapat memiliki nilai kunci partisi yang sama.

Tabel Orang yang dijelaskan dalam [Tabel, item, dan atribut](#) adalah contoh dari sebuah tabel dengan kunci primer sederhana (PersonID). Anda dapat mengakses item apa pun di tabel Orang secara langsung dengan memberikan PersonID nilai untuk item tersebut.

- Kunci partisi dan kunci urutan – Disebut sebagai kunci primer komposit, jenis kunci ini terdiri dari dua atribut. Atribut pertama adalah kunci partisi, dan atribut kedua adalah kunci urutan.

DynamoDB menggunakan nilai kunci partisi sebagai input untuk fungsi hash internal. Output dari fungsi hash menentukan partisi (penyimpanan fisik internal pada DynamoDB) di mana item akan disimpan. Semua item dengan nilai kunci partisi yang sama disimpan bersama-sama, dalam urutan yang diurutkan berdasarkan nilai kunci urutan.

Dalam tabel yang memiliki kunci partisi dan kunci urutan, mungkin bagi beberapa item memiliki nilai kunci partisi yang sama. Namun, item tersebut harus memiliki nilai kunci urutan yang berbeda.

Tabel Musik yang dijelaskan dalam [Tabel, item, dan atribut](#) adalah contoh tabel dengan kunci primer komposit (Artis dan SongTitle). Anda dapat mengakses item apa pun di tabel Musik secara langsung, jika Anda memberikan Artis dan SongTitle nilai untuk item tersebut.

Kunci primer komposit memberi Anda fleksibilitas tambahan saat mengkueri data. Misalnya, jika Anda hanya memberikan nilai untuk Artist, DynamoDB mengambil semua lagu oleh penyanyi tersebut. Untuk mengambil hanya subset lagu oleh artis tertentu, Anda dapat memberikan nilai untuk Artis bersama dengan berbagai nilai untuk SongTitle

Note

Kunci partisi dari item juga dikenal sebagai atribut hash. Istilah atribut hash berasal dari penggunaan fungsi hash internal di DynamoDB yang mendistribusikan item data secara merata di seluruh partisi, berdasarkan nilai kunci partisi mereka.

Kunci urutan item juga dikenal sebagai atribut rentang. Istilah atribut rentang berasal dari cara DynamoDB menyimpan item dengan kunci partisi yang sama yang secara fisik berdekatan, dalam urutan berdasarkan nilai kunci urutan.

Setiap atribut kunci primer harus skalar (yang berarti bahwa atribut itu hanya dapat menyimpan nilai tunggal). Satu-satunya tipe data yang diperbolehkan untuk atribut kunci primer adalah string, nomor, atau biner. Tidak ada batasan seperti itu untuk atribut non-kunci lainnya.

Indeks sekunder

Anda dapat membuat satu atau beberapa indeks sekunder pada tabel. Indeks sekunder memungkinkan Anda mengkueri data dalam tabel menggunakan kunci alternatif, selain kueri terhadap kunci primer. DynamoDB tidak mengharuskan Anda menggunakan indeks, tetapi mereka memberikan aplikasi Anda fleksibilitas lebih ketika mengkueri pada data Anda. Setelah Anda membuat indeks sekunder pada tabel, Anda bisa membaca data dari indeks dengan cara yang sama seperti yang Anda lakukan dari tabel.

DynamoDB mendukung dua jenis indeks:

- Indeks sekunder global – Indeks dengan kunci partisi dan kunci urutan yang mungkin berbeda dari yang ada di tabel.
- Indeks sekunder lokal – Indeks yang memiliki kunci partisi yang sama dengan tabel, namun kunci urutan berbeda.

Di DynamoDB, indeks sekunder global (GSI) adalah indeks yang menjangkau seluruh tabel, memungkinkan Anda untuk melakukan kueri di semua kunci partisi. Indeks sekunder lokal (LSI) adalah indeks yang memiliki kunci partisi yang sama dengan tabel dasar tetapi kunci pengurutan yang berbeda.

Setiap tabel di DynamoDB memiliki kuota 20 indeks sekunder global (kuota default) dan 5 indeks sekunder lokal.

Dalam contoh tabel Musik yang ditampilkan sebelumnya, Anda dapat meminta item data berdasarkan Artis (kunci partisi) atau oleh Artis dan SongTitle(kunci partisi dan kunci sortir). Bagaimana jika Anda juga ingin menanyakan data berdasarkan Genre dan AlbumTitle? Untuk melakukan ini, Anda dapat membuat indeks pada Genre dan AlbumTitle, lalu menanyakan indeks dengan cara yang sama seperti Anda menanyakan tabel Musik.

Diagram berikut menunjukkan contoh tabel Musik, dengan indeks baru yang disebut GenreAlbumTitle. Dalam indeks, Genre adalah kunci partisi dan AlbumTitle merupakan kunci sortir.

Tabel Musik	GenreAlbumTitle
<pre>{ "Artist": "No One You Know", "SongTitle": "My Dog Spot", "AlbumTitle": "Hey Now", "Price": 1.98, "Genre": "Country", "CriticRating": 8.4 }</pre>	<pre>{ "Genre": "Country", "AlbumTitle": "Hey Now", "Artist": "No One You Know", "SongTitle": "My Dog Spot" }</pre>
<pre>{ "Artist": "No One You Know", "SongTitle": "Somewhere Down The Road", "AlbumTitle": "Somewhat Famous", "Genre": "Country", "CriticRating": 8.4, "Year": 1984 }</pre>	<pre>{ "Genre": "Country", "AlbumTitle": "Somewhat Famous", "Artist": "No One You Know", "SongTitle": "Somewhere Down The Road" }</pre>
<pre>{ "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": {</pre>	<pre>{ "Genre": "Rock", "AlbumTitle": "The Buck Starts Here", "Artist": "The Acme Band", "SongTitle": "Still In Love" }</pre>

Tabel Musik	GenreAlbumTitle
<pre> "Seattle": "20150622", "Cleveland": "20150630" }, "Rotation": "Heavy" } } </pre>	
<pre> { "Artist": "The Acme Band", "SongTitle": "Look Out, World", "AlbumTitle": "The Buck Starts Here", "Price": 0.99, "Genre": "Rock" } </pre>	<pre> { "Genre": "Rock", "AlbumTitle": "The Buck Starts Here", "Artist": "The Acme Band", "SongTitle": "Look Out, World" } </pre>

Perhatikan hal berikut tentang GenreAlbumTitle indeks:

- Setiap indeks adalah milik sebuah tabel, yang disebut tabel dasar untuk indeks. Dalam contoh sebelumnya, Musik adalah tabel dasar untuk indeks. GenreAlbumTitle
- DynamoDB mengelola indeks secara otomatis. Saat Anda menambahkan, memperbarui, atau menghapus item di tabel dasar, DynamoDB menambahkan, memperbarui, atau menghapus item terkait di indeks mana pun yang termasuk dalam tabel tersebut.
- Saat Anda membuat indeks, Anda menentukan atribut mana yang akan disalin, atau diproyeksikan, dari tabel dasar ke indeks. Minimal, DynamoDB memproyeksikan atribut kunci dari tabel dasar ke indeks. Hal ini terjadi pada GenreAlbumTitle, dimana hanya atribut kunci dari tabel yang Music diproyeksikan ke dalam indeks.

Anda dapat menanyakan GenreAlbumTitle indeks untuk menemukan semua album dari genre tertentu (misalnya, semua album Rock). Anda juga dapat menanyakan indeks untuk menemukan semua album dalam genre tertentu yang memiliki judul album tertentu (misalnya, semua album Country dengan judul yang dimulai dengan huruf H).

Untuk informasi selengkapnya, lihat [Meningkatkan akses data dengan indeks sekunder](#).

DynamoDB Streams

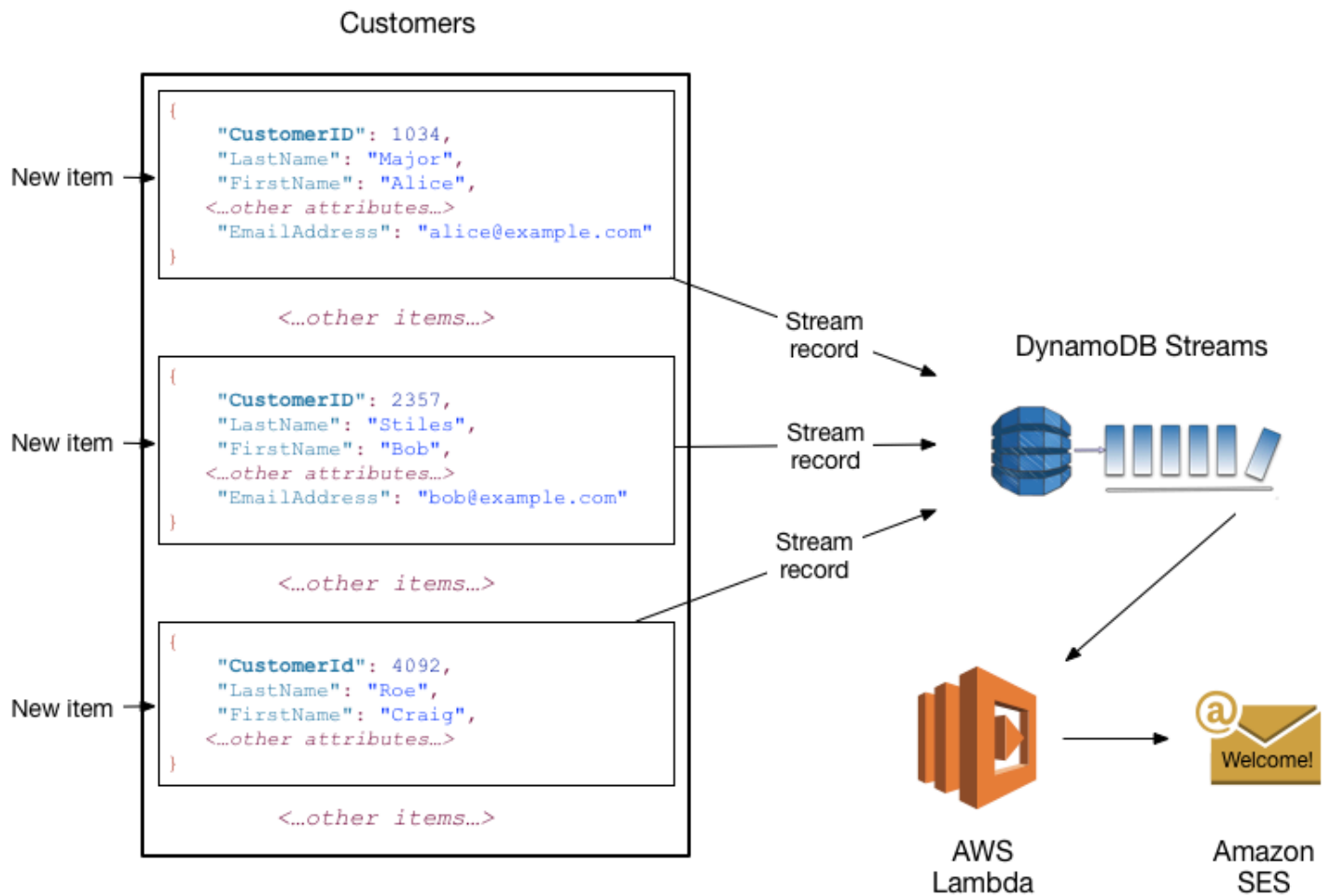
DynamoDB Streams adalah fitur opsional yang menangkap peristiwa modifikasi data dalam tabel DynamoDB. Data tentang peristiwa ini muncul di aliran hampir dalam waktu nyata, dan sesuai urutan terjadinya peristiwa tersebut.

Setiap acara diwakili oleh catatan aliran. Jika Anda mengaktifkan aliran pada tabel, DynamoDB Streams menulis catatan aliran setiap kali salah satu peristiwa berikut terjadi:

- Item baru ditambahkan ke tabel: Aliran mengambil gambar keseluruhan item, termasuk semua atributnya.
- Item diperbarui: Aliran menangkap gambar "sebelum" dan "sesudah" dari setiap atribut yang diubah dalam item.
- Item dihapus dari tabel: Aliran mengambil gambar keseluruhan item sebelum dihapus.

Setiap rekaman aliran juga berisi nama tabel, stempel waktu peristiwa, dan metadata lainnya. Rekaman streaming memiliki masa pakai 24 jam; setelah itu, mereka secara otomatis dihapus dari aliran.

Anda dapat menggunakan DynamoDB Streams AWS Lambda bersama-sama untuk membuat pemicu —kode yang berjalan secara otomatis setiap kali peristiwa yang diinginkan muncul dalam aliran. Misalnya, pertimbangkan tabel Pelanggan yang berisi informasi pelanggan untuk sebuah perusahaan. Misalkan Anda ingin mengirim email "selamat datang" ke setiap pelanggan baru. Anda dapat mengaktifkan aliran pada tabel itu, dan kemudian mengaitkan aliran tersebut dengan fungsi Lambda. Fungsi Lambda akan berjalan setiap kali rekaman aliran baru muncul, namun hanya memproses item baru yang ditambahkan ke tabel Pelanggan. Untuk item apa pun yang memiliki atribut `EmailAddress`, fungsi Lambda akan memanggil Amazon Simple Email Service (Amazon SES) untuk mengirim email ke alamat tersebut.



Note

Dalam contoh ini, pelanggan terakhir, Craig Roe, tidak akan menerima email karena dia tidak memiliki EmailAddress.

Selain pemicu, DynamoDB Streams memungkinkan solusi canggih seperti replikasi data di dalam dan AWS di seluruh Wilayah, tampilan data yang terwujud dalam tabel DynamoDB, analisis data menggunakan tampilan terwujud Kinesis, dan banyak lagi.

Untuk informasi selengkapnya, lihat [Ubah tangkapan data DynamoDB Streams](#).

API DynamoDB

Untuk bekerja dengan Amazon DynamoDB, aplikasi Anda harus menggunakan beberapa operasi API sederhana. Berikut ini adalah ringkasan operasi-operasi ini, yang disusun berdasarkan kategori.

 Note

Untuk daftar lengkap operasi API, lihat [Referensi API Amazon DynamoDB](#).

Topik

- [Bidang kontrol](#)
- [Bidang data](#)
- [DynamoDB Streams](#)
- [Transaksi](#)

Bidang kontrol

Operasi bidang kontrol memungkinkan Anda membuat dan mengelola tabel DynamoDB. Operasi ini juga memungkinkan Anda bekerja dengan indeks, aliran, dan objek lain yang tergantung pada tabel.

- `CreateTable` — Membuat tabel baru. Selain itu, Anda dapat membuat satu atau beberapa indeks sekunder, dan mengaktifkan DynamoDB Streams untuk tabel.
- `DescribeTable` — Mengembalikan informasi tentang tabel, seperti skema kunci primer, pengaturan throughput, dan informasi indeks.
- `ListTables` — Mengembalikan nama semua tabel Anda dalam daftar.
- `UpdateTable` — Memodifikasi pengaturan tabel atau indeks, menciptakan atau menghapus indeks baru pada tabel, atau memodifikasi pengaturan DynamoDB Streams untuk tabel.
- `DeleteTable` — Menghapus meja dan semua objek yang bergantung dari DynamoDB.

Bidang data

Operasi bidang data memungkinkan Anda membuat, membaca, memperbarui, dan menghapus tindakan (juga disebut CRUD) pada data dalam tabel. Beberapa operasi bidang data juga memungkinkan Anda membaca data dari indeks sekunder.

Anda dapat menggunakan [PartiQL - Bahasa kueri yang kompatibel dengan SQL untuk Amazon DynamoDB](#), untuk melakukan operasi CRUD ini atau Anda dapat menggunakan API CRUD klasik DynamoDB yang memisahkan setiap operasi menjadi panggilan API yang berbeda.

PartiQL - Bahasa kueri yang kompatibel dengan SQL

- `ExecuteStatement` – Membaca beberapa item dari tabel. Anda juga dapat menulis atau memperbarui satu item dari tabel. Saat menulis atau memperbarui satu item, Anda harus menentukan atribut kunci primer.
- `BatchExecuteStatement` – Menulis, memperbarui, atau membaca beberapa item dari tabel. Ini lebih efisien dari `ExecuteStatement` karena aplikasi Anda hanya memerlukan satu jaringan bolak-balik untuk menulis atau membaca item.

API Klasik

Membuat data

- `PutItem` – Menulis satu item pada tabel. Anda harus menentukan atribut kunci primer, tetapi Anda tidak perlu menentukan atribut lainnya.
- `BatchWriteItem` – Menulis hingga 25 item pada tabel. Ini lebih efisien dari menjalankan `PutItem` beberapa kali karena aplikasi Anda hanya memerlukan satu jaringan bolak-balik untuk menulis item.

Membaca data

- `GetItem` – Mengambil satu item dari tabel. Anda harus menentukan kunci primer untuk item yang Anda inginkan. Anda dapat mengambil seluruh item, atau hanya sebagian dari atributnya.
- `BatchGetItem` – Mengambil hingga 100 item dari satu atau beberapa tabel. Ini lebih efisien dari menjalankan `GetItem` beberapa kali karena aplikasi Anda hanya memerlukan satu jaringan bolak-balik untuk membaca item.
- `Query` – Mengambil semua item yang memiliki kunci partisi tertentu. Anda harus menentukan nilai kunci partisi. Anda dapat mengambil seluruh item, atau hanya sebagian dari atributnya. Secara opsional, Anda dapat menerapkan ketentuan pada nilai kunci urutan sehingga Anda hanya mengambil subkumpulan data yang memiliki kunci partisi yang sama. Anda dapat menggunakan operasi ini pada tabel, asalkan tabel tersebut memiliki kunci partisi dan kunci urutan. Anda juga dapat menggunakan operasi ini pada indeks, asalkan indeks tersebut memiliki kunci partisi dan kunci urutan.
- `Scan` – Mengambil semua item dalam tabel atau indeks yang ditentukan. Anda dapat mengambil seluruh item, atau hanya sebagian dari atributnya. Secara opsional, Anda dapat menerapkan kondisi pemfilteran untuk hanya mengembalikan nilai yang Anda minati dan membuang sisanya.

Memperbarui data

- `UpdateItem` — Memodifikasi satu atau beberapa atribut dalam item. Memodifikasi satu atau lebih atribut dalam suatu item. Anda harus menentukan kunci primer untuk item yang ingin Anda ubah. Anda dapat menambahkan atribut baru dan mengubah atau menghapus atribut yang sudah ada. Anda juga dapat melakukan pembaruan bersyarat, sehingga pembaruan hanya berhasil bila kondisi yang ditentukan pengguna terpenuhi. Secara opsional, Anda dapat menerapkan penghitung atom, yang menambah atau mengurangi atribut numerik tanpa mengganggu permintaan tulis lainnya.

Menghapus data

- `DeleteItem` – Menghapus satu item dari tabel. Anda harus menentukan kunci primer untuk item yang ingin Anda hapus.
- `BatchWriteItem` – Menghapus hingga 25 item dari satu atau beberapa tabel. Ini lebih efisien dari menjalankan `DeleteItem` beberapa kali karena aplikasi Anda hanya memerlukan satu jaringan bolak-balik untuk menghapus item.

Note

Anda dapat menggunakan `BatchWriteItem` untuk membuat data dan menghapus data.

DynamoDB Streams

Operasi DynamoDB Streams memungkinkan Anda mengaktifkan atau menonaktifkan stream pada tabel, dan memungkinkan akses ke catatan modifikasi data yang terkandung dalam aliran.

- `ListStreams` – Menampilkan daftar semua stream Anda, atau hanya stream untuk tabel tertentu.
- `DescribeStream` – Menampilkan informasi tentang stream, seperti Amazon Resource Name (ARN) dan di mana aplikasi Anda dapat mulai membaca beberapa catatan stream pertama.
- `GetShardIterator` – Menampilkan iterator serpihan, yang merupakan struktur data yang menggunakan aplikasi Anda untuk mengambil catatan dari stream.
- `GetRecords` – Mengambil satu atau lebih catatan stream, menggunakan iterator serpihan yang diberikan.

Transaksi

Transaksi memberikan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) yang memungkinkan Anda menjaga kebenaran data dalam aplikasi Anda dengan lebih mudah.

Anda dapat menggunakan [PartiQL - Bahasa kueri yang kompatibel dengan SQL untuk Amazon DynamoDB](#), untuk melakukan operasi transaksional atau Anda dapat menggunakan API CRUD klasik DynamoDB yang memisahkan setiap operasi menjadi panggilan API yang berbeda.

PartiQL - Bahasa kueri yang kompatibel dengan SQL

- `ExecuteTransaction`— Operasi batch yang memungkinkan operasi CRUD ke beberapa item baik di dalam maupun di seluruh tabel dengan all-or-nothing hasil yang dijamin.

API Klasik

- `TransactWriteItems`— Operasi batch yang memungkinkan `Put`, `Update`, dan `Delete` operasi ke beberapa item baik di dalam maupun di seluruh tabel dengan all-or-nothing hasil yang dijamin.
- `TransactGetItems` – Sebuah operasi batch yang memungkinkan operasi `Get` untuk mengambil beberapa item dari satu atau beberapa tabel.

Jenis data dan aturan penamaan yang didukung di Amazon DynamoDB

Bagian ini menjelaskan aturan penamaan Amazon DynamoDB dan berbagai tipe daya yang didukung DynamoDB. Ada batasan yang berlaku untuk tipe daya. Untuk informasi selengkapnya, lihat [Tipe Data](#).

Topik

- [Peraturan penamaan](#)
- [Tipe Data](#)
- [Deskriptor tipe daya](#)

Peraturan penamaan

Tabel, atribut, dan objek lain di DynamoDB harus memiliki nama. Nama harus bermakna dan ringkas—misalnya, nama-nama seperti `Produk`, `Buku`, dan `Penulis` sudah cukup jelas.

Berikut ini adalah aturan penamaan untuk DynamoDB:

- Semua nama harus dikodekan menggunakan UTF-8, dan peka huruf kapital.
- Nama tabel dan nama indeks panjangnya harus antara 3 hingga 255 karakter, dan hanya dapat berisi karakter berikut:
 - a-z
 - A-Z
 - 0-9
 - _ (garis bawah)
 - - (dasbor)
 - . (titik)
- Nama atribut harus memiliki panjang minimal satu karakter dan berukuran kurang dari 64 KB. Hal ini dianggap sebagai praktik terbaik untuk menjaga nama atribut Anda sesingkat mungkin. Hal ini membantu mengurangi penggunaan unit permintaan baca, karena nama atribut disertakan dalam pengukuran penyimpanan dan penggunaan throughput.

Berikut ini adalah pengecualiannya. Panjang nama atribut berikut tidak boleh lebih dari 255 karakter:

- Nama kunci partisi indeks sekunder
- Nama kunci urutan indeks sekunder
- Nama atribut proyeksi yang ditentukan pengguna (hanya berlaku untuk indeks sekunder lokal)

Kata-kata khusus dan karakter khusus

DynamoDB memiliki daftar kata-kata khusus dan karakter khusus. Untuk daftar lengkap, lihat [Disimpan kata-kata di DynamoDB](#). Selain itu, karakter berikut memiliki arti khusus dalam DynamoDB: # (hash) dan :(titik dua).

Meskipun DynamoDB mengizinkan Anda menggunakan kata-kata khusus dan karakter khusus ini untuk nama, kami menyarankan agar Anda menghindari melakukannya karena Anda harus menentukan variabel placeholder setiap kali Anda menggunakan nama-nama ini dalam sebuah ekspresi. Untuk informasi selengkapnya, lihat [Nama atribut ekspresi di DynamoDB](#).

Tipe Data

DynamoDB mendukung banyak tipe data berbeda untuk atribut dalam tabel. Hal tersebut dapat **dikategorikan sebagai berikut:**

Jenis data yang didukung dan aturan penamaan

Anda dapat menggunakan tipe data angka untuk mewakili tanggal atau stempel waktu. Salah satu cara untuk melakukannya adalah dengan menggunakan epoch time—jumlah detik sejak 00:00:00 UTC pada tanggal 1 Januari 1970. Sebagai contoh, jangka waktu 1437136300 mewakili 12:31:40 PM UTC pada tanggal 17 Juli 2015.

Untuk informasi selengkapnya, lihat http://en.wikipedia.org/wiki/Unix_time.

String

String adalah Unicode dengan pengodean biner UTF-8. Panjang minimum string bisa nol, jika atribut tidak digunakan sebagai kunci untuk indeks atau tabel, dan dibatasi oleh batas ukuran item DynamoDB maksimum sebesar 400 KB.

Batasan tambahan berikut berlaku untuk atribut kunci primer yang didefinisikan sebagai tipe string:

- Untuk kunci primer sederhana, panjang maksimum nilai atribut pertama (kunci partisi) adalah 2048 byte.
- Untuk kunci primer komposit, panjang maksimum nilai atribut kedua (kunci urutan) adalah 1024 byte.

DynamoDB menyusun dan membandingkan string menggunakan byte pengkodean string UTF-8 yang mendasarinya. Misalnya, "a" (0x61) lebih besar dari "A" (0x41), dan "¿" (0xC2BF) lebih besar dari "z" (0x7A).

Anda dapat menggunakan tipe data string untuk mewakili tanggal atau stempel waktu. Salah satu cara untuk melakukannya adalah dengan menggunakan string ISO 8601, seperti yang ditunjukkan dalam contoh berikut:

- 2016-02-15
- 2015-12-21T17:42:34Z
- 20150311T122706Z

Untuk informasi selengkapnya, lihat http://en.wikipedia.org/wiki/ISO_8601.

Note

Tidak seperti database relasional konvensional, DynamoDB tidak mendukung tipe data tanggal dan waktu. Akan lebih berguna jika menyimpan data tanggal dan waktu sebagai tipe data angka, menggunakan jangka waktu Unix.

Biner

Atribut tipe biner dapat menyimpan data biner apa pun, seperti teks terkompresi, data terenkripsi, atau gambar. Setiap kali DynamoDB membandingkan nilai biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak terdaftar.

Panjang atribut biner bisa nol, jika atribut tersebut tidak digunakan sebagai kunci untuk indeks atau tabel, dan dibatasi oleh batas ukuran item DynamoDB maksimum sebesar 400 KB.

Jika Anda mendefinisikan atribut kunci primer sebagai atribut tipe biner, batasan tambahan berikut akan berlaku:

- Untuk kunci primer sederhana, panjang maksimum nilai atribut pertama (kunci partisi) adalah 2048 byte.
- Untuk kunci primer komposit, panjang maksimum nilai atribut kedua (kunci urutan) adalah 1024 byte.

Aplikasi Anda harus mengkodekan nilai biner dalam format yang dikodekan base64 sebelum mengirimkannya ke DynamoDB. Setelah menerima nilai-nilai ini, DynamoDB mendekodekan data ke dalam array byte yang tidak ditandatangani dan menggunakannya sebagai panjang atribut biner.

Contoh berikut adalah atribut biner, menggunakan teks berkode base64.

```
dGhpcyB0ZXh0IG1zIGJhc2U2NC11bmNvZGVk
```

Boolean

Atribut tipe Boolean dapat menyimpan salah satu `true` atau `false`.

Null

Null melambangkan atribut dengan status tidak diketahui atau tidak ditentukan.

Jenis dokumen

Jenis dokumen adalah daftar dan peta. Tipe data ini dapat disarangkan satu sama lain, untuk mewakili struktur data kompleks hingga sedalam 32.

Tidak ada batasan jumlah nilai dalam daftar atau peta, selama item yang berisi nilai tersebut sesuai dengan batas ukuran item DynamoDB (400 KB).

Nilai atribut dapat berupa string kosong atau nilai biner kosong jika atribut tersebut tidak digunakan untuk tabel atau kunci indeks. Nilai atribut tidak boleh berupa set kosong (set string, set angka, atau set biner), namun daftar dan peta kosong diperbolehkan. String kosong dan nilai biner diperbolehkan dalam daftar dan peta. Untuk informasi selengkapnya, lihat [Atribut](#).

Daftar

Atribut jenis daftar dapat menyimpan kumpulan nilai yang diurutkan. Daftar diapit tanda kurung siku: [...]

Daftar mirip dengan array JSON. Tidak ada batasan pada tipe data yang dapat disimpan dalam elemen daftar, dan elemen dalam elemen daftar tidak harus bertipe sama.

Contoh berikut memperlihatkan daftar yang berisi dua string dan sebuah angka.

```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

Note

DynamoDB memungkinkan Anda bekerja dengan elemen individual dalam daftar, meskipun elemen tersebut sangat bersarang. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#).

Peta

Atribut jenis peta dapat menyimpan kumpulan pasangan nama-nilai yang tidak berurutan. Peta diapit kurung kurawal: { ... }

Peta serupa dengan objek JSON. Tidak ada batasan pada tipe data yang dapat disimpan dalam elemen peta, dan elemen dalam peta tidak harus bertipe sama.

Peta ideal untuk menyimpan dokumen JSON di DynamoDB. Contoh berikut memperlihatkan peta yang berisi string, angka, dan daftar bertumpuk yang berisi peta lain.

```
{
  Day: "Monday",
  UnreadEmails: 42,
  ItemsOnMyDesk: [
    "Coffee Cup",
    "Telephone",
    {
      Pens: { Quantity : 3},
      Pencils: { Quantity : 2},
      Erasers: { Quantity : 1}
    }
  ]
}
```

Note

DynamoDB memungkinkan Anda bekerja dengan elemen individual dalam peta, meskipun elemen tersebut sangat besar. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#).

Set

DynamoDB mendukung jenis yang mewakili set angka, string, atau nilai-nilai biner. Semua elemen di dalam set harus dari jenis yang sama. Misalnya, Set Angka hanya dapat berisi angka dan Set String hanya dapat berisi string.

Tidak ada batasan jumlah nilai dalam set, selama item yang berisi nilai tersebut sesuai dengan batas ukuran item DynamoDB (400 KB).

Setiap nilai dalam suatu set harus unik. Urutan nilai dalam suatu set tidak dipertahankan. Oleh karena itu, aplikasi Anda tidak boleh bergantung pada urutan elemen tertentu dalam set tersebut. DynamoDB tidak mendukung set kosong, namun string kosong dan nilai biner diperbolehkan dalam satu set.

Contoh berikut menunjukkan set string, set angka, dan set biner:

```
["Black", "Green", "Red"]
```

```
[42.2, -19, 7.5, 3.14]
```

```
["U3Vubnk=", "UmFpbnk=", "U25vd3k="]
```

Deskriptor tipe daya

Protokol API DynamoDB tingkat rendah menggunakan Deskriptor tipe data sebagai token yang memberi tahu DynamoDB cara menafsirkan setiap atribut.

Berikut ini adalah daftar lengkap deskriptor tipe daya DynamoDB:

- **S** – String
- **N** – Nomor
- **B** – Biner
- **BOOL** – Boolean
- **NULL** – Null
- **M** – Peta
- **L** – Daftar
- **SS** – Set String
- **NS** – Set Nomor
- **BS** – Set Biner

Konsistensi baca

Amazon DynamoDB membaca data dari tabel, indeks sekunder lokal (LSI), indeks sekunder global (GSI), dan aliran. Untuk informasi selengkapnya, lihat [Komponen inti dari Amazon DynamoDB](#). Tabel dan LSI menyediakan dua opsi konsistensi pembacaan: pembacaan yang pada akhirnya konsisten (default) dan pembacaan sangat konsisten. Semua pembacaan dari GSI dan streaming pada akhirnya konsisten.

Saat aplikasi Anda menulis data ke tabel DynamoDB dan menerima respons HTTP 200 (OK), itu berarti penulisan berhasil diselesaikan dan bertahan lama. DynamoDB menyediakan isolasi komitmen baca dan memastikan bahwa operasi baca selalu mengembalikan nilai komitmen untuk suatu item. Pembacaan tidak akan pernah menampilkan tampilan item dari penulisan yang pada

akhirnya tidak berhasil. Isolasi komitmen baca tidak mencegah modifikasi item segera setelah operasi baca.

Bacaan Akhir Konsisten

Akhir konsisten adalah model konsisten baca default untuk semua operasi baca. Saat mengeluarkan pembacaan yang konsisten pada tabel atau indeks DynamoDB, responsnya mungkin tidak mencerminkan hasil operasi penulisan yang baru saja diselesaikan. Jika Anda mengulangi permintaan baca setelah beberapa saat, responsnya pada akhirnya akan mengembalikan item yang lebih baru. Bacaan akhir konsisten didukung pada tabel, indeks sekunder lokal, dan indeks sekunder global. Perhatikan juga bahwa semua pembacaan dari aliran DynamoDB pada akhirnya juga konsisten.

Bacaan akhir konsisten hanya menghabiskan separuh biaya dari bacaan sangat konsisten. Untuk informasi selengkapnya, lihat Harga [Amazon DynamoDB](#).

Bacaan Sangat Konsisten

Baca operasi seperti `GetItem`, `Query`, dan `Scan` menyediakan parameter `ConsistentRead` opsional. Jika Anda menyetel `ConsistentRead` ke `true`, DynamoDB mengembalikan respons dengan data up-to-date terbanyak, yang mencerminkan pembaruan dari semua operasi penulisan sebelumnya yang berhasil. Pembacaan yang sangat konsisten hanya didukung pada tabel dan indeks sekunder lokal. Pembacaan yang sangat konsisten dari indeks sekunder global atau aliran DynamoDB tidak didukung.

Tabel global membaca konsistensi

DynamoDB juga mendukung [tabel global](#) untuk replikasi multi-aktif dan Multi-wilayah. Tabel global terdiri dari beberapa tabel replika di AWS Wilayah yang berbeda. Setiap perubahan yang dilakukan pada item apa pun di tabel replika mana pun akan direplikasi ke semua replika lain dalam tabel global yang sama, biasanya dalam satu detik, dan pada akhirnya konsisten. Untuk informasi selengkapnya, lihat [Konsistensi dan resolusi konflik](#).

Mode kapasitas baca/tulis

Amazon DynamoDB memiliki dua mode kapasitas baca/tulis untuk memproses baca dan tulis pada tabel Anda:

- Sesuai permintaan
- Disediakan (default, memenuhi syarat tingkat gratis)

Mode kapasitas baca/tulis mengontrol cara Anda dikenakan biaya untuk throughput baca dan tulis serta cara Anda mengelola kapasitas. Anda dapat mengatur mode kapasitas baca/tulis saat membuat tabel atau Anda dapat mengubahnya nanti.

Indeks sekunder mewarisi mode kapasitas baca/tulis dari tabel dasar. Untuk informasi selengkapnya, lihat [Pertimbangan saat mengubah Mode Kapasitas baca/tulis](#).

Video berikut akan memberi Anda gambaran pengantar tentang mode kapasitas tabel.

[Mode kapasitas tabel](#)

Untuk informasi selengkapnya tentang praktik terbaik dalam mengoptimalkan biaya tabel DynamoDB Anda, lihat [Mengoptimalkan biaya pada tabel DynamoDB](#).

Topik

- [Mode sesuai permintaan](#)
- [Mode yang disediakan](#)

Mode sesuai permintaan

Amazon DynamoDB sesuai permintaan adalah opsi penagihan fleksibel yang mampu melayani ribuan permintaan per detik tanpa perencanaan kapasitas. DynamoDB on-demand pay-per-request menawarkan harga untuk permintaan baca dan tulis sehingga Anda hanya membayar untuk apa yang Anda gunakan.

Saat Anda memilih mode sesuai permintaan, DynamoDB langsung mengakomodasi beban kerja Anda saat beban kerja tersebut meningkat atau menurun ke tingkat lalu lintas yang dicapai sebelumnya. Jika tingkat lalu lintas beban kerja mencapai puncak baru, DynamoDB beradaptasi dengan cepat untuk mengakomodasi beban kerja. Tabel yang menggunakan mode sesuai permintaan memberikan latensi milidetik satu digit, komitmen perjanjian tingkat layanan (SLA), dan keamanan yang sama dengan yang sudah ditawarkan DynamoDB. Anda dapat memilih sesuai permintaan untuk tabel baru dan yang sudah ada dan Anda dapat terus menggunakan API DynamoDB yang ada tanpa mengubah kode.

Mode sesuai permintaan adalah pilihan yang baik jika salah satu hal berikut ini benar:

- Anda membuat tabel baru dengan beban kerja yang tidak diketahui.
- Anda memiliki lalu lintas aplikasi yang tidak dapat diprediksi.
- Anda lebih menyukai kemudahan membayar hanya untuk apa yang Anda gunakan.

Tingkat permintaan hanya dibatasi oleh kuota tabel default throughput DynamoDB, namun dapat dinaikkan berdasarkan permintaan. Untuk informasi selengkapnya, lihat [Kuota default throughput](#).

Untuk memulai sesuai permintaan, Anda dapat membuat atau memperbarui tabel untuk menggunakan mode sesuai permintaan. Untuk informasi selengkapnya, lihat [Operasi dasar pada tabel DynamoDB](#).

Tabel dapat dialihkan ke mode sesuai permintaan setiap 24 jam sekali. Membuat tabel sesuai permintaan juga memulai periode 24 jam ini. Tabel dapat dikembalikan ke mode kapasitas yang disediakan kapan saja. Untuk masalah yang perlu dipertimbangkan saat mengganti mode kapasitas baca/tulis, lihat [Pertimbangan saat mengubah Mode Kapasitas baca/tulis](#).

Topik

- [Unit permintaan baca dan unit permintaan tulis](#)
- [Properti lalu lintas puncak dan penskalaan](#)
- [Throughput awal untuk mode kapasitas sesuai permintaan](#)
- [Perilaku tabel saat mengganti mode kapasitas baca/tulis](#)
- [Pemanasan awal tabel untuk mode kapasitas sesuai permintaan](#)

Unit permintaan baca dan unit permintaan tulis

Untuk tabel mode on-demand, Anda tidak perlu menentukan berapa banyak throughput baca dan tulis yang Anda harapkan untuk dijalankan oleh aplikasi Anda. DynamoDB menagih Anda untuk membaca dan menulis yang dilakukan aplikasi Anda pada tabel Anda dalam hal unit permintaan baca dan unit permintaan tulis.


Permintaan baca DynamoDB bisa sangat konsisten, pada akhirnya konsisten, atau bersifat transaksional.

- Permintaan bacaan sangat konsisten untuk suatu item hingga 4 KB memerlukan satu unit permintaan baca.
- Permintaan bacaan akhir konsisten hingga 4 KB memerlukan setengah unit permintaan baca.
- Permintaan baca transaksional suatu item hingga 4 KB memerlukan dua unit permintaan baca.

Jika Anda perlu membaca item yang lebih besar dari 4 KB, DynamoDB memerlukan unit permintaan baca tambahan. Jumlah total unit permintaan baca yang diperlukan bergantung pada ukuran item,

dan apakah Anda menginginkan bacaan akhir konsisten atau bacaan sangat konsisten. Misalnya, jika ukuran item Anda adalah 8 KB, Anda memerlukan 2 unit permintaan baca untuk mempertahankan satu unit permintaan baca yang sangat konsisten, 1 unit permintaan baca jika Anda akhirnya memilih pembacaan yang konsisten, atau 4 unit permintaan baca untuk permintaan baca transaksional.

Untuk mempelajari selengkapnya tentang model konsistensi baca DynamoDB, lihat [Konsistensi baca](#).

 Important

Jika Anda melakukan operasi baca pada item yang tidak ada, DynamoDB akan tetap menggunakan throughput baca seperti diuraikan di atas.

Satu unit permintaan tulis mewakili satu penulisan untuk item berukuran hingga 1 KB. Jika Anda perlu menulis item yang lebih besar dari 1 KB, DynamoDB perlu menggunakan unit permintaan tulis tambahan. Permintaan tulis transaksional memerlukan 2 unit permintaan tulis untuk melakukan satu penulisan untuk item hingga 1 KB. Jumlah total unit permintaan tulis yang diperlukan bergantung pada ukuran item. Misalnya, jika ukuran item Anda adalah 2 KB, Anda memerlukan 2 unit permintaan tulis untuk mempertahankan satu permintaan tulis atau 4 unit permintaan tulis untuk permintaan tulis transaksional.

Untuk contoh harga terperinci dan memperkirakan biaya menggunakan kalkulator harga, lihat [Harga Amazon DynamoDB](#).

Properti lalu lintas puncak dan penskalaan

Tabel DynamoDB yang menggunakan mode kapasitas sesuai permintaan secara otomatis beradaptasi dengan volume lalu lintas aplikasi Anda. Mode kapasitas sesuai permintaan secara instan mengakomodasi hingga dua kali lipat lalu lintas puncak sebelumnya pada sebuah tabel. Misalnya, jika pola lalu lintas aplikasi Anda bervariasi antara 25.000 dan 50.000 bacaan sangat konsisten per detik dengan 50.000 pembacaan per detik adalah puncak lalu lintas sebelumnya, mode kapasitas sesuai permintaan langsung mengakomodasi lalu lintas berkelanjutan hingga 100.000 pembacaan per detik. Jika aplikasi Anda mempertahankan lalu lintas sebanyak 100.000 pembacaan per detik, puncak tersebut akan menjadi puncak baru Anda sebelumnya, sehingga memungkinkan lalu lintas berikutnya mencapai hingga 200.000 pembacaan per detik.

Jika Anda memerlukan lebih dari dua kali lipat puncak tabel sebelumnya, DynamoDB secara otomatis mengalokasikan lebih banyak kapasitas seiring peningkatan volume lalu lintas untuk membantu memastikan bahwa beban kerja Anda tidak mengalami throttling. Namun, throttling dapat terjadi jika

Anda melebihi dua kali lipat puncak sebelumnya dalam waktu 30 menit. Misalnya, jika pola lalu lintas aplikasi Anda bervariasi antara 25.000 dan 50.000 pembacaan per detik yang sangat konsisten di mana 50.000 pembacaan per detik adalah puncak lalu lintas yang dicapai sebelumnya, DynamoDB merekomendasikan untuk mengatur jarak pertumbuhan lalu lintas Anda setidaknya selama 30 menit sebelum mendorong lebih dari 100.000 pembacaan per detik.

Throughput awal untuk mode kapasitas sesuai permintaan

Jika Anda baru saja mengalihkan tabel yang ada ke mode kapasitas sesuai permintaan untuk pertama kalinya, atau jika Anda membuat tabel baru dengan mode kapasitas sesuai permintaan diaktifkan, tabel tersebut memiliki pengaturan puncak sebelumnya sebagai berikut, meskipun tabel tersebut belum melayani lalu lintas sebelumnya menggunakan mode kapasitas sesuai permintaan:

Berikut ini adalah contoh skenario yang mungkin terjadi.

- Tabel yang disediakan dikonfigurasi sebagai 100 WCU dan 100 RCU. Saat tabel ini dialihkan ke sesuai permintaan untuk pertama kalinya, DynamoDB akan memastikan skalanya diperluas untuk secara instan mempertahankan setidaknya 4.000 unit tulis/detik dan 12.000 unit baca/detik.
- Tabel yang disediakan dikonfigurasi sebagai 8.000 WCU dan 24.000 RCU. Ketika tabel ini dialihkan ke sesuai permintaan, tabel ini akan terus mampu mempertahankan setidaknya 8.000 unit tulis/detik dan 24.000 unit baca/detik setiap saat.
- Tabel yang disediakan dikonfigurasi dengan 8.000 WCU dan 24.000 RCU, yang menggunakan 6.000 unit tulis/dtk dan 18.000 unit baca/dtk untuk jangka waktu berkelanjutan. Ketika tabel ini dialihkan ke sesuai permintaan, tabel ini akan terus mampu mempertahankan setidaknya 8.000 unit tulis/detik dan 24.000 unit baca/detik. Lalu lintas sebelumnya memungkinkan tabel untuk mempertahankan tingkat lalu lintas yang jauh lebih tinggi tanpa throttling.
- Tabel yang sebelumnya tersedia 10.000 WCU dan 10.000 RCU, namun saat ini tersedia 10 RCU dan 10 WCU. Jika tabel ini dialihkan ke sesuai permintaan, tabel ini akan mampu mempertahankan setidaknya 10.000 unit tulis/detik dan 10.000 unit baca/detik.

Perilaku tabel saat mengganti mode kapasitas baca/tulis

Saat Anda mengalihkan tabel dari mode kapasitas yang disediakan ke mode kapasitas sesuai permintaan, DynamoDB membuat beberapa perubahan pada struktur tabel dan partisi Anda. Proses ini dapat memakan waktu beberapa menit. Selama periode peralihan, tabel Anda memberikan throughput yang konsisten dengan jumlah unit kapasitas tulis dan kapasitas baca yang disediakan sebelumnya. Saat beralih dari mode kapasitas sesuai permintaan kembali ke mode kapasitas yang

disediakan, tabel Anda memberikan throughput yang konsisten dengan puncak sebelumnya yang dicapai ketika tabel diatur ke mode kapasitas sesuai permintaan.

Pemanasan awal tabel untuk mode kapasitas sesuai permintaan

Dengan mode kapasitas sesuai permintaan, permintaan dapat melonjak hingga dua kali lipat dari puncak sebelumnya di tabel. Perhatikan bahwa pembatasan dapat terjadi jika permintaan melonjak lebih dari dua kali lipat kapasitas default atau tingkat permintaan puncak yang dicapai sebelumnya dalam waktu 30 menit. Salah satu solusinya adalah dengan menghangatkan meja terlebih dahulu hingga mencapai kapasitas puncak lonjakan yang diantisipasi.

Untuk memanaskan tabel terlebih dahulu, ikuti langkah-langkah berikut:

1. Pastikan untuk memeriksa batas akun Anda dan konfirmasi bahwa Anda dapat mencapai kapasitas yang diinginkan dalam mode yang disediakan.
2. Jika Anda melakukan pemanasan awal pada tabel yang sudah ada, atau tabel baru dalam mode sesuai permintaan, mulailah proses ini setidaknya 24 jam sebelum perkiraan puncak. Anda hanya dapat beralih antara mode sesuai permintaan dan mode disediakan satu kali setiap 24 jam.
3. Untuk melakukan pra-pemanasan tabel yang saat ini dalam mode sesuai permintaan, alihkan ke mode yang disediakan dan tunggu hingga tabel aktif. Lalu lanjutkan ke langkah berikutnya.

Jika Anda ingin melakukan pemanasan awal pada meja baru yang berada dalam mode yang disediakan, atau telah berada dalam mode yang disediakan selama 24 jam, Anda dapat melanjutkan ke langkah berikutnya tanpa menunggu.

4. Atur throughput penulisan tabel ke nilai puncak yang diinginkan, dan simpan di sana selama beberapa menit. Anda akan dikenakan biaya akibat volume throughput yang tinggi ini hingga Anda beralih kembali ke sesuai permintaan.
5. Beralih ke mode kapasitas Sesuai Permintaan. Hal ini harus mempertahankan nilai kapasitas throughput yang disediakan.

Mode yang disediakan

Jika Anda memilih mode yang disediakan, Anda menentukan jumlah baca dan tulis per detik yang Anda perlukan untuk aplikasi Anda. Anda dapat menggunakan penskalaan otomatis untuk menyesuaikan kapasitas yang disediakan tabel Anda secara otomatis sebagai respons terhadap perubahan lalu lintas. Hal ini membantu Anda mengatur penggunaan DynamoDB agar tetap pada atau di bawah tingkat permintaan yang ditentukan untuk memperoleh prediktabilitas biaya.

Mode yang disediakan adalah pilihan yang baik jika salah satu hal berikut ini benar:

- Anda memiliki lalu lintas aplikasi yang dapat diprediksi.
- Anda menjalankan aplikasi yang lalu lintasnya konsisten atau meningkat secara bertahap.
- Anda dapat memperkirakan kebutuhan kapasitas untuk mengendalikan biaya.

Unit kapasitas baca dan unit kapasitas tulis

Untuk tabel mode yang disediakan, Anda menentukan kapasitas throughput dalam bentuk unit kapasitas baca (RCU) dan unit kapasitas tulis (WCU):

- Satu unit kapasitas baca mewakili satu bacaan konsisten yang kuat per detik, atau dua bacaan akhir konsisten per detik, untuk sebuah item hingga ukuran 4 KB. Permintaan baca transaksional memerlukan dua unit kapasitas baca untuk melakukan satu pembacaan per detik untuk item hingga 4 KB. Jika Anda perlu membaca item yang lebih besar dari 4 KB, DynamoDB harus menggunakan unit kapasitas baca tambahan. Jumlah total unit kapasitas baca yang diperlukan bergantung pada ukuran item, dan apakah Anda menginginkan pembacaan yang konsisten atau sangat konsisten. Misalnya, jika ukuran item Anda adalah 8 KB, Anda memerlukan 2 unit kapasitas baca untuk mempertahankan satu bacaan sangat konsisten per detik, 1 unit kapasitas baca jika Anda memilih bacaan akhir konsisten, atau 4 unit kapasitas baca untuk permintaan baca transaksional. Untuk informasi selengkapnya, lihat [Konsumsi unit kapasitas untuk bacaan](#).

Note

Untuk mempelajari selengkapnya tentang model konsistensi baca DynamoDB, lihat [Konsistensi baca](#).

- Satu unit kapasitas tulis mewakili satu tulis per detik untuk item hingga ukuran 1 KB. Jika Anda perlu menulis item yang lebih besar dari 1 KB, DynamoDB harus menggunakan unit kapasitas tulis tambahan. Permintaan tulis transaksional memerlukan 2 unit kapasitas tulis untuk melakukan satu penulisan per detik untuk item hingga 1 KB. Jumlah total unit kapasitas tulis yang diperlukan bergantung pada ukuran item. Misalnya, jika ukuran item Anda adalah 2 KB, Anda memerlukan 2 unit kapasitas tulis untuk mempertahankan satu permintaan tulis per detik atau 4 unit kapasitas tulis untuk permintaan tulis transaksional. Untuk informasi selengkapnya, lihat [Konsumsi unit kapasitas untuk penulisan](#).

⚠ Important

Ketika menjalankan `DescribeTable` pada tabel sesuai permintaan, unit kapasitas baca dan unit kapasitas tulis ditetapkan ke 0.

Jika aplikasi Anda membaca atau menulis item yang lebih besar (hingga ukuran item maksimum DynamoDB sebesar 400 KB), akan menggunakan lebih banyak unit kapasitas.

Misalnya, Anda membuat tabel yang tersedia dengan 6 unit kapasitas baca dan 6 unit kapasitas tulis. Dengan pengaturan ini, aplikasi Anda dapat melakukan hal berikut:

- Lakukan bacaan sangat konsisten hingga 24 KB per detik (unit kapasitas baca $4 \text{ KB} \times 6$).
- Lakukan pembacaan yang konsisten hingga 48 KB per detik (throughput baca dua kali lebih banyak).
- Lakukan permintaan baca transaksional hingga 12 KB per detik.
- Tulis hingga 6 KB per detik ($1 \text{ KB} \times 6$ unit kapasitas tulis).
- Lakukan permintaan tulis transaksional hingga 3 KB per detik.

Untuk informasi selengkapnya, lihat [Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB](#).

Throughput yang disediakan adalah jumlah kapasitas maksimum yang dapat digunakan aplikasi dari tabel atau indeks. Jika aplikasi Anda melebihi kapasitas throughput yang disediakan pada tabel atau indeks, aplikasi tersebut akan dikenakan throttling permintaan.

Pembatasan mencegah aplikasi Anda menggunakan terlalu banyak unit kapasitas. Ketika permintaan di-throttle, permintaan tersebut gagal dengan kode HTTP 400 (Bad Request) dan `ProvisionedThroughputExceededException`. AWS SDK memiliki dukungan bawaan untuk mencoba kembali permintaan yang dibatasi (lihat [Percobaan ulang kesalahan dan mundur eksponensial](#)), jadi Anda tidak perlu menulis logika ini sendiri. Untuk informasi selengkapnya tentang cara mengatasi masalah throttling, lihat [Mengapa tabel Amazon DynamoDB saya di-throttle?](#)

Anda dapat menggunakan AWS Management Console untuk memantau throughput yang disediakan dan aktual, dan untuk memodifikasi pengaturan throughput Anda jika perlu.

Untuk contoh harga terperinci dan memperkirakan biaya menggunakan kalkulator harga, lihat [Harga Amazon DynamoDB](#).

Penskalaan otomatis DynamoDB

Penskalaan otomatis DynamoDB secara aktif mengelola kapasitas throughput untuk tabel dan indeks sekunder global. Dengan penskalaan otomatis, Anda menentukan rentang (batas atas dan bawah) untuk unit kapasitas baca dan tulis. Anda juga menentukan persentase pemanfaatan target dalam rentang tersebut. Penskalaan otomatis DynamoDB berupaya mempertahankan pemanfaatan target Anda, bahkan saat beban kerja aplikasi Anda meningkat atau menurun.

Dengan penskalaan otomatis DynamoDB, tabel atau indeks sekunder global dapat meningkatkan kapasitas baca dan tulis yang disediakan untuk menangani peningkatan lalu lintas secara tiba-tiba, tanpa pembatasan permintaan. Ketika beban kerja berkurang, penskalaan otomatis DynamoDB dapat menurunkan throughput sehingga Anda tidak membayar kapasitas tersedia yang tidak terpakai.

Note

Jika Anda menggunakan AWS Management Console untuk membuat tabel atau indeks sekunder global, DynamoDB auto scaling diaktifkan secara default.

Anda dapat mengelola pengaturan penskalaan otomatis kapan saja dengan menggunakan konsol, SDK AWS CLI, atau salah satu AWS SDK.

Untuk informasi selengkapnya, lihat [Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB](#).

Kapasitas terpesan

[Sebagai pelanggan DynamoDB, Anda dapat membeli kapasitas terpesan terlebih dahulu untuk tabel yang menggunakan kelas tabel Standar DynamoDB, seperti yang dijelaskan pada Harga Amazon DynamoDB](#). Dengan kapasitas yang dicadangkan, Anda membayar biaya satu kali di muka dan berkomitmen pada tingkat penggunaan minimum yang disediakan selama jangka waktu tertentu. Kapasitas cadangan Anda ditagih berdasarkan tarif kapasitas cadangan per jam. Dengan memesan unit kapasitas baca dan tulis terlebih dahulu, Anda mewujudkan penghematan biaya yang signifikan pada biaya kapasitas yang disediakan. Kapasitas apa pun yang Anda sediakan melebihi kapasitas cadangan Anda akan ditagih sesuai tarif kapasitas standar yang disediakan.

Diskon kapasitas cadangan diterapkan terlebih dahulu ke akun yang membeli kapasitas cadangan. Setiap diskon kapasitas cadangan yang tidak terpakai kemudian diterapkan ke akun lain di AWS organisasi yang sama dengan akun pembelian. Anda dapat menonaktifkan pembagian diskon

Instans Cadangan di halaman Preferensi di konsol Manajemen Penagihan dan Biaya. Untuk informasi selengkapnya, lihat [Menonaktifkan instans terpesan dan pembagian diskon Savings Plans](#).

Note

Kapasitas cadangan tidak tersedia untuk unit kapasitas tulis yang direplikasi. Kapasitas Terpesan hanya diterapkan pada wilayah tempat pembeliannya. Kapasitas terpesan juga tidak tersedia untuk tabel yang menggunakan kelas tabel DynamoDB Standard-IA atau mode kapasitas sesuai permintaan.

Untuk mengelola kapasitas terpesan, buka [Konsol DynamoDB](#) dan pilih Kapasitas Terpesan.

Note

Anda dapat mencegah pengguna melihat atau membeli kapasitas yang dipesan, sambil tetap mengizinkan mereka mengakses konsol lainnya. Untuk informasi lebih lanjut, lihat "Memberikan Izin untuk Mencegah Pembelian Penawaran Kapasitas Cadangan" di [Manajemen Identitas dan Akses untuk Amazon DynamoDB](#).

Untuk informasi selengkapnya tentang harga, lihat [Harga Amazon DynamoDB](#).

Kelas tabel

DynamoDB menawarkan dua kelas tabel yang dirancang untuk membantu Anda mengoptimalkan biaya. Kelas tabel Standar DynamoDB adalah defaultnya, dan direkomendasikan untuk sebagian besar beban kerja. Kelas tabel Akses Jarang Standar DynamoDB (DynamoDB Standard-IA) dioptimalkan untuk tabel di mana penyimpanan merupakan biaya dominan. Misalnya, tabel yang menyimpan data yang jarang diakses, seperti log aplikasi, postingan media sosial lama, riwayat pesanan e-niaga, dan pencapaian game sebelumnya, merupakan kandidat yang baik untuk kelas tabel Standard-IA. Lihat [Perincian Amazon DynamoDB](#) untuk mendapatkan rincian harga.

Setiap tabel DynamoDB dikaitkan dengan kelas tabel (DynamoDB Standard secara default). Semua indeks sekunder yang terkait dengan tabel menggunakan kelas tabel yang sama. Setiap kelas tabel menawarkan harga berbeda untuk penyimpanan data serta permintaan baca dan tulis. Anda dapat memilih kelas tabel yang paling hemat biaya untuk tabel Anda berdasarkan pola penggunaan penyimpanan dan throughputnya.

Pilihan kelas tabel tidak permanen—Anda dapat mengubah pengaturan ini menggunakan, AWS Management Console AWS CLI, atau SDK. AWS DynamoDB juga mendukung pengelolaan kelas tabel Anda AWS CloudFormation menggunakan tabel Single-region dan tabel global. Untuk mempelajari selengkapnya tentang memilih kelas tabel Anda, lihat [Pertimbangan saat memilih kelas tabel](#).

Partisi dan distribusi data

Amazon DynamoDB menyimpan data dalam partisi. Partisi adalah alokasi penyimpanan untuk tabel, didukung oleh solid state drive (SSD) dan secara otomatis direplikasi di beberapa Availability Zone dalam suatu Wilayah. AWS Manajemen partisi ditangani sepenuhnya oleh DynamoDB—Anda tidak perlu mengelola partisi sendiri.

Saat Anda membuat tabel, status awal tabel tersebut adalah CREATING. Selama fase ini, DynamoDB mengalokasikan partisi yang memadai ke tabel sehingga dapat menangani persyaratan throughput yang Anda sediakan. Anda dapat mulai menulis dan membaca data tabel setelah status tabel berubah menjadi ACTIVE.

DynamoDB mengalokasikan partisi tambahan ke tabel dalam situasi berikut:

- Jika Anda meningkatkan pengaturan throughput tabel yang tersedia melebihi apa yang dapat didukung oleh partisi yang ada.
- Jika partisi yang ada memenuhi kapasitas dan diperlukan lebih banyak ruang penyimpanan.

Manajemen partisi terjadi secara otomatis di latar belakang dan transparan bagi aplikasi Anda. Tabel Anda tetap tersedia dan sepenuhnya mendukung persyaratan throughput yang Anda sediakan.

Untuk rincian selengkapnya, lihat [Desain kunci partisi](#).

Indeks sekunder global di DynamoDB juga terdiri dari partisi. Data dalam indeks sekunder global disimpan secara terpisah dari data dalam tabel dasarnya, namun partisi indeks berperilaku sama seperti partisi tabel.

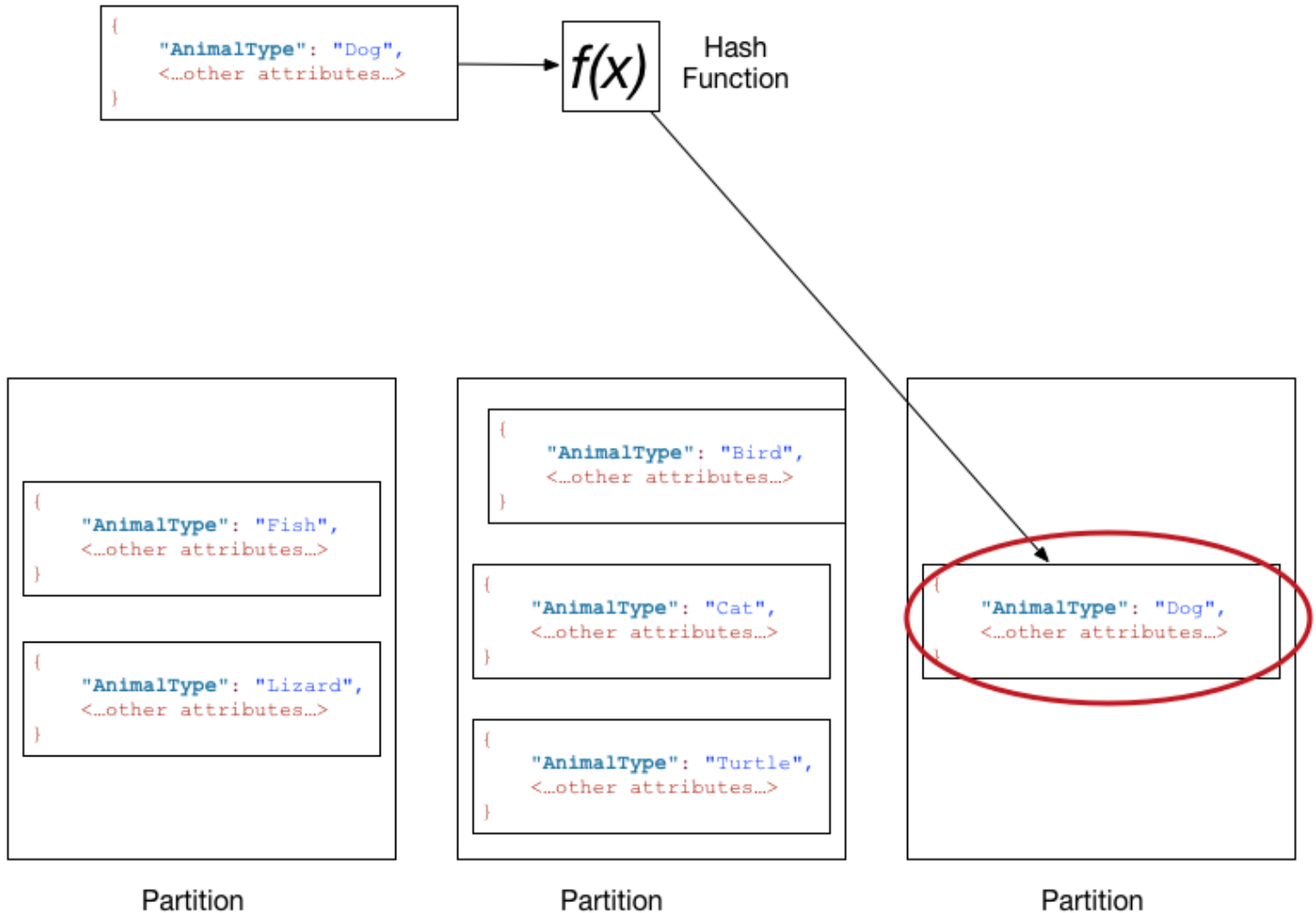
Distribusi data: Kunci partisi

Jika tabel Anda memiliki kunci primer sederhana (hanya kunci partisi), DynamoDB menyimpan dan mengambil setiap item berdasarkan nilai kunci partisinya.

Untuk menulis item ke tabel, DynamoDB menggunakan nilai kunci partisi sebagai input ke fungsi hash internal. Nilai keluaran dari fungsi hash menentukan partisi tempat item akan disimpan.

Untuk membaca item dari tabel, Anda harus menentukan nilai kunci partisi untuk item tersebut. DynamoDB menggunakan nilai ini sebagai masukan ke fungsi hashnya, menghasilkan partisi tempat item dapat ditemukan.

Diagram berikut menunjukkan tabel bernama Pets, yang mencakup beberapa partisi. Kunci utama tabel adalah `AnimalType` (hanya atribut kunci ini yang ditampilkan). DynamoDB menggunakan fungsi hashnya untuk menentukan tempat menyimpan item baru, dalam hal ini berdasarkan nilai hash string `Dog`. Perhatikan bahwa item tidak disimpan dalam urutan yang diurutkan. Lokasi setiap item ditentukan oleh nilai hash kunci partisinya.



Note

DynamoDB dioptimalkan untuk distribusi item yang seragam di seluruh partisi tabel, tidak peduli berapa banyak partisi yang ada. Kami menyarankan Anda memilih kunci partisi yang

dapat memiliki sejumlah besar nilai berbeda yang berhubungan dengan jumlah item dalam tabel.

Distribusi data: Kunci partisi dan kunci urutan

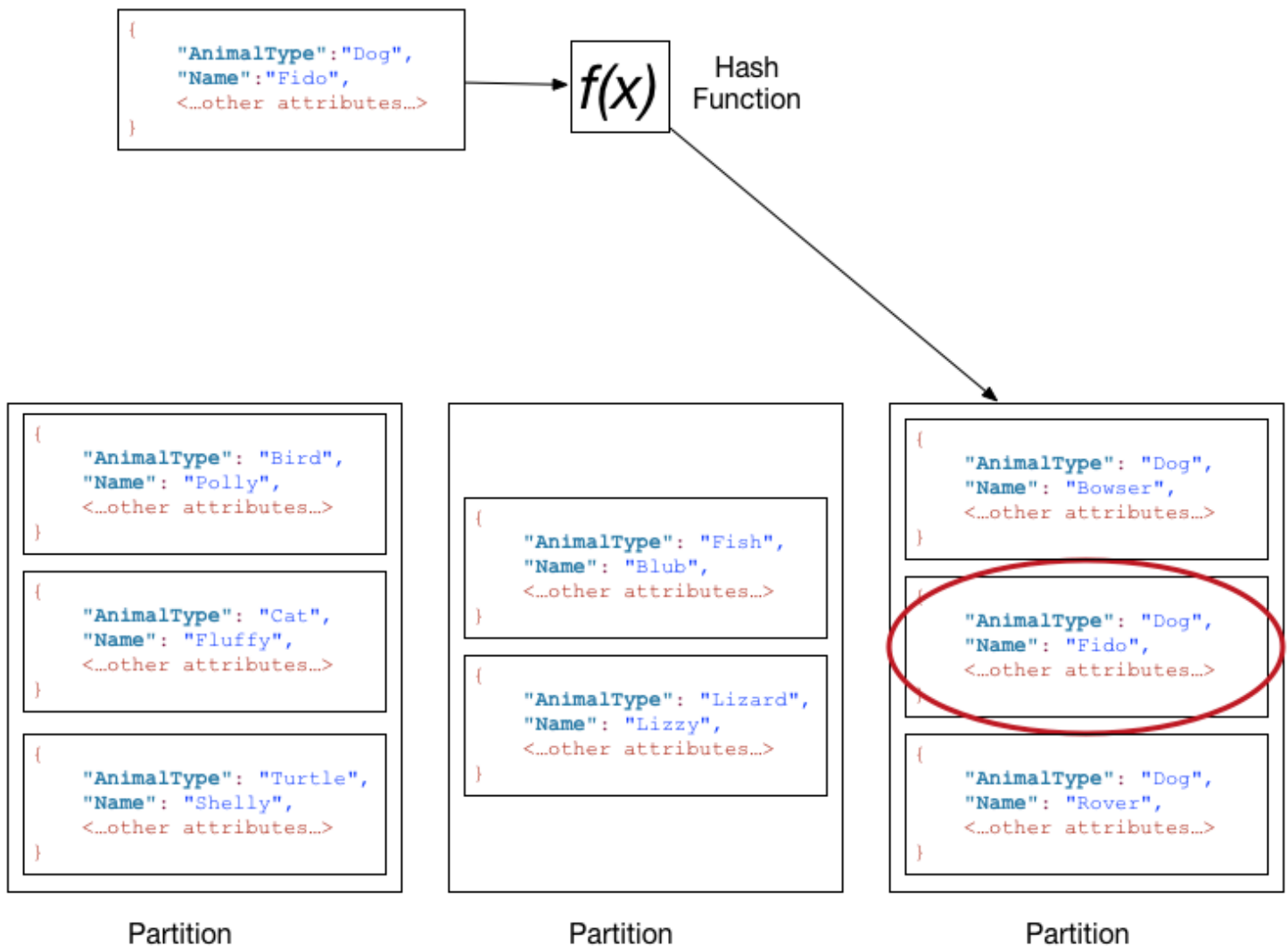
Jika tabel memiliki kunci primer gabungan (kunci partisi dan kunci urutan), DynamoDB menghitung nilai hash kunci partisi dengan cara yang sama seperti yang dijelaskan dalam [Distribusi data: Kunci partisi](#). Namun, ia cenderung menjaga item yang memiliki nilai kunci partisi yang sama berdekatan dan diurutkan berdasarkan nilai atribut kunci sortir. Kumpulan item yang memiliki nilai kunci partisi yang sama disebut koleksi item. Koleksi item dioptimalkan untuk pengambilan rentang item yang efisien dalam koleksi. Jika tabel Anda tidak memiliki indeks sekunder lokal, DynamoDB akan secara otomatis membagi koleksi item Anda menjadi partisi sebanyak yang diperlukan untuk menyimpan data dan menyajikan throughput baca dan tulis.

Untuk menulis item ke tabel, DynamoDB menghitung nilai hash kunci partisi untuk menentukan partisi mana yang harus berisi item tersebut. Di partisi itu, beberapa item mungkin memiliki nilai kunci partisi yang sama. Jadi DynamoDB menyimpan item tersebut di antara item lainnya dengan kunci partisi yang sama, dalam urutan menaik berdasarkan kunci urutan.

Untuk membaca item dari tabel, Anda harus menentukan nilai kunci partisi dan nilai kunci urutan. DynamoDB menghitung nilai hash kunci partisi, menghasilkan partisi tempat item dapat ditemukan.

Anda dapat membaca beberapa item dari tabel dalam satu operasi (`Query`) jika item yang Anda inginkan memiliki nilai kunci partisi yang sama. DynamoDB mengembalikan semua item dengan nilai kunci partisi tersebut. Secara opsional, Anda dapat menerapkan ketentuan pada kunci urutan sehingga hanya mengembalikan item dalam rentang nilai tertentu.

Misalkan tabel `Pets` memiliki kunci primer komposit yang terdiri dari `AnimalType`(kunci partisi) dan `Nama` (kunci sortir). Diagram berikut menunjukkan DynamoDB menulis item dengan nilai kunci partisi `Dog` dan nilai kunci urutan `Fido`.



Untuk membaca item yang sama dari tabel Pets, DynamoDB menghitung nilai hash dari Dog, menghasilkan partisi tempat item tersebut disimpan. DynamoDB kemudian memindai nilai atribut kunci urutan hingga menemukan Fido.

Untuk membaca semua item dengan Dog, Anda dapat mengeluarkan Query operasi tanpa menentukan kondisi kunci pengurutan. AnimalType Secara default, item dikembalikan dalam urutan penyimpanannya (yaitu, dalam urutan naik oleh kunci urutan). Atau, Anda dapat meminta urutan menurun.

Untuk mengkueri hanya beberapa item Dog, Anda dapat menerapkan syarat untuk kunci urutan (misalnya, hanya item Dog di mana Name dimulai dengan huruf yang berada dalam rentang A hingga K).

Note

Dalam tabel DynamoDB, tidak ada batasan atas jumlah nilai kunci urutan yang berbeda per nilai kunci partisi. Jika Anda perlu untuk menyimpan jutaan item Dog di tabel Pets, DynamoDB akan mengalokasikan cukup penyimpanan untuk menangani persyaratan ini secara otomatis.

Dari SQL ke NoSQL

Jika Anda seorang developer aplikasi, Anda mungkin memiliki beberapa pengalaman menggunakan sistem manajemen basis data relasional (RDBMS) dan Structured Query Language (SQL). Ketika Anda mulai bekerja dengan Amazon DynamoDB, Anda akan menemukan banyak kesamaan, tetapi juga banyak hal yang berbeda. NoSQL adalah istilah yang digunakan untuk menggambarkan sistem basis data nonrelational yang sangat tersedia, dapat diskalakan, dan dioptimalkan untuk performa tinggi. Alih-alih model relasional, basis data NoSQL (seperti DynamoDB) menggunakan model alternatif untuk manajemen data, seperti pasangan nilai kunci atau penyimpanan dokumen. Untuk informasi selengkapnya, lihat [Apa itu NoSQL?](#).

Amazon DynamoDB mendukung [PartiQL](#), bahasa kueri open-source yang kompatibel dengan SQL yang memudahkan Anda untuk melakukan kueri data secara efisien, terlepas dari di mana atau dalam format apa itu disimpan. Dengan PartiQL, Anda dapat dengan mudah memproses data terstruktur dari basis data relasional, data semi-terstruktur dan bersarang dalam format data terbuka, dan bahkan data tanpa skema di NoSQL atau basis data dokumen yang memungkinkan atribut berbeda untuk baris yang berbeda. Untuk informasi selengkapnya, lihat [bahasa kueri PartiQL](#).

Bagian berikut menjelaskan tugas basis data umum, membandingkan dan mengontraskan pernyataan SQL dengan operasi DynamoDB setara mereka.

Note

Contoh SQL di bagian ini kompatibel dengan MySQL RDBMS.
Contoh DynamoDB di bagian ini menunjukkan nama operasi DynamoDB, bersama dengan parameter untuk operasi dalam format JSON. Untuk contoh kode yang menggunakan operasi ini, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Topik

- [Relasional \(SQL\) atau NoSQL?](#)
- [Karakteristik basis data](#)
- [Membuat tabel](#)
- [Mendapatkan informasi tentang tabel](#)
- [Menulis data ke tabel](#)
- [Perbedaan utama antara SQL dan DynamoDB saat membaca data dari tabel](#)
- [Mengelola indeks](#)
- [Memodifikasi data dalam tabel](#)
- [Menghapus data dari tabel](#)
- [Menghapus tabel](#)

Relasional (SQL) atau NoSQL?

Aplikasi masa kini memiliki persyaratan yang lebih menuntut daripada sebelumnya. Misalnya, game online mungkin dimulai dengan hanya beberapa pengguna dan jumlah data yang sangat kecil. Namun, jika game menjadi sukses, itu dapat dengan mudah melampaui sumber daya dari sistem manajemen basis data yang mendasarinya. Sudah biasa bagi aplikasi berbasis web untuk memiliki ratusan, ribuan, atau jutaan pengguna bersamaan, dengan data baru seukuran terabyte atau lebih yang dihasilkan per hari. Basis data untuk aplikasi tersebut harus menangani puluhan (atau ratusan) dari ribuan baca dan tulis per detik.

Amazon DynamoDB sangat cocok untuk jenis beban kerja tersebut. Sebagai developer, Anda dapat memulai dari kecil dan secara bertahap meningkatkan pemanfaatan Anda seiring aplikasi Anda menjadi lebih populer. DynamoDB menskalakan tanpa hambatan untuk menangani jumlah data yang sangat besar dan jumlah pengguna yang sangat besar.

Untuk informasi lebih lanjut tentang pemodelan basis data relasional tradisional dan cara menyesuaikannya untuk DynamoDB, lihat [Praktik terbaik untuk memodelkan data relasional di DynamoDB](#).

Tabel berikut menunjukkan beberapa perbedaan tingkat tinggi antara sistem manajemen basis data relasional (RDBMS) dan DynamoDB.

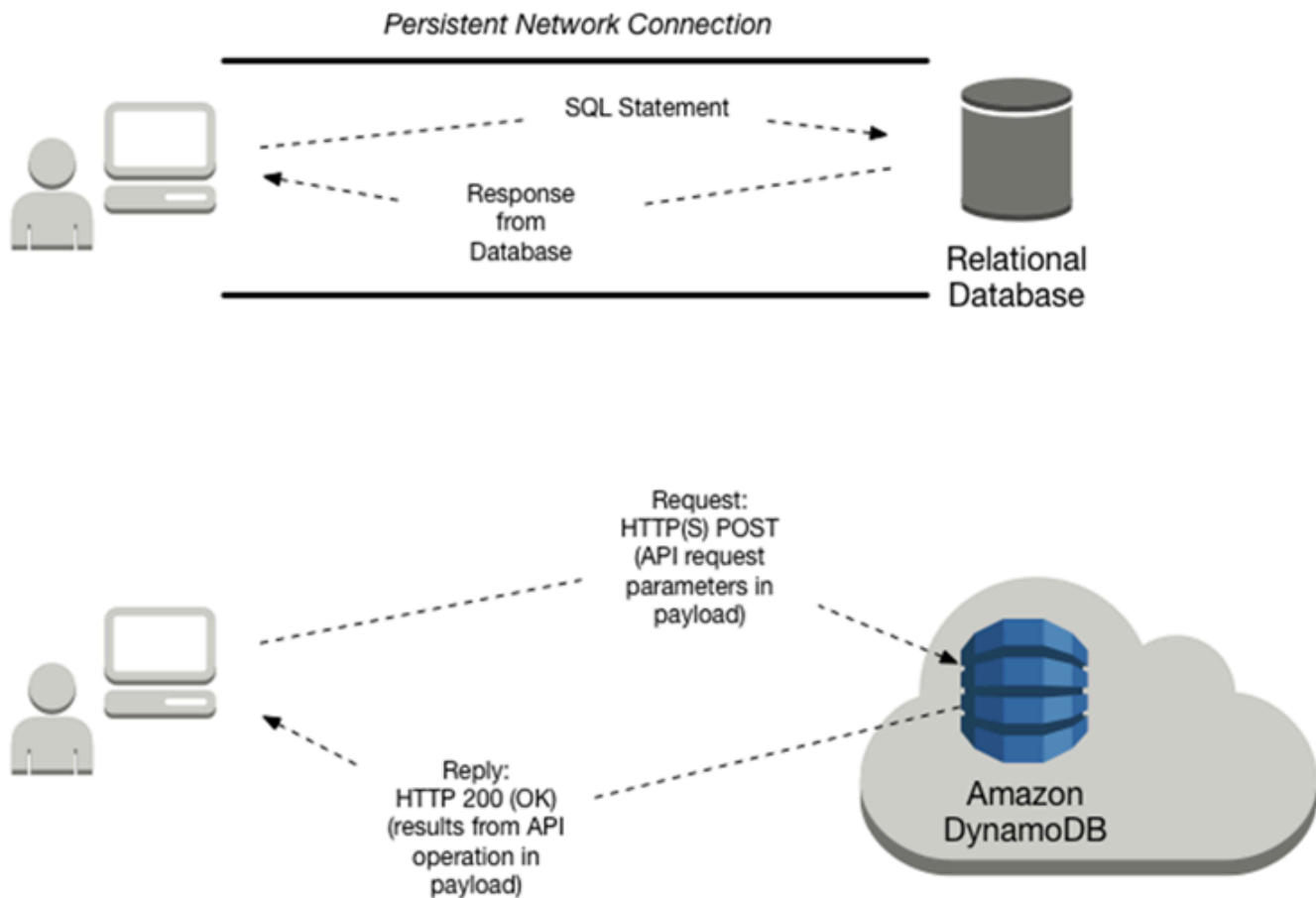
Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
Beban Kerja Optimal	Kueri ad hoc; pergudangan data; OLAP (pemrosesan analitis online).	Aplikasi berskala web, termasuk jejaring sosial, game, berbagi media, dan Internet untuk Segala (IoT).
Model Data	Model relasional membutuhkan skema yang didefinisikan dengan baik, di mana data dinormalisasi ke dalam tabel, baris, dan kolom. Selain itu, semua hubungan didefinisikan antara tabel, kolom, indeks, dan elemen basis data lainnya.	DynamoDB tidak memiliki skema. Setiap tabel harus memiliki kunci primer untuk secara unik mengidentifikasi setiap item data, tetapi tidak ada kendala yang sama pada atribut non-kunci lainnya. DynamoDB dapat mengelola data terstruktur atau semi terstruktur, termasuk dokumen JSON.
Akses Data	SQL adalah standar untuk menyimpan dan mengambil data. Basis data relasional menawarkan seperangkat alat untuk menyederhanakan pengembangan aplikasi yang didorong basis data, tetapi semua alat ini menggunakan SQL.	Anda dapat menggunakan AWS Management Console, the AWS CLI, atau WorkBench NoSQL untuk bekerja dengan DynamoDB dan melakukan tugas ad hoc. PartiQL , bahasa kueri yang kompatibel dengan SQL, memungkinkan Anda memilih, menyisipkan, memperbarui, dan menghapus data di DynamoDB. Aplikasi dapat menggunakan AWS perangkat pengembangan perangkat lunak (SDK) untuk bekerja dengan DynamoDB menggunakan antarmuka

Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
		berbasis objek, berpusat pada dokumen, atau tingkat rendah.
Performa	Basis data relasional dioptimalkan untuk penyimpanan, sehingga performa umumnya tergantung pada subsistem disk. Developer dan administrator basis data harus mengoptimalkan permintaan, indeks, dan struktur tabel untuk mencapai puncak performa.	DynamoDB dioptimalkan untuk melakukan komputasi, sehingga performa utamanya adalah fungsi dari hardware yang mendasari dan latensi jaringan. Sebagai layanan terkelola, DynamoDB mengisolasi Anda dan aplikasi Anda dari detail implementasi ini, sehingga Anda dapat fokus pada merancang dan membangun aplikasi yang kuat dan berperforma tinggi.
Penskalaan	Paling mudah untuk menaikkan skala dengan perangkat keras yang lebih cepat. Mungkin juga bagi tabel basis data untuk menjangkau beberapa host dalam sistem terdistribusi, tetapi ini memerlukan investasi tambahan. basis data relasional memiliki ukuran maksimum untuk jumlah dan ukuran file, yang membebani batas atas pada skalabilitas.	DynamoDB dirancang untuk diperluas menggunakan kluster perangkat keras terdistribusi. Desain ini memungkinkan peningkatan throughput tanpa peningkatan latensi. Pelanggan menentukan persyaratan throughput mereka, dan DynamoDB mengalokasikan sumber daya yang cukup untuk memenuhi persyaratan tersebut. Tidak ada batas atas pada jumlah item per tabel, atau ukuran total tabel tersebut.

Karakteristik basis data

Sebelum aplikasi Anda dapat mengakses basis data, aplikasi harus diautentikasi untuk memastikan bahwa aplikasi diperbolehkan untuk menggunakan basis data. Aplikasi harus diotorisasi sehingga aplikasi hanya dapat melakukan tindakan yang memiliki izin.

Diagram berikut menunjukkan interaksi klien dengan basis data relasional dan dengan Amazon DynamoDB.



Tabel berikut memiliki lebih banyak detail tentang tugas interaksi klien.

Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
Alat untuk Mengakses Basis Data	Kebanyakan basis data relasional menyediakan antarmuka baris perintah	Dalam sebagian besar kasus, Anda menulis kode aplikasi. Anda juga dapat

Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
	<p>(CLI) sehingga Anda dapat memasukkan pernyataan SQL ad hoc dan melihat hasilnya segera.</p>	<p>menggunakan Workbench AWS Management Console, the AWS Command Line Interface (AWS CLI), atau NoSQL untuk mengirim permintaan ad hoc ke DynamoDB dan melihat hasilnya. PartiQL , bahasa kueri yang kompatibel dengan SQL, memungkinkan Anda memilih, menyisipkan, memperbarui, dan menghapus data di DynamoDB.</p>
Menghubungkan ke Basis Data	<p>Program aplikasi menetapkan dan memelihara koneksi jaringan dengan basis data. Ketika aplikasi selesai, aplikasi mengakhiri koneksi.</p>	<p>DynamoDB adalah layanan web, dan interaksi dengannya adalah stateless. Aplikasi tidak perlu mempertahankan koneksi jaringan yang persisten. Sebaliknya, interaksi dengan DynamoDB terjadi menggunakan permintaan dan respons HTTP(S).</p>

Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
Autentikasi	Aplikasi tidak dapat terhubung ke basis data hingga diautentikasi. RDBMS dapat melakukan autentikasi itu sendiri, atau RDBMS dapat mengirimkan tugas ini ke sistem operasi host atau directory service.	Setiap permintaan untuk DynamoDB harus disertai dengan tanda tangan kriptografi, yang mengautentikasi permintaan tertentu. AWS SDK menyediakan semua logika yang diperlukan untuk membuat tanda tangan dan permintaan penandatanganan. Untuk informasi selengkapnya, lihat Menandatangani permintaan AWS API di bagian Referensi Umum AWS.
Otorisasi	Aplikasi hanya dapat melakukan tindakan yang telah mendapatkan otorisasi. Administrator basis data atau pemilik aplikasi dapat menggunakan SQL GRANT dan REVOKE pernyataan untuk mengontrol akses ke objek basis data (misalnya tabel), data (misalnya baris dalam tabel), atau kemampuan untuk menerbitkan pernyataan SQL tertentu.	Di DynamoDB, otorisasi ditangani AWS Identity and Access Management oleh (IAM). Anda dapat menulis kebijakan IAM untuk memberikan izin pada sumber daya DynamoDB (misalnya tabel), dan kemudian mengizinkan pengguna dan peran IAM untuk menggunakan kebijakan tersebut. IAM juga dilengkapi kontrol akses terperinci untuk item data individual dalam tabel DynamoDB. Untuk informasi selengkapnya, lihat Manajemen Identitas dan Akses untuk Amazon DynamoDB .

Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
Mengirim Permintaan	Aplikasi menerbitkan pernyataan SQL untuk setiap operasi basis data yang ingin dilakukan. Sebelum penerimaan pernyataan SQL, RDBMS memeriksa sintaks, membuat rencana untuk melakukan operasi, kemudian menjalankan rencana.	Aplikasi mengirimkan permintaan HTTP(S) untuk DynamoDB. Permintaan berisi nama operasi DynamoDB untuk dilakukan, bersama dengan parameter. DynamoDB menjalankan permintaan segera.
Menerima Respons	RDBMS mengembalikan hasil dari pernyataan SQL. Jika ada kesalahan, RDBMS mengembalikan status dan pesan kesalahan.	DynamoDB mengembalikan respons HTTP(S) yang berisi hasil operasi. Jika ada kesalahan, DynamoDB mengembalikan status dan pesan kesalahan HTTP.

Membuat tabel

Tabel adalah struktur data mendasar dalam basis data relasional dan di Amazon DynamoDB. Sistem manajemen basis data relasional (RDBMS) mengharuskan Anda untuk menentukan skema tabel ketika Anda membuatnya. Sebaliknya, tabel DynamoDB tidak memiliki skema-selain kunci primer, Anda tidak perlu mendefinisikan atribut tambahan atau tipe data ketika Anda membuat tabel.

Bagian berikut membandingkan bagaimana Anda akan membuat tabel dengan SQL dengan bagaimana Anda akan membuatnya dengan DynamoDB.

Topik

- [Membuat tabel dengan SQL](#)
- [Membuat tabel dengan DynamoDB](#)

Membuat tabel dengan SQL

Dengan SQL Anda akan menggunakan pernyataan `CREATE TABLE` untuk membuat tabel, seperti yang ditunjukkan pada contoh berikut.

```
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)  
);
```

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Anda harus menentukan semua kolom tabel dan tipe data, serta kunci primer tabel. (Anda dapat menggunakan pernyataan `ALTER TABLE` untuk mengubah definisi ini nanti, jika perlu.)

Banyak implementasi SQL memungkinkan Anda menentukan spesifikasi penyimpanan untuk tabel Anda, sebagai bagian dari pernyataan `CREATE TABLE`. Kecuali jika Anda mengindikasikan sebaliknya, tabel dibuat dengan pengaturan penyimpanan default. Dalam lingkungan produksi, administrator basis data dapat membantu menentukan parameter penyimpanan optimal.

Membuat tabel dengan DynamoDB

Gunakan operasi `CreateTable` untuk membuat tabel mode yang ditetapkan, menentukan parameter seperti yang ditunjukkan berikut:

```
{  
    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
  }
}
```

Kunci utama untuk tabel ini terdiri dari Artist (partition key) dan SongTitle(sort key).

Anda harus memberikan parameter berikut untuk CreateTable:

- **TableName** — Nama tabel.
- **KeySchema** — Atribut yang digunakan untuk kunci primer. Untuk informasi selengkapnya, lihat [Tabel, item, dan atribut](#) dan [Kunci primer](#).
- **AttributeDefinitions** — Tipe data untuk atribut skema kunci.
- **ProvisionedThroughput (for provisioned tables)** — Jumlah baca dan tulis per detik yang Anda butuhkan untuk tabel ini. DynamoDB mencadangkan cukup penyimpanan dan sumber daya sistem sehingga persyaratan throughput Anda selalu dipenuhi. Anda dapat menggunakan operasi UpdateTable untuk mengubah ini nanti, jika perlu. Anda tidak perlu menentukan persyaratan penyimpanan tabel karena alokasi penyimpanan dikelola sepenuhnya oleh DynamoDB.

Note

Untuk contoh kode yang menggunakan CreateTable, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Mendapatkan informasi tentang tabel

Anda dapat memverifikasi bahwa tabel telah dibuat sesuai dengan spesifikasi Anda. Dalam basis data relasional, semua skema tabel ditampilkan. Tabel Amazon DynamoDB tidak memiliki skema, sehingga hanya atribut kunci primer yang ditampilkan.

Topik

- [Mendapatkan informasi tentang tabel dengan SQL](#)
- [Dapatkan informasi tentang tabel DynamoDB](#)

Mendapatkan informasi tentang tabel dengan SQL

Sebagian besar sistem manajemen basis data relasional (RDBMS) memungkinkan Anda untuk menggambarkan struktur tabel - kolom, tipe data, definisi kunci primer, dan sebagainya. Tidak ada cara standar untuk melakukan hal ini di SQL. Namun, banyak sistem basis data menyediakan perintah DESCRIBE. Berikut adalah contoh dari MySQL.

```
DESCRIBE Music;
```

Ini mengembalikan struktur tabel Anda, dengan semua nama kolom, tipe data, dan ukuran.

```
+-----+-----+-----+-----+-----+-----+
| 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    |      |
+-----+-----+-----+-----+-----+-----+
```

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Dapatkan informasi tentang tabel DynamoDB

DynamoDB memiliki operasi, `DescribeTable` yang serupa. Satu-satunya parameter adalah nama tabel.

```
{
  TableName : "Music"
}
```

Balasan dari DescribeTable terlihat seperti berikut ini.

```
{
  "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 juga mengembalikan informasi tentang indeks pada tabel, pengaturan throughput yang ditetapkan, jumlah item perkiraan, dan metadata lainnya.

Menulis data ke tabel

Tabel basis data relasional berisi barisan data. Baris terdiri dari kolom. Tabel Amazon DynamoDB berisi item. Item terdiri dari atribut.

Bagian ini menjelaskan cara menulis satu baris (atau item) ke tabel.

Topik

- [Menulis data ke tabel dengan SQL](#)
- [Menulis data ke tabel di DynamoDB](#)

Menulis data ke tabel dengan SQL

Sebuah tabel dalam basis data relasional adalah struktur data dua dimensi yang terdiri dari baris dan kolom. Beberapa sistem manajemen basis data juga menyediakan dukungan untuk data semi terstruktur, biasanya dengan JSON asli atau tipe data XML. Namun, detail implementasi bervariasi antara vendor.

Dalam SQL, Anda akan menggunakan pernyataan INSERT untuk menambahkan baris ke tabel.

```
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}'
);
```

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle. Anda harus menentukan nilai untuk kolom ini.

Note

Contoh ini menggunakan kolom Tag untuk menyimpan data semi terstruktur tentang lagu di tabel Music. Kolom Tag didefinisikan sebagai tipe TEXT, yang dapat menyimpan hingga 65.535 karakter di MySQL.

Menulis data ke tabel di DynamoDB

Di Amazon DynamoDB, Anda dapat menggunakan DynamoDB API atau [PartiQL](#) (bahasa kueri yang kompatibel dengan SQL) untuk menambahkan item ke tabel.

DynamoDB API


Dengan DynamoDB API, Anda menggunakan operasi `PutItem` untuk menambahkan item ke tabel.

```
{
  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
    }
  }
}
```

Kunci utama untuk tabel ini terdiri dari `Artist` dan `SongTitle`. Anda harus menentukan nilai untuk atribut ini.

Berikut adalah beberapa hal penting untuk diketahui tentang contoh `PutItem`:

- DynamoDB menyediakan dukungan asli untuk dokumen, menggunakan JSON. Hal ini membuat DynamoDB ideal untuk menyimpan data semi terstruktur, seperti Tag. Anda juga dapat mengambil dan memanipulasi data dari dalam dokumen JSON.
- Tabel Musik tidak memiliki atribut yang telah ditentukan sebelumnya, selain kunci utama (`Artist` dan `SongTitle`).
- Sebagian besar basis data SQL berorientasi transaksi. Ketika Anda menerbitkan pernyataan `INSERT`, modifikasi data tidak permanen sampai Anda menerbitkan pernyataan `COMMIT`. Dengan Amazon DynamoDB, efek operasi `PutItem` bersifat permanen ketika DynamoDB membalas dengan kode status HTTP 200 (OK).

 Note

Untuk contoh kode menggunakan PutItem, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Berikut adalah beberapa contoh PutItem lainnya.

```
{
  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

Selain `PutItem`, DynamoDB mendukung operasi `BatchWriteItem` untuk menulis beberapa item pada saat yang sama.

PartiQL for DynamoDB


Dengan PartiQL, Anda menggunakan operasi `ExecuteStatement` untuk menambahkan item ke tabel, menggunakan pernyataan `Insert PartiQL`.

```

INSERT into Music value {
  'Artist': 'No One You Know',
  'SongTitle': 'Call Me Today',
  'AlbumTitle': 'Somewhat Famous',
  'Year' : '2015',
  'Genre' : 'Acme'
}

```

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle. Anda harus menentukan nilai untuk atribut ini.

 Note


Untuk contoh kode menggunakan Insert dan ExecuteStatement, lihat [Pernyataan sisipkan PartiQL untuk DynamoDB](#).

Perbedaan utama antara SQL dan DynamoDB saat membaca data dari tabel

Dengan SQL, Anda menggunakan pernyataan SELECT untuk mengambil satu baris atau lebih dari tabel. Anda menggunakan klausul WHERE untuk menentukan data yang dikembalikan kepada Anda.

Ini berbeda dengan menggunakan Amazon DynamoDB yang menyediakan operasi berikut untuk membaca data:

- ExecuteStatement mengambil satu atau beberapa item dari tabel. BatchExecuteStatement mengambil beberapa item dari tabel yang berbeda dalam satu operasi. Kedua operasi ini menggunakan [PartiQL](#), bahasa kueri yang kompatibel dengan SQL.
- GetItem – Mengambil satu item dari tabel. Ini adalah cara yang paling efisien untuk membaca satu item karena ini menyediakan akses langsung ke lokasi fisik item. (DynamoDB juga menyediakan operasi BatchGetItem, memungkinkan Anda untuk melakukan hingga 100 panggilan GetItem dalam satu operasi.)
- Query – Mengambil semua item yang memiliki kunci partisi tertentu. Dalam item tersebut, Anda dapat menerapkan syarat untuk kunci urutan dan mengambil hanya subset dari data. Query menyediakan akses cepat dan efisien ke partisi tempat data disimpan. (Untuk informasi selengkapnya, lihat [Partisi dan distribusi data](#).)
- Scan – Mengambil semua item dalam tabel yang ditentukan. (Operasi ini tidak boleh digunakan dengan tabel besar karena dapat mengkonsumsi sejumlah besar sumber daya sistem.)

 Note

Dengan basis data relasional, Anda dapat menggunakan pernyataan SELECT untuk menggabungkan data dari beberapa tabel dan mengembalikan hasil. Gabungan sangat

penting untuk model relasional. Untuk memastikan bahwa gabungan berjalan efisien, basis data dan aplikasi harus disetel performanya secara berkelanjutan. DynamoDB adalah basis data NoSQL non-relasional yang tidak mendukung gabungan tabel. Sebaliknya, aplikasi membaca data dari satu tabel pada satu waktu.

Bagian berikut menjelaskan kasus penggunaan yang berbeda untuk membaca data, dan cara melakukan tugas-tugas ini dengan basis data relasional dan dengan DynamoDB.

Topik

- [Membaca item menggunakan kunci primernya](#)
- [Mengkueri tabel](#)
- [Memindai tabel](#)

Membaca item menggunakan kunci primernya

Salah satu pola akses umum untuk basis data adalah untuk membaca satu item dari tabel. Anda harus menentukan kunci primer dari item yang Anda inginkan.

Topik

- [Membaca item menggunakan kunci primernya dengan SQL](#)
- [Membaca item menggunakan kunci primernya di DynamoDB](#)

Membaca item menggunakan kunci primernya dengan SQL

Dalam SQL, Anda akan menggunakan pernyataan SELECT untuk mengambil data dari tabel. Anda dapat meminta satu kolom atau lebih dalam hasil (atau semuanya, jika Anda menggunakan operator *). Klausa WHERE menentukan baris mana yang dikembalikan.

Berikut ini adalah pernyataan SELECT untuk mengambil satu baris dari tabel Music. Klausa WHERE menentukan nilai-nilai kunci primer.

```
SELECT *
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Anda dapat memodifikasi kueri ini untuk mengambil hanya subset dari kolom.


```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Perhatikan bahwa kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Membaca item menggunakan kunci primernya di DynamoDB

Di Amazon DynamoDB, Anda dapat menggunakan DynamoDB API atau [PartiQL](#) (bahasa kueri yang kompatibel dengan SQL) untuk membaca item dari tabel.

DynamoDB API

Dengan DynamoDB API, Anda menggunakan operasi `PutItem` untuk menambahkan item ke tabel.

DynamoDB menyediakan operasi `GetItem` untuk mengambil item dengan kunci primernya. `GetItem` sangat efisien karena menyediakan akses langsung ke lokasi fisik item. (Untuk informasi selengkapnya, lihat [Partisi dan distribusi data](#).)

Secara default, `GetItem` mengembalikan seluruh item dengan semua atribut.

```
{
  TableName: "Music",
  Key: {
    "Artist": "No One You Know",
    "SongTitle": "Call Me Today"
  }
}
```

Anda dapat menambahkan parameter `ProjectionExpression` untuk mengembalikan hanya beberapa atribut.

```
{
  TableName: "Music",
  Key: {
    "Artist": "No One You Know",
    "SongTitle": "Call Me Today"
  },
  "ProjectionExpression": "AlbumTitle, Year, Price"
```

```
}
```

Perhatikan bahwa kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Operasi `GetItem` DynamoDB sangat efisien. Ini menggunakan nilai-nilai kunci primer untuk menentukan lokasi penyimpanan yang tepat dari item yang bersangkutan, dan mengambilnya langsung dari sana. Pernyataan `SELECT SQL` sama efisiennya, dalam hal mengambil item dengan nilai-nilai kunci primer.

Pernyataan `SELECT` mendukung berbagai jenis tabel kueri dan pindai. DynamoDB menyediakan fungsionalitas serupa dengan operasi `Query` dan `Scan`, yang dijelaskan dalam [Mengkueri tabel](#) dan [Memindai tabel](#).

Pernyataan `SELECT SQL` dapat melakukan gabungan tabel, memungkinkan Anda untuk mengambil data dari beberapa tabel pada waktu yang sama. Gabungan paling efektif di mana tabel basis data dinormalisasi dan hubungan antara tabel jelas. Namun, jika Anda menggabungkan terlalu banyak tabel dalam satu performa aplikasi pernyataan `SELECT` dapat terpengaruh. Anda dapat mengatasi masalah tersebut dengan menggunakan replikasi basis data, tampilan terwujud, atau penulisan ulang kueri.

DynamoDB adalah basis data non-relasional dan tidak mendukung gabungan tabel. Jika Anda memigrasikan aplikasi yang ada dari basis data relasional ke DynamoDB, Anda perlu mendenormalisasi model data Anda untuk menghapus kebutuhan gabungan.

Note

Untuk contoh kode yang menggunakan `GetItem`, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

PartiQL for DynamoDB

Dengan PartiQL, Anda menggunakan operasi `ExecuteStatement` untuk membaca item dari tabel, menggunakan pernyataan `Select PartiQL`.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Perhatikan bahwa kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Note

Pernyataan pilih PartiQL dapat digunakan juga pada tabel Kueri atau Pindai DynamoDB

Untuk contoh kode menggunakan `Select` dan `ExecuteStatement`, lihat [Pernyataan pemilihan PartiQL untuk DynamoDB](#).

Mengkueri tabel

Pola akses umum lainnya adalah membaca beberapa item dari tabel, berdasarkan kriteria kueri Anda.

Topik

- [Mengkueri tabel dengan SQL](#)
- [Mengkueri tabel di DynamoDB](#)

Mengkueri tabel dengan SQL

Saat menggunakan SQL, pernyataan `SELECT` memungkinkan Anda melakukan kueri pada kolom kunci, kolom non-kunci, atau kombinasi apa pun. Klausula `WHERE` menentukan baris mana yang dikembalikan, seperti yang ditunjukkan dalam contoh berikut.

```
/* 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;
```

Perhatikan bahwa kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Mengkueri tabel di DynamoDB

Di Amazon DynamoDB, Anda dapat menggunakan DynamoDB API atau [PartiQL](#) (bahasa kueri yang kompatibel dengan SQL) untuk menanyakan item dari tabel.

DynamoDB API

Dengan Amazon DynamoDB, Anda dapat menggunakan operasi Query untuk mengambil data dengan cara yang serupa. Operasi Query ini menyediakan akses cepat dan efisien ke lokasi fisik di mana data disimpan. Untuk informasi selengkapnya, lihat [Partisi dan distribusi data](#).

Anda dapat menggunakan Query dengan tabel atau indeks sekunder apa pun. Anda harus menentukan syarat kesetaraan untuk nilai kunci partisi, dan secara opsional Anda dapat menyediakan syarat lain untuk atribut kunci urutan jika ditentukan.

Parameter KeyConditionExpression menentukan nilai kunci yang ingin Anda kueri. Anda dapat menggunakan opsi FilterExpression untuk menghapus item tertentu dari hasil sebelum item dikembalikan kepada Anda.

Di DynamoDB, Anda harus menggunakan ExpressionAttributeValue sebagai placeholder dalam parameter ekspresi (seperti KeyConditionExpression dan FilterExpression). Hal ini analog dengan penggunaan variabel pengikatan dalam basis data relasional, di mana Anda mengganti nilai-nilai aktual ke pernyataan SELECT pada saat runtime.

Perhatikan bahwa kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Berikut adalah beberapa contoh Query DynamoDB.

```
// 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"
    }
}
```

Note

Untuk contoh kode yang menggunakan Query, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

PartiQL for DynamoDB

Dengan PartiQL, Anda dapat melakukan kueri dengan menggunakan operasi `ExecuteStatement` dan pernyataan `Select` pada kunci partisi.

```
SELECT AlbumTitle, Year, Price
```

```
FROM Music
WHERE Artist='No One You Know'
```

Menggunakan pernyataan SELECT dengan cara ini mengembalikan semua lagu yang terkait dengan Artist tertentu.

Untuk contoh kode menggunakan Select dan ExecuteStatement, lihat [Pernyataan pemilihan PartiQL untuk DynamoDB](#).

Memindai tabel

Dalam SQL, pernyataan SELECT tanpa sebuah klausul WHERE akan mengembalikan setiap baris dalam sebuah tabel. Di Amazon DynamoDB, operasi Scan melakukan hal yang sama. Dalam kedua kasus, Anda dapat mengambil semua item atau hanya beberapa item.

Entah Anda menggunakan basis data SQL atau NoSQL, pindaian tidak boleh sering digunakan karena pindaian dapat mengonsumsi sejumlah besar sumber daya sistem. Terkadang pemindaian layak digunakan (seperti memindai tabel kecil) atau tidak dapat dihindari (seperti melakukan ekspor data secara massal). Namun, sebagai aturan umum, Anda harus merancang aplikasi Anda agar tidak melakukan pemindaian. Untuk informasi selengkapnya, lihat [Operasi kueri di DynamoDB](#).

Note

Melakukan ekspor massal juga membuat setidaknya 1 file per partisi. Semua item di setiap file berasal dari keyspace hash partisi tertentu.

Topik

- [Memindai tabel dengan SQL](#)
- [Memindai tabel di DynamoDB](#)

Memindai tabel dengan SQL

Saat menggunakan SQL Anda dapat memindai tabel dan mengambil semua data dengan menggunakan pernyataan SELECT tanpa menentukan klausul WHERE. Anda dapat meminta satu kolom atau lebih dalam hasil. Atau, Anda dapat meminta semuanya jika Anda menggunakan karakter wildcard (*).

Berikut ini adalah contoh-contoh penggunaan pernyataan SELECT.

```
/* 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;
```

Memindai tabel di DynamoDB

Di Amazon DynamoDB, Anda dapat menggunakan DynamoDB API atau [PartiQL](#) (bahasa kueri yang kompatibel dengan SQL) untuk melakukan pemindaian pada tabel.

DynamoDB API

Dengan DynamoDB API, Anda menggunakan operasi Scan untuk mengembalikan satu atribut item dan item dengan mengakses setiap item dalam tabel atau indeks sekunder.

```
// Return all of the data in the table  
{  
  TableName: "Music"  
}
```

```
// Return all of the values for Artist and Title  
{  
  TableName: "Music",  
  ProjectionExpression: "Artist, Title"  
}
```

Operasi Scan ini juga menyediakan parameter `FilterExpression`, yang dapat Anda gunakan untuk membuang item yang tidak ingin Anda tampilkan di hasil. Sebuah `FilterExpression` diterapkan setelah pemindaian dilakukan, namun sebelum hasilnya dikembalikan kepada Anda. (Ini tidak dianjurkan dengan tabel besar. Anda masih dikenakan biaya untuk seluruh Scan, bahkan jika hanya beberapa item yang cocok dikembalikan.)

Note

Untuk contoh kode yang menggunakan Scan, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

PartiQL for DynamoDB

Dengan PartiQL, Anda melakukan pemindaian dengan menggunakan operasi `ExecuteStatement` untuk mengembalikan semua konten untuk tabel menggunakan pernyataan `Select`.

```
SELECT AlbumTitle, Year, Price
FROM Music
```

Ingat bahwa pernyataan ini akan mengembalikan semua item untuk dalam tabel `Music`.

Untuk contoh kode menggunakan `Select` dan `ExecuteStatement`, lihat [Pernyataan pemilihan PartiQL untuk DynamoDB](#).

Mengelola indeks

Indeks memberikan Anda akses ke pola kueri alternatif, dan dapat mempercepat kueri. Bagian ini membandingkan dan mengontraskan pembuatan indeks dan penggunaan dalam SQL dan Amazon DynamoDB.

Entah Anda menggunakan basis data relasional atau DynamoDB, Anda harus bijaksana dengan pembuatan indeks. Setiap kali sebuah peristiwa tulis terjadi pada tabel, semua indeks tabel harus diperbarui. Dalam lingkungan yang banyak menulis dengan tabel besar, hal ini dapat menghabiskan banyak sumber daya sistem. Dalam lingkungan `read-only` atau `read-mostly`, hal ini tidak terlalu menjadi perhatian. Namun, Anda harus memastikan bahwa indeks tersebut benar-benar digunakan oleh aplikasi Anda, dan tidak hanya menghabiskan ruang.

Topik

- [Membuat indeks](#)
- [Mengkueri dan memindai indeks](#)

Membuat indeks

Bandingkan pernyataan `CREATE INDEX` di SQL dengan operasi `UpdateTable` di Amazon DynamoDB.

Topik

- [Membuat indeks dengan SQL](#)

- [Membuat indeks di DynamoDB](#)

Membuat indeks dengan SQL

Dalam basis data relasional, indeks adalah struktur data yang memungkinkan Anda melakukan kueri cepat pada kolom yang berbeda dalam sebuah tabel. Anda dapat menggunakan pernyataan SQL `CREATE INDEX` untuk menambahkan indeks ke tabel yang ada, menentukan kolom yang akan diindeks. Setelah indeks dibuat, Anda dapat melakukan kueri data di tabel seperti biasa, tetapi sekarang basis data dapat menggunakan indeks untuk menemukan baris yang ditentukan dengan cepat di tabel alih-alih memindai keseluruhan tabel.

Setelah Anda membuat indeks, basis data mengelolanya untuk Anda. Setiap kali Anda memodifikasi data dalam tabel, indeks secara otomatis dimodifikasi untuk mencerminkan perubahan dalam tabel.

Dalam MySQL, Anda akan membuat indeks seperti berikut.

```
CREATE INDEX GenreAndPriceIndex
ON Music (genre, price);
```

Membuat indeks di DynamoDB

Dalam DynamoDB, Anda dapat membuat dan menggunakan indeks sekunder untuk tujuan yang serupa.

Indeks di DynamoDB berbeda dari rekan-rekan relasional mereka. Ketika Anda membuat indeks sekunder, Anda harus menentukan atribut kunci—kunci partisi dan kunci urutan. Setelah membuat indeks sekunder, Anda dapat melakukan `Query` atau `Scan` padanya seperti yang Anda lakukan dengan tabel. DynamoDB tidak memiliki pengoptimal kueri, sehingga indeks sekunder hanya digunakan ketika Anda melakukan `Query` atau `Scan` padanya.

DynamoDB mendukung dua jenis indeks yang berbeda:

- Indeks sekunder global - Kunci primer dari indeks dapat berupa setiap dua atribut dari tabel.
- Indeks sekunder lokal - Kunci partisi indeks harus sama dengan kunci partisi dari tabel. Namun, kunci urutan dapat berupa atribut lainnya.

DynamoDB memastikan bahwa data dalam indeks sekunder akhirnya konsisten dengan tabel. Anda dapat meminta operasi `Scan` atau `Query` yang sangat konsisten pada tabel atau indeks sekunder lokal. Namun, indeks sekunder global hanya mendukung konsistensi akhir.

Anda dapat menambahkan indeks sekunder global ke tabel yang ada, menggunakan operasi `UpdateTable` dan menentukan `GlobalSecondaryIndexUpdates`.

```
{
  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
          }
        }
      }
    ]
  }
}
```

Anda harus memberikan parameter berikut untuk `UpdateTable`:

- `TableName` – Tabel yang akan dikaitkan dengan indeks.
- `AttributeDefinitions` – Tipe data untuk atribut skema kunci dari indeks.
- `GlobalSecondaryIndexUpdates` – Detail tentang indeks yang ingin Anda buat:
 - `IndexName` – Nama untuk indeks.
 - `KeySchema` – Atribut yang digunakan untuk kunci primer indeks.
 - `Projection` – Atribut dari tabel yang disalin ke indeks. Dalam kasus ini, ALL berarti bahwa semua atribut disalin.

- `ProvisionedThroughput` (for provisioned tables) – Jumlah baca dan tulis per detik yang Anda butuhkan untuk tabel ini. (Ini terpisah dari pengaturan throughput yang ditetapkan pada tabel.)

Bagian dari operasi ini melibatkan pengisian data dari tabel ke indeks baru. Selama pengisian, tabel tetap tersedia. Namun, indeks tidak siap hingga atribut `Backfilling` miliknya berubah dari `true` ke `false`. Anda dapat menggunakan operasi `DescribeTable` untuk melihat atribut ini.

Note

Untuk contoh kode yang menggunakan `UpdateTable`, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Mengkueri dan memindai indeks

Bandingkan mengkueri dan memindai indeks menggunakan pernyataan `SELECT` dalam SQL dengan operasi `Query` dan `Scan` di Amazon DynamoDB.

Topik

- [Mengkueri dan memindai indeks dengan SQL](#)
- [Mengkueri dan memindai indeks di DynamoDB](#)

Mengkueri dan memindai indeks dengan SQL

Dalam basis data relasional, Anda tidak bekerja secara langsung dengan indeks. Sebaliknya, Anda membuat kueri tabel dengan menerbitkan pernyataan `SELECT`, dan pengoptimal kueri dapat menggunakan setiap indeks.

Pengoptimal kueri adalah komponen sistem manajemen basis data relasional (RDBMS) yang mengevaluasi indeks yang tersedia dan menentukan apakah komponen dapat digunakan untuk mempercepat kueri. Jika indeks dapat digunakan untuk mempercepat kueri, RDBMS mengakses indeks pertama kemudian menggunakannya untuk menemukan data dalam tabel.

Berikut adalah beberapa pernyataan SQL yang dapat digunakan `GenreAndPriceIndex` untuk meningkatkan kinerja. Kami berasumsi bahwa tabel `Music` memiliki cukup data di dalamnya hingga pengoptimal kueri memutuskan untuk menggunakan indeks ini, alih-alih memindai seluruh tabel.

```
/* 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;
```

Mengkueri dan memindai indeks di DynamoDB

Dalam DynamoDB, Anda melakukan operasi Query dan Scan secara langsung pada indeks, dengan cara yang sama seperti yang Anda lakukan pada tabel. Anda dapat menggunakan DynamoDB API atau [PartiQL](#) (bahasa kueri yang kompatibel dengan SQL) untuk mengkueri atau memindai indeks. Anda harus menentukan baik TableName dan IndexName.

Berikut ini adalah beberapa pertanyaan GenreAndPriceIndex di DynamoDB. (Skema kunci untuk indeks ini terdiri dari Genre dan Price.)

DynamoDB API

```
// All of the rock songs  
  
{  
  TableName: "Music",  
  IndexName: "GenreAndPriceIndex",  
  KeyConditionExpression: "Genre = :genre",  
  ExpressionAttributeValues: {  
    ":genre": "Rock"  
  },  
};
```

Contoh ini menggunakan ProjectionExpression untuk menunjukkan bahwa Anda hanya ingin beberapa atribut, bukan semuanya, untuk muncul dalam hasil.

```
// 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"
};
```

Berikut ini adalah pemindaian GenreAndPriceIndex.

```
// Return all of the data in the index

{
  TableName: "Music",
  IndexName: "GenreAndPriceIndex"
}
```

PartiQL for DynamoDB

Dengan PartiQL, Anda menggunakan pernyataan PartiQL Select untuk melakukan kueri dan pemindaian pada indeks.

```
// 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
```

Berikut ini adalah pemindaian GenreAndPriceIndex.

```
// Return all of the data in the index

SELECT *
FROM Music.GenreAndPriceIndex
```

Note

Untuk contoh kode menggunakan `Select`, lihat [Pernyataan pemilihan PartiQL untuk DynamoDB](#).

Memodifikasi data dalam tabel

Bahasa SQL menyediakan pernyataan `UPDATE` untuk memodifikasi data. Amazon DynamoDB menggunakan operasi `UpdateItem` untuk menyelesaikan tugas yang serupa.

Topik

- [Memodifikasi data dalam tabel dengan SQL](#)
- [Memodifikasi data dalam tabel di DynamoDB](#)

Memodifikasi data dalam tabel dengan SQL

Dalam SQL, Anda akan menggunakan pernyataan `UPDATE` untuk memodifikasi satu baris atau lebih. Klausula `SET` menentukan nilai-nilai baru untuk satu kolom atau lebih, dan klausul `WHERE` menentukan baris yang dimodifikasi. Berikut adalah contohnya.

```
UPDATE Music
SET RecordLabel = 'Global Records'
WHERE Artist = 'No One You Know' AND SongTitle = 'Call Me Today';
```

Jika tidak ada baris yang cocok dengan klausul `WHERE`, pernyataan `UPDATE` tidak berpengaruh.

Memodifikasi data dalam tabel di DynamoDB

Di DynamoDB, Anda dapat menggunakan DynamoDB API atau [PartiQL](#) (bahasa kueri yang kompatibel dengan SQL) untuk modifikasi satu item. Jika Anda ingin memodifikasi beberapa item, Anda harus menggunakan beberapa operasi.

DynamoDB API

Dengan DynamoDB API, Anda menggunakan operasi `UpdateItem` untuk memodifikasi satu item.

```
{
```

```

    TableName: "Music",
    Key: {
      "Artist": "No One You Know",
      "SongTitle": "Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ExpressionAttributeValues: {
      ":label": "Global Records"
    }
  }
}

```

Anda harus menentukan atribut Key item yang akan dimodifikasi dan UpdateExpression untuk menentukan nilai atribut. UpdateItem berperilaku seperti operasi “upsert”. Item diperbarui jika ada di tabel, tetapi jika tidak, item baru ditambahkan (disisipkan).

UpdateItem mendukung tulis bersyarat, di mana operasi hanya berhasil jika ConditionExpression tertentu bernilai true. Misalnya, operasi UpdateItem berikut tidak melakukan pembaruan kecuali jika harga lagu lebih besar dari atau sama dengan 2,00.

```

{
  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
  }
}

```

UpdateItem juga mendukung penghitung atom, atau atribut tipe Number yang dapat ditambahkan atau dikurangi. Penghitung atom serupa dalam banyak hal dengan generator urutan, kolom identitas, atau bidang penambahan otomatis dalam basis data SQL.

Berikut ini adalah contoh operasi UpdateItem untuk menginisialisasi atribut baru (Play) untuk melacak berapa kali lagu telah diputar.

```

{

```

```
    TableName: "Music",
    Key: {
      "Artist": "No One You Know",
      "SongTitle": "Call Me Today"
    },
    UpdateExpression: "SET Plays = :val",
    ExpressionAttributeValues: {
      ":val": 0
    },
    ReturnValues: "UPDATED_NEW"
  }
```

Parameter `ReturnValues` diatur ke `UPDATED_NEW`, yang mengembalikan nilai-nilai baru dari setiap atribut yang diperbarui. Dalam hal ini, parameter mengembalikan 0 (nol).

Setiap kali seseorang memutar lagu ini, kita dapat menggunakan operasi `UpdateItem` berikut untuk menambahkan Play sebanyak satu.

```
{
  TableName: "Music",
  Key: {
    "Artist": "No One You Know",
    "SongTitle": "Call Me Today"
  },
  UpdateExpression: "SET Plays = Plays + :incr",
  ExpressionAttributeValues: {
    ":incr": 1
  },
  ReturnValues: "UPDATED_NEW"
}
```

Note

Untuk contoh kode yang menggunakan `UpdateItem`, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

PartiQL for DynamoDB

Dengan PartiQL, Anda menggunakan operasi `ExecuteStatement` untuk memodifikasi item pada tabel, menggunakan pernyataan `Update PartiQL`.

Kunci utama untuk tabel ini terdiri dari `Artist` dan `SongTitle`. Anda harus menentukan nilai untuk atribut ini.

```
UPDATE Music
SET RecordLabel = 'Global Records'
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Anda juga dapat memodifikasi beberapa bidang sekaligus, seperti pada contoh berikut.

```
UPDATE Music
SET RecordLabel = 'Global Records'
SET AwardsWon = 10
WHERE Artist = 'No One You Know' AND SongTitle='Call Me Today'
```

Update juga mendukung penghitung atom, atau atribut tipe `Number` yang dapat ditambahkan atau dikurangi. Penghitung atom serupa dalam banyak hal dengan generator urutan, kolom identitas, atau bidang penambahan otomatis dalam basis data SQL.

Berikut ini adalah contoh pernyataan `Update` untuk menginisialisasi atribut baru (`Play`) untuk melacak berapa kali lagu telah diputar.

```
UPDATE Music
SET Plays = 0
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Setiap kali seseorang memutar lagu ini, kita dapat menggunakan pernyataan `Update` berikut untuk menambahkan `Play` sebanyak satu.

```
UPDATE Music
SET Plays = Plays + 1
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Note

Untuk contoh kode menggunakan `Update` dan `ExecuteStatement`, lihat [Pernyataan pembaruan PartiQL untuk DynamoDB](#).

Menghapus data dari tabel

Di SQL, pernyataan DELETE menghapus satu baris atau lebih dari tabel. Amazon DynamoDB menggunakan operasi DeleteItem untuk menghapus satu item pada satu waktu.

Topik

- [Menghapus data dari tabel dengan SQL](#)
- [Menghapus data dari tabel di DynamoDB](#)

Menghapus data dari tabel dengan SQL

Dalam SQL, Anda menggunakan pernyataan DELETE untuk menghapus satu baris atau lebih. Klausula WHERE menentukan baris yang ingin Anda modifikasi. Berikut adalah contohnya.

```
DELETE FROM Music
WHERE Artist = 'The Acme Band' AND SongTitle = 'Look Out, World';
```

Anda dapat memodifikasi klausul WHERE untuk menghapus beberapa baris. Misalnya, Anda dapat menghapus semua lagu dari artis tertentu, seperti yang ditunjukkan dalam contoh berikut.

```
DELETE FROM Music WHERE Artist = 'The Acme Band'
```

Note

Jika Anda menghilangkan klausul WHERE, basis data mencoba untuk menghapus semua baris dari tabel.

Menghapus data dari tabel di DynamoDB

Di DynamoDB, Anda dapat menggunakan DynamoDB API [atau](#) PartiQL (bahasa kueri yang kompatibel dengan SQL) untuk menghapus satu item. Jika Anda ingin memodifikasi beberapa item, Anda harus menggunakan beberapa operasi.

DynamoDB API

Dengan DynamoDB API, Anda menggunakan operasi DeleteItem untuk menghapus data dari tabel, satu item pada satu waktu. Anda harus menentukan nilai kunci primer item.

```
{
  TableName: "Music",
  Key: {
    Artist: "The Acme Band",
    SongTitle: "Look Out, World"
  }
}
```

Note

Selain `deleteItem`, Amazon DynamoDB mendukung operasi `batchWriteItem` untuk menghapus beberapa item pada saat yang sama.

`DeleteItem` mendukung tulis bersyarat, di mana operasi hanya berhasil jika `ConditionExpression` tertentu bernilai `true`. Misalnya, `DeleteItem` operasi berikut menghapus item hanya jika memiliki `RecordLabel` atribut.

```
{
  TableName: "Music",
  Key: {
    Artist: "The Acme Band",
    SongTitle: "Look Out, World"
  },
  ConditionExpression: "attribute_exists(RecordLabel)"
}
```

Note

Untuk contoh kode yang menggunakan `DeleteItem`, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

PartiQL for DynamoDB

Dengan PartiQL, Anda menggunakan pernyataan `Delete` melalui operasi `ExecuteStatement` untuk menghapus data dari tabel, satu item pada satu waktu. Anda harus menentukan nilai kunci primer item.

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle. Anda harus menentukan nilai untuk atribut ini.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks'
```

Anda juga dapat menentukan kondisi tambahan untuk operasi. Operasi DELETE berikut hanya menghapus item jika memiliki lebih dari 11 Penghargaan.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks' AND Awards > 11
```

Note

Untuk contoh kode menggunakan DELETE dan ExecuteStatement, lihat [Pernyataan hapus PartiQL untuk DynamoDB](#).

Menghapus tabel

Dalam SQL, Anda menggunakan pernyataan DROP TABLE untuk menghapus tabel. Di Amazon DynamoDB, Anda menggunakan operasi DeleteTable.

Topik

- [Menghapus tabel dengan SQL](#)
- [Menghapus tabel di DynamoDB](#)

Menghapus tabel dengan SQL

Ketika Anda tidak lagi membutuhkan tabel dan ingin membuangnya secara permanen, Anda akan menggunakan DROP TABLE pernyataan dalam SQL.

```
DROP TABLE Music;
```

Setelah tabel dilepaskan, tabel tidak dapat dipulihkan. (Beberapa basis data relasional memungkinkan Anda untuk membatalkan operasi DROP TABLE, tetapi ini adalah fungsionalitas khusus vendor dan tidak diterapkan secara luas.)

Menghapus tabel di DynamoDB

Pada DynamoDB, `DeleteTable` adalah operasi yang serupa. Pada contoh berikut, tabel dihapus secara permanen.

```
{
  TableName: "Music"
}
```

Note

Untuk contoh kode yang menggunakan `DeleteTable`, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Sumber daya tambahan untuk Amazon DynamoDB

Anda dapat menggunakan sumber daya tambahan berikut untuk memahami dan menggunakan DynamoDB.

Topik

- [Alat untuk pengodean dan visualisasi](#)
- [Artikel Panduan Preskriptif](#)
- [Artikel Pusat Pengetahuan](#)
- [Posting blog, repositori, dan panduan](#)
- [Pemodelan data dan presentasi pola desain](#)
- [Kursus pelatihan](#)

Alat untuk pengodean dan visualisasi

Anda dapat menggunakan alat pengodean dan visualisasi berikut ketika menggunakan DynamoDB:

- [NoSQL Workbench untuk Amazon DynamoDB](#) — Alat visual terpadu yang membantu Anda merancang, membuat, mengirim kueri, dan mengelola tabel DynamoDB. Alat ini menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri.

- [Dynobase](#) – Alat desktop yang memudahkan Anda melihat tabel DynamoDB Anda dan menggunakannya, membuat kode aplikasi, dan mengedit catatan dengan validasi waktu nyata.
- [DynamoDB Toolbox](#) — Sebuah proyek dari Jeremy Daly yang menyediakan utilitas bermanfaat untuk bekerja dengan pemodelan data dan Node.js. andJavaScript
- [DynamoDB Streams Processor](#) – Alat sederhana yang dapat Anda gunakan untuk bekerja dengan [aliran DynamoDB](#).

Artikel Panduan Preskriptif

Panduan Preskriptif AWS menyediakan strategi, panduan, dan pola yang telah teruji waktu untuk membantu mempercepat proyek Anda. Sumber daya ini dikembangkan oleh pakar teknologi AWS dan komunitas Partner AWSglobal, berdasarkan pengalaman bertahun-tahun mereka membantu pelanggan mencapai tujuan bisnis mereka.

Pemodelan dan migrasi data

- [Model data hierarkis di DynamoDB](#)
- [Pemodelan data dengan DynamoDB](#)
- [Memigrasikan basis data Oracle ke DynamoDB menggunakan AWS DMS](#)

Tabel global

- [Menggunakan tabel global Amazon DynamoDB](#)

Nirserver

- [Menerapkan pola saga nirserver dengan AWS Step Functions](#)

Arsitektur SaaS

- [Mengelola penghuni di beberapa produk SaaS pada satu bidang kontrol](#)
- [Orientasi penghuni dalam arsitektur SaaS untuk model silo menggunakan C# dan AWS CDK](#)

Perlindungan data dan pergerakan data

- [Mengonfigurasi akses lintas akun ke Amazon DynamoDB](#)

- [Opsi salinan tabel lengkap untuk DynamoDB](#)
- [Strategi pemulihan bencana untuk basis data di AWS](#)

Lain-lain

- [Membantu menegakkan pemberian tag di DynamoDB](#)

Video panduan preskriptif

- [Menggunakan Arsitektur Nirserver untuk Membuat Alur Data](#)
- [Novartis - Buying Engine: Portal Pengadaan Berkemampuan AI](#)
- [Veritiv: Menghadirkan Wawasan untuk Memprediksi Permintaan Penjualan di Danau Data AWS](#)
- [mimik: Cloud Edge Hibrida Memanfaatkan AWS untuk Mendukung Mesh Layanan Mikro Edge](#)
- [Mengubah Penangkapan Data dengan Amazon DynamoDB](#)

Artikel dan video Panduan Preskriptif lainnya untuk DynamoDB bisa dilihat di [Panduan Preskriptif](#).

Artikel Pusat Pengetahuan

Artikel dan video Pusat Pengetahuan AWS mencakup pertanyaan dan permintaan yang paling sering kami terima dari pelanggan AWS. Berikut ini adalah beberapa artikel Pusat Pengetahuan saat ini tentang tugas-tugas tertentu yang berhubungan dengan DynamoDB:

Optimasi biaya

- [Bagaimana cara mengoptimalkan biaya dengan Amazon DynamoDB?](#)

Throttling dan latensi

- [Mengapa metrik latensi maksimum DynamoDB saya tinggi ketika latensi rata-rata normal?](#)
- [Mengapa tabel DynamoDB saya dikenakan throttling?](#)
- [Mengapa tabel DynamoDB sesuai permintaan saya dikenakan throttling?](#)

Paginasi

- [Bagaimana cara menerapkan paginasi di DynamoDB](#)

Transaksi

- [Mengapa panggilan API TransactWriteItems saya gagal di DynamoDB](#)

Pemecahan Masalah

- [Bagaimana cara mengatasi masalah penskalaan otomatis DynamoDB?](#)
- [Bagaimana cara memecahkan masalah kesalahan HTTP 4XX di DynamoDB](#)

Artikel dan video lainnya untuk DynamoDB bisa dilihat di [artikel Pusat Pengetahuan](#).

Posting blog, repositori, dan panduan

Selain [Panduan Developer DynamoDB](#), ada banyak sumber daya yang berguna terkait bekerja dengan DynamoDB. Berikut adalah beberapa posting blog, repositori, dan panduan pilihan untuk bekerja dengan DynamoDB:

- Repositori AWS berisi [contoh kode DynamoDB](#) dalam berbagai bahasa SDK AWS: [Node.js](#), [Java](#), [Python](#), [.Net](#), [Go](#), dan [Rust](#).
- Buku [DynamoDB — Panduan komprehensif dari DeBrie Alex yang mengajarkan pendekatan berbasis strategi untuk pemodelan data dengan DynamoDB](#).
- Panduan [DynamoDB](#) — Panduan terbuka [dari DeBrie Alex](#) yang berjalan melalui konsep dasar dan fitur-fitur canggih dari database DynamoDB NoSQL.
- [How to switch from RDBMS to DynamoDB in 20 easy steps](#) – Daftar langkah-langkah yang berguna untuk mempelajari pemodelan data dari [Jeremy Daly](#).
- Lembar [contekan JavaScript DocumentClient DynamoDB](#) - Lembar contekan untuk membantu Anda mulai membangun aplikasi dengan DynamoDB di Node.js atau lingkungan. JavaScript
- [DynamoDB Core Concept Videos](#) – Daftar putar ini membahas banyak konsep inti DynamoDB.

Pemodelan data dan presentasi pola desain

Anda dapat menggunakan sumber daya berikut tentang pemodelan data dan pola desain untuk membantu Anda mendapatkan hasil maksimal dari DynamoDB:

- [AWS re:Invent 2019: Data modeling with DynamoDB](#)

- Pembicaraan oleh [Alex DeBrie](#) yang membantu Anda memulai dengan prinsip-prinsip pemodelan data DynamoDB.
- [AWS re:Invent 2020: Data modeling with DynamoDB – Part 1](#)
- [AWS re:Invent 2020: Data modeling with DynamoDB – Part 2](#)
- [AWS re:Invent 2017: Advanced design patterns](#)
- [AWS re:Invent 2018: Advanced design patterns](#)
- [AWS re:Invent 2019: Advanced design patterns](#)
 - Jeremy Daly membagikan [12 poin penting](#) lewat sesi ini.
- [AWS re:Invent 2020: DynamoDB advanced design patterns – Part 1](#)
- [AWS re:Invent 2020: DynamoDB advanced design patterns – Part 2](#)
- [DynamoDB Office Hours di Twitch](#)

Note

Setiap sesi membahas contoh dan kasus penggunaan yang berbeda.

Kursus pelatihan

Ada banyak opsi kursus pelatihan dan edukasi yang berbeda untuk mempelajari DynamoDB lebih lanjut. Berikut ini contoh yang terbaru:

- [Mengembangkan dengan Amazon DynamoDB](#) - Dirancang oleh AWS untuk memberi penjelasan mulai tingkat pemula hingga ahli dalam mengembangkan aplikasi dunia nyata dengan pemodelan data untuk Amazon DynamoDB.
- [DynamoDB deep-dive course](#) – Kursus pelatihan dari A Cloud Guru.
- [Amazon DynamoDB: Building NoSQL database-driven applications](#) – Kursus dari tim Pelatihan dan Sertifikasi AWS yang diselenggarakan di edX.

Menyiapkan DynamoDB

Selain layanan web Amazon DynamoDB AWS , menyediakan versi DynamoDB yang dapat diunduh yang dapat Anda jalankan di komputer Anda. Versi yang dapat diunduh berguna untuk mengembangkan dan menguji kode Anda. Ini memungkinkan Anda menulis dan menguji aplikasi secara lokal tanpa mengakses layanan web DynamoDB.

Topik di bagian ini menjelaskan cara menyiapkan DynamoDB (versi yang dapat diunduh) dan layanan web DynamoDB.

Topik

- [Menyiapkan DynamoDB lokal \(versi yang dapat diunduh\)](#)
- [Menyiapkan DynamoDB \(layanan web\)](#)

Menyiapkan DynamoDB lokal (versi yang dapat diunduh)

Dengan versi Amazon DynamoDB yang dapat diunduh, Anda dapat mengembangkan dan menguji aplikasi tanpa mengakses layanan web DynamoDB. Sebaliknya, basis data bersifat mandiri di komputer Anda. Saat Anda siap untuk menyebarkan aplikasi Anda dalam produksi, Anda menghapus titik akhir lokal dalam kode, dan kemudian menunjuk ke layanan web DynamoDB.

Memiliki versi lokal ini membantu Anda menghemat throughput, penyimpanan data, dan biaya transfer data. Selain itu, Anda tidak memerlukan koneksi internet saat mengembangkan aplikasi.

DynamoDB lokal tersedia sebagai [unduh](#) (memerlukan JRE), sebagai [dependensi Apache Maven](#), atau sebagai [citra Docker](#).

Jika Anda lebih memilih menggunakan layanan web Amazon DynamoDB, lihat [Menyiapkan DynamoDB \(layanan web\)](#) .

Topik

- [Men-deploy DynamoDB secara lokal pada komputer Anda](#)
- [Catatan penggunaan DynamoDB lokal](#)
- [Riwayat rilis DynamoDB lokal](#)
- [Telemetri di DynamoDB lokal](#)

Men-deploy DynamoDB secara lokal pada komputer Anda

Important

Toples lokal DynamoDB dapat diunduh dari tautan distribusi AWS CloudFront kami yang direferensikan di sini. Mulai 1 Januari 2025, bucket distribusi S3 lama tidak akan lagi aktif dan DynamoDB lokal akan didistribusikan melalui tautan distribusi saja. CloudFront

Ada dua versi utama DynamoDB lokal yang tersedia: DynamoDB local v2.x (Current) dan DynamoDB local v1.x (Legacy). Pelanggan harus menggunakan versi 2.x (Current) bila memungkinkan, karena mendukung versi terbaru Java Runtime Environment dan kompatibel dengan namespace jakarta.* untuk proyek Maven. DynamoDB local v1.x akan mencapai akhir dukungan standar mulai 1 Januari 2025. Setelah tanggal ini, v1.x tidak akan lagi menerima pembaruan atau perbaikan bug.

Note

DynamoDB lokal `AWS_ACCESS_KEY_ID` hanya dapat berisi huruf (A–Z, a–z) dan angka (0–9).

Unduh DynamoDB lokal

Ikuti langkah berikut untuk menyiapkan dan menjalankan DynamoDB pada komputer Anda.

Untuk menyiapkan DynamoDB pada komputer Anda

1. Unduh DynamoDB lokal secara gratis dari salah satu lokasi berikut.

Tautan Unduhan	Checksum
.tar.gz .zip	.tar.gz.sha256 .zip.sha256

⚠ Important

Untuk menjalankan DynamoDB v2.3.0 atau lebih tinggi di komputer Anda, Anda harus memiliki Java Runtime Environment (JRE) versi 17.x atau yang lebih baru. Aplikasi tidak berjalan pada versi JRE sebelumnya.

2. Setelah Anda mengunduh arsip, ekstrak isinya dan salin direktori yang diekstrak ke lokasi pilihan Anda.
3. Untuk memulai DynamoDB di komputer Anda, buka jendela prompt perintah, navigasikan ke direktori tempat Anda mengekstrak `DynamoDBLocal.jar`, dan ketik perintah berikut:

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

ℹ Note

Jika Anda menggunakan Windows PowerShell, pastikan untuk melampirkan nama parameter atau seluruh nama dan nilai seperti ini:

```
java -D"java.library.path=./DynamoDBLocal_lib" -jar  
DynamoDBLocal.jar
```

DynamoDB memproses permintaan masuk hingga Anda menghentikannya. Untuk menghentikan DynamoDB, tekan `Ctrl+C` pada prompt perintah.

DynamoDB menggunakan port 8000 secara default. Jika port 8000 tidak tersedia, perintah ini memunculkan pengecualian. Untuk daftar lengkap opsi runtime DynamoDB, termasuk `-port`, masukkan perintah ini.

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

4. Sebelum Anda dapat mengakses DynamoDB secara terprogram atau melalui AWS Command Line Interface (AWS CLI), Anda harus mengonfigurasi kredensial Anda untuk mengaktifkan otorisasi bagi aplikasi Anda. DynamoDB yang dapat diunduh memerlukan kredensial untuk bisa berfungsi, seperti yang ditunjukkan dalam contoh berikut.

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

Anda dapat menggunakan perintah `aws configure` dari AWS CLI untuk menyiapkan kredensial. Untuk informasi selengkapnya, lihat [Menggunakan AWS CLI](#).

5. Mulai menulis aplikasi. Untuk mengakses DynamoDB berjalan secara lokal dengan, gunakan parameter AWS CLI. `--endpoint-url` Misalnya, gunakan perintah berikut untuk membuat daftar tabel DynamoDB.

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

Jalankan DynamoDB lokal sebagai image Docker

Versi Amazon DynamoDB yang dapat diunduh tersedia sebagai citra Docker. Untuk informasi selengkapnya, lihat [dynamodb-local](#). Untuk melihat versi lokal DynamoDB Anda saat ini, masukkan perintah berikut:

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

Untuk contoh penggunaan DynamoDB lokal sebagai bagian dari aplikasi REST yang dibangun di atas AWS Serverless Application Model AWS SAM(), lihat aplikasi [SAM DynamoDB](#) untuk mengelola pesanan. Contoh aplikasi ini menunjukkan cara menggunakan DynamoDB lokal untuk pengujian.

Jika Anda ingin menjalankan aplikasi multi-kontainer yang juga menggunakan kontainer lokal DynamoDB, gunakan Docker Compose untuk menentukan dan menjalankan semua layanan di aplikasi Anda, termasuk DynamoDB lokal.

Untuk menginstal dan menjalankan DynamoDB lokal dengan penulisan Docker:

1. Unduh dan instal [desktop Docker](#).
2. Salin kode berikut ke file dan simpan sebagai `docker-compose.yml`.

```
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
```

Jika Anda ingin aplikasi Anda dan DynamoDB lokal berada dalam kontainer terpisah, gunakan file yml berikut.

```
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
```

Skrip docker-compose.yml ini membuat kontainer app-node dan kontainer dynamodb-local. Skrip menjalankan perintah di kontainer app-node yang menggunakan AWS CLI untuk terhubung ke kontainer dynamodb-local dan menjelaskan batas akun dan tabel.

Untuk digunakan dengan gambar aplikasi Anda sendiri, ganti nilai image pada contoh di bawah dengan nilai gambar aplikasi Anda.

```
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

Skrip YAMB mengharuskan Anda menentukan kunci AWS akses dan kunci AWS rahasia, tetapi skrip tersebut tidak diharuskan menjadi kunci yang valid AWS bagi Anda untuk mengakses DynamoDB lokal.

3. Jalankan perintah baris perintah berikut:

```
docker-compose up
```

Jalankan DynamoDB lokal sebagai dependensi Apache Maven

Ikuti langkah-langkah berikut untuk menggunakan Amazon DynamoDB dalam aplikasi Anda sebagai dependensi.

Untuk men-deploy DynamoDB sebagai repositori Apache Maven

1. Unduh dan instal Apache Maven. Untuk informasi selengkapnya, lihat [Mengunduh Apache Maven](#) dan [Menginstal Apache Maven](#).
2. Tambahkan repositori DynamoDB Maven ke file Project Object Model (POM) aplikasi Anda.

```
<!--Dependency:-->
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>DynamoDBLocal</artifactId>
    <version>2.3.0</version>
  </dependency>
</dependencies>
```

Contoh template untuk digunakan dengan Spring Boot 3 dan/atau Spring Framework 6:

```
<?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>com.amazonaws</groupId>
  <artifactId>DynamoDBLocal</artifactId>
  <version>2.3.0</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

Anda juga dapat menggunakan URL [repositori pusat Maven](#).

[Untuk contoh proyek sampel yang menampilkan beberapa pendekatan untuk menyiapkan dan menggunakan DynamoDB lokal, termasuk mengunduh file JAR, menjalankannya sebagai image Docker, dan menggunakannya sebagai dependensi Maven, lihat DynamoDB Local Sample Java Project.](#)

Catatan penggunaan DynamoDB lokal

Kecuali titik akhir, aplikasi yang berjalan dengan versi Amazon DynamoDB yang dapat diunduh juga harus bekerja dengan layanan web DynamoDB. Namun, saat menggunakan DynamoDB secara lokal, Anda harus mengetahui hal berikut:

- Jika Anda menggunakan `-sharedDb` opsi, DynamoDB membuat file database tunggal bernama `.db.shared-local-instance` Setiap program yang terhubung ke DynamoDB mengakses file ini. Jika Anda menghapus file, Anda kehilangan semua data yang Anda simpan di dalamnya.
- Jika Anda menghilangkan `-sharedDb`, file database bernama `myaccesskeyid_region.db`, dengan ID kunci AWS akses dan AWS Region seperti yang muncul dalam konfigurasi aplikasi Anda. Jika Anda menghapus file, Anda kehilangan semua data yang Anda simpan di dalamnya.
- Jika Anda menggunakan opsi `-inMemory`, DynamoDB tidak menulis file basis data apa pun sama sekali. Sebaliknya, semua data ditulis ke memori, dan data tidak disimpan saat Anda menghentikan DynamoDB.
- Jika Anda menggunakan opsi `-inMemory`, opsi `-sharedDb` juga diperlukan.
- Jika Anda menggunakan opsi `-optimizeDbBeforeStartup`, Anda juga harus menentukan parameter `-dbPath` agar DynamoDB dapat menemukan file basis datanya.
- AWS SDK untuk DynamoDB mengharuskan konfigurasi aplikasi Anda menentukan nilai kunci akses dan nilai Region. AWS Kecuali Anda menggunakan opsi `-sharedDb` atau `-inMemory`, DynamoDB menggunakan nilai-nilai ini untuk memberi nama file basis data lokal. Nilai-nilai ini tidak harus berupa AWS nilai yang valid untuk dijalankan secara lokal. Namun, Anda mungkin merasa nyaman menggunakan nilai yang valid sehingga Anda dapat menjalankan kode di cloud di lain waktu dengan mengubah titik akhir yang Anda gunakan.
- DynamoDB lokal selalu mengembalikan null untuk `billingModeSummary`.

- DynamoDB lokal `AWS_ACCESS_KEY_ID` hanya dapat berisi huruf (A–Z, a–z) dan angka (0–9).

Topik

- [Opsi baris perintah](#)
- [Mengatur Titik Akhir Lokal](#)
- [Perbedaan antara DynamoDB yang dapat diunduh dan layanan web DynamoDB.](#)

Opsi baris perintah

Anda dapat menggunakan opsi baris perintah berikut dengan versi DynamoDB yang dapat diunduh:

- `-corsvalue`— Memungkinkan dukungan untuk berbagi sumber daya lintas asal (CORS) untuk JavaScript. Anda harus memberikan daftar domain tertentu yang "diizinkan" yang dipisahkan koma. Pengaturan default untuk `-cors` adalah tanda bintang (*), yang memungkinkan akses publik.
- `-dbPath value` — Direktori tempat DynamoDB menulis file basis datanya. Jika Anda tidak menentukan opsi ini, file akan ditulis ke direktori saat ini. Anda tidak dapat menentukan `-dbPath` dan `-inMemory` sekaligus.
- `-delayTransientStatuses` — Menyebabkan DynamoDB menimbulkan penundaan untuk operasi tertentu. DynamoDB (versi yang dapat diunduh) dapat melakukan beberapa tugas hampir secara instan, seperti operasi membuat/memperbarui/menghapus pada tabel dan indeks. Namun, layanan DynamoDB memerlukan lebih banyak waktu untuk tugas ini. Menyetel parameter ini membantu DynamoDB yang berjalan di komputer Anda menyimulasikan perilaku layanan web DynamoDB dengan lebih dekat. (Saat ini, parameter ini menyebabkan penundaan hanya untuk indeks sekunder global yang berstatus CREATING atau DELETING.)
- `-help` — Mencetak ringkasan dan opsi penggunaan.
- `-inMemory` — DynamoDB berjalan di memori alih-alih menggunakan file basis data. Saat Anda menghentikan DynamoDB, tidak ada data yang disimpan. Anda tidak dapat menentukan `-dbPath` dan `-inMemory` sekaligus.
- `-optimizeDbBeforeStartup` — Mengoptimalkan tabel basis data yang mendasarinya sebelum memulai DynamoDB di komputer Anda. Anda juga harus menentukan `-dbPath` saat menggunakan parameter ini.
- `-port value` — Nomor port yang digunakan DynamoDB untuk berkomunikasi dengan aplikasi Anda. Jika Anda tidak menentukan opsi ini, port defaultnya adalah 8000.

Note

DynamoDB menggunakan port 8000 secara default. Jika port 8000 tidak tersedia, perintah ini memunculkan pengecualian. Anda dapat menggunakan opsi `-port` untuk menentukan nomor port yang berbeda. Untuk daftar lengkap opsi runtime DynamoDB, termasuk `-port`, masukkan perintah ini.

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

- `-sharedDb` — Jika Anda menentukan `-sharedDb`, DynamoDB menggunakan file basis data tunggal, bukan file terpisah untuk setiap kredensial dan Wilayah.
- `-disableTelemetry` — Ketika ditentukan, DynamoDB lokal tidak akan mengirim telemetry apa pun.
- `-version`— Mencetak versi DynamoDB lokal.

Mengatur Titik Akhir Lokal

Secara default, AWS SDK dan alat menggunakan titik akhir untuk layanan web Amazon DynamoDB. Untuk menggunakan SDK dan alat-alat dengan DynamoDB versi yang dapat diunduh, Anda harus menentukan titik akhir lokal:

```
http://localhost:8000
```

AWS Command Line Interface

Anda dapat menggunakan AWS Command Line Interface (AWS CLI) untuk berinteraksi dengan DynamoDB yang dapat diunduh. Misalnya, Anda dapat menggunakannya untuk melakukan semua langkah-langkah dalam [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Untuk mengakses DynamoDB yang berjalan secara lokal, gunakan parameter `--endpoint-url`. Berikut ini adalah contoh menggunakan daftar tabel di AWS CLI DynamoDB di komputer Anda.

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

Note

Tidak AWS CLI dapat menggunakan versi DynamoDB yang dapat diunduh sebagai titik akhir default. Karena itu, Anda harus menentukan `--endpoint-url` dengan setiap AWS CLI perintah.

AWS SDK

Cara Anda menentukan titik akhir bergantung pada bahasa pemrograman dan AWS SDK yang Anda gunakan. Bagian berikut menjelaskan cara melakukannya:

- [Java: Mengatur AWS Wilayah dan Titik Akhir](#) (DynamoDB lokal mendukung SDK AWS for Java V1 dan V2)
- [.NET: Mengatur AWS Wilayah dan Titik Akhir](#)

Note

Untuk contoh dalam bahasa pemrograman lain, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Perbedaan antara DynamoDB yang dapat diunduh dan layanan web DynamoDB.

Versi DynamoDB yang dapat diunduh ditujukan hanya untuk pengembangan dan pengujian. Sebagai perbandingan, layanan web DynamoDB adalah layanan terkelola dengan skalabilitas, ketersediaan, dan fitur daya tahan yang membuatnya sangat cocok untuk penggunaan produksi.

Versi DynamoDB yang dapat diunduh berbeda dengan layanan web dengan cara berikut:

- Wilayah AWS dan Akun AWS berbeda tidak didukung di tingkat klien.
- Pengaturan throughput yang disediakan diabaikan dalam DynamoDB yang dapat diunduh, meskipun operasi `CreateTable` memerlukannya. Untuk `CreateTable`, Anda dapat menentukan nomor yang Anda inginkan untuk throughput baca dan tulis yang disediakan, meskipun angka-angka ini tidak digunakan. Anda dapat memanggil `UpdateTable` sebanyak yang Anda inginkan per hari. Namun, perubahan nilai throughput yang disediakan akan diabaikan.
- Operasi `Scan` dilakukan secara berurutan. Pemindaian paralel tidak didukung. Parameter `Segment` dan `TotalSegments` dari operasi `Scan` akan diabaikan.

- Kecepatan operasi baca dan tulis pada data tabel hanya dibatasi oleh kecepatan komputer Anda. Operasi `CreateTable`, `UpdateTable`, dan `DeleteTable` terjadi segera, dan status tabel selalu `ACTIVE`. Operasi `UpdateTable` yang hanya mengubah pengaturan `throughput` yang disediakan pada tabel atau indeks sekunder global terjadi segera. Jika operasi `UpdateTable` membuat atau menghapus setiap indeks sekunder global, indeks tersebut melakukan transisi melalui status normal (seperti `CREATING` dan `DELETING`) sebelum status indeks menjadi `ACTIVE`. Tabel tetap `ACTIVE` selama waktu ini.
- Operasi baca bersifat konsisten pada akhirnya. Namun, karena kecepatan DynamoDB yang berjalan di komputer Anda, sebagian besar operasi baca terlihat sangat konsisten.
- Metrik koleksi item dan ukuran koleksi item tidak dilacak. Dalam respons operasi, `null` akan dikembalikan, bukan metrik koleksi item.
- Dalam DynamoDB, terdapat batas data sebesar 1 MB yang dikembalikan per set hasil. Layanan web DynamoDB dan versi yang dapat diunduh menerapkan batas ini. Namun, ketika mengkueri indeks, layanan DynamoDB menghitung hanya ukuran kunci dan atribut terproyeksi. Sebaliknya, versi DynamoDB yang dapat diunduh menghitung ukuran keseluruhan item.
- Jika Anda menggunakan DynamoDB Streams, laju serpihan yang dibuat mungkin berbeda. Dalam layanan web DynamoDB, perilaku pembuatan serpihan sebagian dipengaruhi oleh aktivitas partisi tabel. Ketika Anda menjalankan DynamoDB secara lokal, tidak ada pembuatan partisi tabel. Dalam kedua kasus, serpihan bersifat fana, sehingga aplikasi Anda tidak harus bergantung pada perilaku serpihan.
- `TransactionConflictException` tidak dilemparkan oleh DynamoDB yang dapat diunduh untuk API transaksional. Sebaiknya Anda menggunakan kerangka kerja mocking Java untuk menyimulasikan `TransactionConflictExceptions` di handler DynamoDB untuk menguji cara aplikasi Anda merespons transaksi yang bertentangan.
- Di layanan web DynamoDB, apakah diakses melalui konsol atau, nama tabel peka huruf AWS CLI besar/kecil. Tabel bernama `Authors` dan yang bernama `authors` bisa ada sebagai tabel terpisah. Dalam versi yang dapat diunduh, nama tabel tidak peka huruf besar-kecil, dan upaya membuat kedua tabel ini akan menghasilkan kesalahan.
- Penandaan tidak didukung dalam versi DynamoDB yang dapat diunduh.
- [Versi DynamoDB yang dapat diunduh mengabaikan parameter `Limit` di `ExecuteStatement`](#)

Riwayat rilis DynamoDB lokal

Tabel berikut menjelaskan perubahan penting dalam setiap rilis DynamoDB lokal.

Versi	Perubahan	Deskripsi	Tanggal
2.3.0	Peningkatan Dermaga dan JDK	<ul style="list-style-type: none"> • Upgrade ke Jetty 12.0.2 • Upgrade ke JDK 17 • Upgrade ANTLR4 ke 4.10.1 	Maret 14, 2024
2.2.0	Ditambahkan dukungan untuk perlindungan penghapusan tabel dan parameter <code>ReturnValuesOnConditionCheckFailure</code>	<ul style="list-style-type: none"> • Menambahkan dukungan perlindungan Hapus Tabel • Ditambahkan dukungan untuk <code>ReturnValuesOnConditionCheckFailure</code> • Ditambahkan dukungan untuk bendera <code>-version</code> 	14 Desember 2023
2.1.0	Dukungan untuk proyek SQLite Native Libraries for Maven dan menambahkan telemetri	<ul style="list-style-type: none"> • Menambahkan telemetri ke DynamoDB lokal • Menyalin proyek SQLite Native Libraries for Maven secara dinamis • Menghapus pustaka <code>io.github.ganadist.sqlite4java</code> dari <code>dependencies</code> di Maven • Memutakhirkan ke <code>GoogleGuava 32.1.1-jre</code> 	23 Oktober 2023

Versi	Perubahan	Deskripsi	Tanggal
2.0.0	Migrasi dari javax ke jakarta namespace dan dukungan JDK11	<ul style="list-style-type: none">• Migrasi dari javax ke jakarta namespace dan dukungan JDK11• Perbaikan untuk menangani akses tidak valid dan kunci rahasia saat server dinyalakan• Memperbaiki kerentanan yang diidentifikasi Maven dengan memperbaiki dependensi	5 Juli 2023
1.25.0	Ditambahkan dukungan untuk perlindungan penghapusan tabel dan parameter ReturnValuesOnConditionCheckFailure	<ul style="list-style-type: none">• Menambahkan dukungan perlindungan Hapus Tabel• Ditambahkan dukungan untuk ReturnValuesOnConditionCheckFailure• Ditambahkan dukungan untuk bendera -version	18 Desember 2023

Versi	Perubahan	Deskripsi	Tanggal
1.24.0	Dukungan untuk proyek SQLite Native Libraries for Maven dan menambahkan telemetri	<ul style="list-style-type: none">• Menambahk an telemetri ke DynamoDB lokal• Menyalin proyek SQLite Native Libraries for Maven secara dinamis• Menghapus pustaka io.github .ganadist.sqlite4j ava dari dependens i Maven• Memutakhirkan ke GoogleGuava 32.1.1-jre	23 Oktober 2023
1.23.0	Penanganan akses tidak valid dan kunci rahasia saat server dinyalakan	<ul style="list-style-type: none">• Perbaikan untuk menangani akses tidak valid dan kunci rahasia saat server dinyalakan• Memperbaiki kerentanan yang diidentifikasi Maven dengan memperbar ui dependensi	28 Juni 2023

Versi	Perubahan	Deskripsi	Tanggal
1.22.0	Dukungan Operasi Batas untuk PartiQL	<ul style="list-style-type: none">• Mengoptimalkan klausul IN untuk PartiQL• Dukungan untuk Operasi Batas• Dukungan M1 untuk proyek Maven	8 Juni 2023
1.21.0	Dukungan untuk 100 tindakan per transaksi	<ul style="list-style-type: none">• Meningkatkan tindakan per transaksi dari 25 menjadi 100• Meningkatkan citra docker Open JDK ke 11• Memperbaiki paritas untuk pengecualian yang dilemparkan saat menduplikasi item di BatchExecuteStatement	26 Januari 2023
1.20.0	Menambahkan dukungan untuk M1 Mac	<ul style="list-style-type: none">• Menambahkan dukungan untuk M1 Mac• Meningkatkan dependensi Jetty ke 9.4.48.v20220622	12 September 2022
1.19.0	Peningkatan Parser PartiQL	Meningkatkan Parser PartiQL dan pustaka terkait lainnya	27 Juli 2022

Versi	Perubahan	Deskripsi	Tanggal
1.18.0	Peningkatan log4j-core dan Jackson-core	Meningkatkan log4j-core ke 2.17.1 dan Jackson-core 2.10.x ke 2.12.0	10 Januari 2022
1.17.2	Peningkatan log4j-core	Meningkatkan dependensi log4j-core ke versi 2.16	16 Januari 2021
1.17.1	Peningkatan log4j-core	Pembaruan dependensi log4j-core untuk mem-patch eksploitasi zero-day guna mencegah eksekusi kode jarak jauh - Log4Shell	10 Januari 2021
1.17.0	Menghentikan penggunaan Javascript Web Shell	<ul style="list-style-type: none">• Memperbarui ketergantungan AWS SDK ke AWS SDK for Java 1.12.x• Menghentikan penggunaan Javascript Web Shell	8 Januari 2021

Telemetri di DynamoDB lokal

Pada AWS, kami mengembangkan dan meluncurkan layanan berdasarkan apa yang kami pelajari dari interaksi dengan pelanggan, dan kami menggunakan umpan balik pelanggan untuk mengulangi produk kami. Telemetri adalah informasi tambahan yang membantu kami untuk lebih memahami kebutuhan pelanggan kami, mendiagnosis masalah, dan memberikan fitur yang meningkatkan pengalaman pelanggan.

DynamoDB lokal mengumpulkan telemetri, seperti metrik penggunaan umum, informasi sistem dan lingkungan, serta kesalahan. Untuk detail tipe telemetri yang dikumpulkan, lihat [Jenis informasi yang dikumpulkan](#)

DynamoDB lokal tidak mengumpulkan informasi pribadi, seperti nama pengguna atau alamat email. CLI ini juga tidak mengekstraksi informasi tingkat proyek yang sensitif.

Sebagai pelanggan, Anda mengontrol apakah telemetri dihidupkan, dan Anda dapat mengubah pengaturan kapan saja. Jika telemetri tetap aktif, DynamoDB lokal mengirimkan data telemetri di latar belakang tanpa memerlukan interaksi pelanggan tambahan.

Matikan telemetri menggunakan opsi baris perintah

Anda dapat mematikan telemetri menggunakan opsi baris perintah saat memulai DynamoDB lokal menggunakan opsi. `-disableTelemetry` Untuk informasi selengkapnya, lihat [Opsi baris perintah](#)

Matikan telemetri untuk satu sesi

Di sistem operasi macOS dan Linux, Anda dapat menonaktifkan telemetri untuk satu sesi. Untuk menonaktifkan telemetri sesi Anda saat ini, jalankan perintah berikut untuk mengatur variabel `DDB_LOCAL_TELEMETRY` lingkungan. `false` Ulangi perintah tersebut untuk setiap terminal atau sesi baru.

```
export DDB_LOCAL_TELEMETRY=0
```

Menonaktifkan telemetri untuk profil Anda di semua sesi

Jalankan perintah berikut untuk menonaktifkan telemetri untuk semua sesi saat menjalankan DynamoDB lokal di sistem operasi Anda.

Untuk menonaktifkan telemetri di Linux

1. Jalankan:

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

2. Jalankan:

```
source ~/.profile
```

Untuk menonaktifkan telemetri di macOS

1. Jalankan:

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

2. Jalankan:

```
source ~/.profile
```

Untuk menonaktifkan telemetri di Windows

1. Jalankan:

```
setx DDB_LOCAL_TELEMETRY 0
```

2. Jalankan:

```
refreshenv
```

Jenis informasi yang dikumpulkan

- Informasi penggunaan — Telemetri generik seperti start/stop server dan API atau Operasi disebut.
- Informasi sistem dan lingkungan — Versi Java, sistem operasi (Windows, Linux atau macOS), lingkungan tempat DynamoDB berjalan (misalnya, Stand alone JAR, atau sebagai Ketergantungan Maven), dan nilai hash atribut penggunaan.

Pelajari selengkapnya

Data telemetri yang dikumpulkan lokal DynamoDB mematuhi kebijakan privasi data. AWS Untuk informasi selengkapnya, lihat yang berikut:

- [AWSketentuan layanan](#)
- [FAQ privasi data](#)

Menyiapkan DynamoDB (layanan web)

Untuk menggunakan layanan web Amazon DynamoDB:

1. [Daftar ke AWS](#).
2. [Dapatkan kunci AWS akses](#) (digunakan untuk mengakses DynamoDB secara terprogram).

Note

Jika Anda berencana untuk berinteraksi dengan DynamoDB hanya melalui AWS Management Console, Anda tidak memerlukan AWS kunci akses, dan Anda dapat melompat ke depan. [Menggunakan konsol](#)

3. [Konfigurasi kredensial Anda](#) (digunakan untuk mengakses DynamoDB secara terprogram).

Mendaftar untuk AWS

Untuk menggunakan layanan DynamoDB, Anda harus memiliki akun AWS. Jika belum memiliki akun, Anda diminta membuatnya saat mendaftar. Anda tidak dikenakan biaya untuk AWS layanan apa pun yang Anda daftarkan kecuali Anda menggunakannya.

Untuk mendaftar AWS

1. Buka <https://portal.aws.amazon.com/billing/signup>.
2. Ikuti petunjuk secara online.

Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWS dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik keamanan, [tetapkan akses administratif ke pengguna administratif](#), dan hanya gunakan pengguna root untuk melakukan [tugas yang memerlukan akses pengguna root](#).

Memberikan akses terprogram

Sebelum Anda dapat mengakses DynamoDB secara terprogram atau melalui AWS CLI(), Anda harus memiliki akses AWS Command Line Interface terprogram. Anda tidak memerlukan akses terprogram jika Anda berencana menggunakan hanya konsol DynamoDB.

Pengguna membutuhkan akses terprogram jika mereka ingin berinteraksi dengan AWS luar. AWS Management Console Cara untuk memberikan akses terprogram tergantung pada jenis pengguna yang mengakses AWS.

Untuk memberi pengguna akses terprogram, pilih salah satu opsi berikut.

Pengguna mana yang membutuhkan akses terprogram?	Untuk	Oleh
Identitas tenaga kerja (Pengguna yang dikelola di Pusat Identitas IAM)	Gunakan kredensial sementara untuk menandatangani permintaan terprogram ke AWS CLI, AWS SDK, atau API. AWS	Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan. <ul style="list-style-type: none"> • Untuk AWS CLI, lihat Mengkonfigurasi yang akan AWS CLI digunakan AWS IAM Identity Center dalam Panduan AWS Command Line Interface Pengguna. • Untuk AWS SDK, alat, dan AWS API, lihat otentikasi Pusat Identitas IAM di Panduan Referensi AWS SDK dan Alat.
IAM	Gunakan kredensial sementara untuk menandatangani permintaan terprogram ke AWS CLI, AWS SDK, atau API. AWS	Mengikuti petunjuk dalam Menggunakan kredensial sementara dengan AWS sumber daya di Panduan Pengguna IAM.

Pengguna mana yang membutuhkan akses terprogram?	Untuk	Oleh
IAM	(Tidak direkomendasikan) Gunakan kredensial jangka panjang untuk menandatangani permintaan terprogram ke AWS CLI, AWS SDK, atau API. AWS	<p>Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan.</p> <ul style="list-style-type: none"> • Untuk mengetahui AWS CLI, lihat Mengautentikasi menggunakan kredensial pengguna IAM di Panduan Pengguna. AWS Command Line Interface • Untuk AWS SDK dan alat bantu, lihat Mengautentikasi menggunakan kredensial jangka panjang di Panduan Referensi AWS SDK dan Alat. • Untuk AWS API, lihat Mengelola kunci akses untuk pengguna IAM di Panduan Pengguna IAM.

Mengonfigurasi kredensial Anda

Sebelum Anda dapat mengakses DynamoDB secara terprogram atau melalui AWS CLI, Anda harus mengonfigurasi kredensialnya untuk mengaktifkan otorisasi untuk aplikasi Anda.

Ada beberapa cara untuk melakukan ini. Misalnya, Anda dapat secara manual membuat file kredensial untuk menyimpan ID kunci akses dan kunci akses rahasia Anda. Anda juga dapat menggunakan AWS CLI perintah `aws configure` untuk membuat file secara otomatis.

Alternatifnya, Anda dapat menggunakan variabel lingkungan. Untuk informasi selengkapnya tentang mengonfigurasi kredensial Anda, lihat panduan pengembang SDK khusus pemrograman AWS .

Untuk menginstal dan mengkonfigurasi AWS CLI, lihat [Menggunakan AWS CLI](#).

Integrasi dengan layanan DynamoDB lainnya

Anda dapat mengintegrasikan DynamoDB dengan banyak layanan lainnya. AWS Untuk informasi selengkapnya, lihat hal berikut:

- [Menggunakan DynamoDB dengan layanan AWS lain](#)
- [AWS CloudFormation untuk DynamoDB](#)
- [Menggunakan AWS Backup dengan DynamoDB](#)
- [AWS Identity and Access Management \(IAM\)](#)
- [Menggunakan AWS Identity and Access Management dengan DynamoDB](#)

Mengakses DynamoDB

Anda dapat mengakses Amazon DynamoDB menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau API DynamoDB.

Topik

- [Menggunakan konsol](#)
- [Menggunakan AWS CLI](#)
- [Menggunakan API](#)
- [Menggunakan NoSQL Workbench untuk DynamoDB](#)
- [Rentang alamat IP](#)

Menggunakan konsol

Anda dapat mengakses AWS Management Console untuk Amazon DynamoDB di <https://console.aws.amazon.com/dynamodb/home>.

Anda dapat menggunakan konsol untuk melakukan hal berikut di DynamoDB:

- Memantau peringatan terbaru, total kapasitas, kesehatan layanan, dan berita DynamoDB terbaru di dasbor DynamoDB.
- Membuat, memperbarui, dan menghapus tabel. Kalkulator kapasitas memberi perkiraan jumlah unit kapasitas untuk diminta berdasarkan informasi penggunaan yang Anda berikan.
- Mengelola aliran.
- Melihat, menambah, memperbarui, dan menghapus item yang disimpan dalam tabel. Mengelola Waktu untuk Beroperasi (TTL) untuk menentukan kapan item dalam tabel berakhir agar item tersebut dapat secara otomatis dihapus dari basis data.
- Mengajukan kueri dan memindai tabel.
- Mengatur dan melihat alarm untuk memantau penggunaan kapasitas tabel Anda. Lihat metrik pemantauan teratas tabel Anda pada grafik real-time dari CloudWatch
- Memodifikasi kapasitas yang disediakan sebuah tabel.
- Memodifikasi kelas tabel pada sebuah tabel.
- Membuat dan menghapus indeks sekunder global.
- Membuat pemicu untuk menghubungkan aliran DynamoDB ke fungsi AWS Lambda.

- Menerapkan tag ke sumber daya Anda untuk membantu mengatur dan mengidentifikasi sumber daya tersebut.
- Membeli kapasitas terpesan.

Konsol menampilkan layar pengantar yang meminta Anda untuk membuat tabel pertama Anda. Untuk melihat tabel Anda, pilih Tabel pada panel navigasi di sisi kiri konsol.

Berikut ini adalah ikhtisar tingkat tinggi tentang tindakan yang tersedia per tabel dalam setiap tab navigasi:

- Ikhtisar – Melihat detail tabel, termasuk jumlah item dan metrik.
- Indeks – Mengelola indeks sekunder global dan lokal.
- Monitor — Lihat alarm, Wawasan CloudWatch Kontributor, dan metrik Cloudwatch.
- Tabel global – Mengelola replika tabel.
- Cadangan - Kelola point-in-time pemulihan dan cadangan sesuai permintaan.
- Ekspor dan aliran – Mengekspor tabel Anda ke Amazon S3 dan mengelola DynamoDB Streams dan Kinesis Data Streams.
- Pengaturan tambahan – Mengelola kapasitas baca/tulis, Pengaturan Waktu untuk Beroperasi, enkripsi, dan tag.

Menggunakan AWS CLI

Anda dapat menggunakan AWS Command Line Interface (AWS CLI) untuk mengontrol beberapa layanan AWS dari baris perintah dan mengotomatiskan layanan tersebut melalui skrip. Anda dapat menggunakan AWS CLI untuk operasi ad hoc, seperti membuat tabel. Anda juga dapat menggunakannya untuk menanamkan operasi Amazon DynamoDB dalam skrip utilitas.

Sebelum dapat menggunakan AWS CLI dengan DynamoDB, Anda harus mendapatkan ID kunci akses dan kunci akses rahasia. Untuk informasi selengkapnya, lihat [Memberikan akses terprogram](#) .

Untuk daftar lengkap tentang semua perintah yang tersedia untuk DynamoDB di AWS CLI, lihat [referensi perintah AWS CLI](#).

Topik

- [Mengunduh dan mengonfigurasi AWS CLI](#)
- [Menggunakan AWS CLI dengan DynamoDB](#)

- [Menggunakan AWS CLI dengan DynamoDB lokal](#)

Mengunduh dan mengonfigurasi AWS CLI

AWS CLI tersedia di <http://aws.amazon.com/cli>. Alat ini berjalan di Windows, macOS, atau Linux. Setelah Anda mengunduh AWS CLI, ikuti langkah berikut untuk menginstal dan mengonfigurasinya:

1. Buka [Panduan Pengguna AWS Command Line Interface](#).
2. Ikuti petunjuk untuk [Menginstal AWS CLI](#) dan [Mengonfigurasi AWS CLI](#).

Menggunakan AWS CLI dengan DynamoDB

Format baris perintah terdiri dari nama operasi DynamoDB, diikuti oleh parameter untuk operasi tersebut. AWS CLI mendukung sintaks singkatan untuk nilai parameter, serta JSON.

Misalnya, perintah berikut membuat tabel bernama Music. Kunci partisi adalah Artist, dan kunci sortir adalah SongTitle. (Agar mudah dibaca, perintah panjang di bagian ini dipecah menjadi baris terpisah.)

```
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=1,WriteCapacityUnits=1 \  
  --table-class STANDARD
```

Perintah berikut menambahkan item baru ke tabel. Contoh-contoh ini menggunakan kombinasi sintaks singkatan dan JSON.

```
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
```

Pada baris perintah, menulis JSON yang valid bisa menjadi hal yang sulit. Namun, AWS CLI dapat membaca file JSON. Sebagai contoh, perhatikan potongan kode JSON berikut, yang disimpan dalam sebuah file bernama `key-conditions.json`.

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

Sekarang, Anda dapat menerbitkan permintaan Query menggunakan AWS CLI. Dalam contoh ini, isi file `key-conditions.json` digunakan untuk parameter `--key-conditions`.

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

Menggunakan AWS CLI dengan DynamoDB lokal

Ini juga AWS CLI dapat berinteraksi dengan DynamoDB lokal (versi yang dapat diunduh) yang berjalan di komputer Anda. Untuk mengaktifkan hal ini, tambahkan parameter berikut ke setiap perintah:

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

Contoh berikut menggunakan AWS CLI untuk mencantumkan tabel dalam basis data lokal.

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

Jika DynamoDB menggunakan nomor port selain nomor default (8000), modifikasi nilai `--endpoint-url` sebagaimana mestinya.

Note

Tidak AWS CLI dapat menggunakan DynamoDB lokal (versi yang dapat diunduh) sebagai titik akhir default. Oleh karena itu, Anda harus menentukan `--endpoint-url` dengan setiap perintah.

Menggunakan API

Anda dapat menggunakan AWS Management Console dan AWS Command Line Interface untuk bekerja secara interaktif dengan Amazon DynamoDB. Namun, untuk mendapatkan hasil maksimal dari DynamoDB, Anda dapat menulis kode aplikasi menggunakan SDK AWS.

[AWS SDK memberikan dukungan luas untuk DynamoDB di Java, JavaScript, dibrowser, .NET, Node.js, PHP, Python, Ruby, C ++, Go, Android, dan iOS.](#) Untuk segera memulai dengan bahasa ini, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Sebelum dapat menggunakan SDK AWS dengan DynamoDB, Anda harus mendapatkan ID kunci akses AWS dan kunci akses rahasia. Untuk informasi selengkapnya, lihat [Menyiapkan DynamoDB \(layanan web\)](#).

Untuk gambaran umum tingkat tinggi tentang pemrograman aplikasi DynamoDB dengan SDK AWS, lihat [Pemrograman dengan DynamoDB dan SDK AWS](#).

Menggunakan NoSQL Workbench untuk DynamoDB

Anda juga dapat mengakses DynamoDB dengan mengunduh dan menggunakan [NoSQL Workbench untuk DynamoDB](#).

NoSQL Workbench untuk Amazon DynamoDB adalah aplikasi GUI sisi klien lintas platform yang dapat Anda gunakan untuk pengembangan dan operasi database modern. Aplikasi ini tersedia untuk Windows, macOS, dan Linux. NoSQL Workbench adalah alat pengembangan visual yang menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri untuk membantu Anda merancang, membuat, melakukan kueri, serta mengelola tabel DynamoDB. NoSQL Workbench kini menyertakan DynamoDB lokal sebagai bagian opsional dari proses instalasi, yang mempermudah Anda untuk memodelkan data Anda di DynamoDB lokal. Untuk mempelajari selengkapnya tentang DynamoDB lokal dan persyaratannya, lihat [Menyiapkan DynamoDB lokal \(versi yang dapat diunduh\)](#)

Note

NoSQL Workbench untuk DynamoDB saat ini tidak AWS mendukung login yang dikonfigurasi dengan otentikasi dua faktor (2FA).

Pemodelan data

Dengan NoSQL Workbench untuk DynamoDB, Anda dapat membuat model data baru dari, atau mendesain model berdasarkan, model data yang sudah ada yang memenuhi pola akses data aplikasi Anda. Anda juga dapat mengimpor dan mengekspor model data yang didesain pada akhir proses. Untuk informasi selengkapnya, lihat [Membangun model data dengan NoSQL Workbench](#).

Visualisasi data

Pemvisualisasi model data menyediakan kanvas tempat Anda dapat memetakan kueri dan memvisualisasikan pola akses (faset) aplikasi tanpa harus menulis kode. Setiap faset berhubungan dengan pola akses yang berbeda di DynamoDB. Anda dapat membuat data sampel secara otomatis untuk digunakan dalam model data Anda. Untuk informasi selengkapnya, lihat [Memvisualisasikan pola akses data](#).

Pembangunan operasi

NoSQL Workbench menyediakan antarmuka pengguna grafis yang kaya bagi Anda untuk mengembangkan dan menguji kueri. Anda dapat menggunakan pembangun operasi untuk melihat, menjelajahi, dan melakukan kueri set data langsung. Anda juga dapat menggunakan pembangun operasi terstruktur untuk membuat dan menjalankan operasi bidang data. Fitur ini mendukung proyeksi dan ekspresi kondisi, serta memungkinkan Anda menghasilkan kode sampel dalam berbagai bahasa. Untuk informasi selengkapnya, lihat [Menjelajahi set data dan membangun operasi dengan NoSQL Workbench](#).

Rentang alamat IP

Amazon Web Services (AWS) menerbitkan rentang alamat IP saat ini dalam format JSON. Untuk melihat rentang saat ini, unduh [ip-ranges.json](#). Untuk informasi selengkapnya, lihat [Rentang alamat IP AWS](#) di Referensi Umum AWS.

Untuk menemukan rentang alamat IP yang dapat Anda gunakan untuk [mengakses tabel dan indeks DynamoDB](#), cari file ip-ranges.json untuk string berikut: "service": "DYNAMODB".

Note

Rentang alamat IP tidak berlaku untuk DynamoDB Streams atau DynamoDB Accelerator (DAX).

Mulai menggunakan DynamoDB

Gunakan tutorial praktis di bagian ini untuk membantu Anda memulai dan mempelajari selengkapnya tentang Amazon DynamoDB.

Topik

- [Konsep dasar di DynamoDB](#)
- [Prasyarat - tutorial memulai](#)
- [Langkah 2: Buat tabel](#)
- [Langkah 2: Tulis data ke tabel menggunakan konsol atau AWS CLI](#)
- [Langkah 3: Baca data dari Tabel](#)
- [Langkah 4: Perbarui data di Tabel](#)
- [Langkah 5: Cari data dalam tabel](#)
- [Langkah 6: Buat indeks sekunder global](#)
- [Langkah 7: Kueri indeks sekunder global](#)
- [Langkah 8: \(Opsional\) hapus sumber daya](#)
- [Mulai menggunakan DynamoDB: Langkah selanjutnya](#)

Konsep dasar di DynamoDB

Sebelum memulai, Anda harus membiasakan diri dengan konsep dasar di Amazon DynamoDB. Untuk informasi selengkapnya, lihat [Komponen inti DynamoDB](#).

Kemudian, lanjutkan ke [Prasyarat](#) untuk mempelajari tentang menyiapkan DynamoDB.

Prasyarat - tutorial memulai

Sebelum memulai tutorial Amazon DynamoDB, ikuti langkah-langkah dalam [Menyiapkan DynamoDB](#). Kemudian lanjutkan ke [Langkah 2: Buat tabel](#).

Note

- Jika berencana untuk berinteraksi dengan DynamoDB hanya melalui AWS Management Console, Anda tidak memerlukan kunci akses AWS. Selesaikan langkah-langkah dalam [Mendaftar AWS](#), lalu lanjutkan ke [Langkah 2: Buat tabel](#).
- Jika tidak ingin mendaftar untuk akun tingkat gratis, Anda dapat mengatur [DynamoDB Lokal \(Versi yang dapat diunduh\)](#). Kemudian lanjutkan ke [Langkah 2: Buat tabel](#).
- Ada beberapa perbedaan saat menggunakan perintah CLI di terminal di Linux dan Windows. Panduan berikut menyajikan perintah yang diformat untuk terminal Linux (ini termasuk macOS), dan perintah yang diformat untuk Windows CMD. Pilih perintah yang paling sesuai dengan aplikasi terminal yang Anda gunakan.

Langkah 2: Buat tabel

Pada langkah ini, Anda membuat tabel `Music` di Amazon DynamoDB. Tabel ini memiliki detail berikut:

- Kunci partisi — `Artist`
- Kunci urutan — `SongTitle`

Untuk informasi selengkapnya tentang operasi tabel, lihat [Bekerja dengan tabel dan data di DynamoDB](#).

Note

Sebelum memulai, pastikan Anda mengikuti langkah-langkah dalam [Prasyarat - tutorial memulai](#).

AWS Management Console

Untuk membuat tabel `Music` baru menggunakan konsol DynamoDB:

1. Masuk ke AWS Management Console, lalu buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.

2. Di panel navigasi di sisi kiri konsol, pilih Dasbor.
3. Di sisi kanan konsol, pilih Buat Tabel.

Create a table

Create an Amazon DynamoDB table for fast and predictable database performance at any scale.

[Learn more](#) 

Create table

4. Masukkan detail tabel sebagai berikut:
 - a. Untuk nama tabel, masukkan **Music**.
 - b. Untuk kunci partisi, masukkan **Artist**.
 - c. Masukkan **SongTitle** sebagai kunci urutan.
 - d. Biarkan Pengaturan default dipilih.
5. Pilih Buat untuk membuat tabel.

Create table

Table details [Info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

 ▼

1 to 255 characters and case sensitive.

Sort key - *optional*

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

 ▼

1 to 255 characters and case sensitive.

Settings

Default settings

The fastest way to create your table. You can modify these settings now or after your table has been created.

Customize settings

Use these advanced features to make DynamoDB work better for your needs.

Default settings

Read/write capacity [Info](#)

Using provisioned capacity mode. Read and write capacity are set to 5 units each with auto scaling enabled.

Secondary indexes [Info](#)

No secondary indexes have been created. Queries will be run by using the table's partition key and sort key only.

Key management for encryption at rest [Info](#)

Using the AWS owned customer master key. This key is managed by DynamoDB at no extra cost.

Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

6. Setelah tabel berstatus ACTIVE, praktik terbaiknya adalah mengaktifkan [oint-in-time Pemulihan P untuk DynamoDB](#) pada tabel dengan melakukan langkah-langkah berikut:
 - a. Klik nama tabel untuk membuka tabel.
 - b. Klik Cadangan.
 - c. Klik tombol Edit di bagian oint-in-time pemulihan P.
 - d. Klik kotak centang untuk mengaktifkan point-in-time pemulihan, lalu klik tombol Simpan perubahan.

oint-in-time Pemulihan P sekarang diaktifkan untuk tabel Musik yang baru Anda buat.

AWS CLI

Contoh AWS CLI berikut membuat tabel Music baru menggunakan create-table.

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 \  
  --provisioned-throughput \  
    ReadCapacityUnits=5,WriteCapacityUnits=5 \  
  --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 ^  
  --provisioned-throughput ^
```

```
ReadCapacityUnits=5,WriteCapacityUnits=5 ^  
--table-class STANDARD
```

Menggunakan `create-table` akan menampilkan hasil sampel berikut.

```
{  
  "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"  
    }  
  }  
}
```

```
}  
}
```

Perhatikan bahwa nilai bidang `TableStatus` diatur ke `CREATING`.

Untuk memverifikasi bahwa DynamoDB selesai membuat tabel `Music`, gunakan perintah `describe-table`.

Linux

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

Windows CMD

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

Perintah ini akan menampilkan hasil berikut. Setelah DynamoDB selesai membuat tabel, nilai bidang `TableStatus` diatur ke `ACTIVE`.

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

Setelah tabel berstatus `ACTIVE`, praktik terbaiknya adalah mengaktifkan [point-in-time Pemulihan P untuk DynamoDB](#) pada tabel dengan menjalankan perintah berikut:

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
```

Perintah ini akan menampilkan hasil berikut.

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

Note

Ada implikasi biaya untuk memungkinkan pencadangan berkelanjutan dengan pemulihan point-in-time Untuk informasi selengkapnya tentang harga, lihat [Harga Amazon DynamoDB](#).

Setelah membuat tabel baru, lanjutkan ke [Langkah 2: Tulis data ke tabel menggunakan konsol atau AWS CLI](#).

Langkah 2: Tulis data ke tabel menggunakan konsol atau AWS CLI

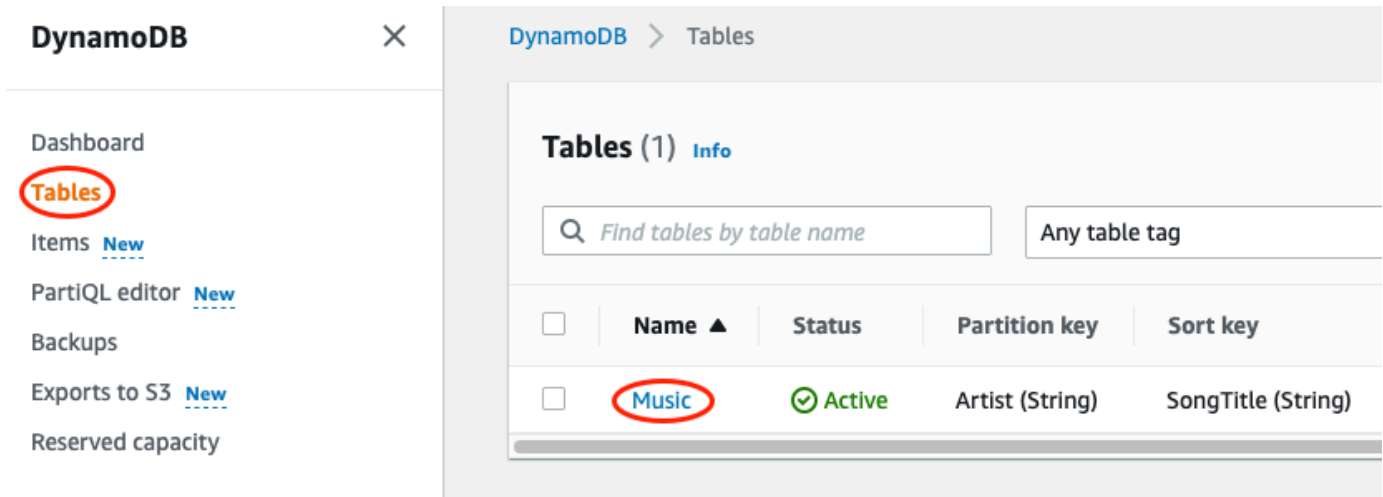
Pada langkah ini, Anda akan memasukkan beberapa item ke tabel `Music` yang Anda buat di [Langkah 2: Buat tabel](#).

Untuk informasi selengkapnya tentang operasi tulis, lihat [Menulis item](#).

AWS Management Console

Ikuti langkah-langkah ini untuk menulis data ke tabel `Music` menggunakan konsol DynamoDB.

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi di sisi kiri konsol, pilih `Tabel`.
3. Di daftar tabel, pilih tabel `Musik`.



DynamoDB ×

DynamoDB > Tables

Tables (1) Info

Find tables by table name Any table tag

<input type="checkbox"/>	Name ▲	Status	Partition key	Sort key
<input type="checkbox"/>	Music	Active	Artist (String)	SongTitle (String)

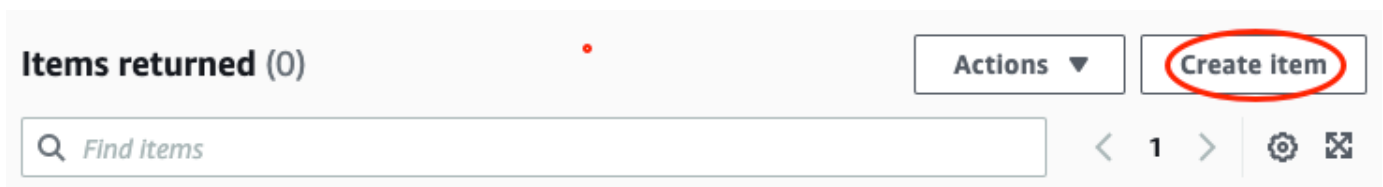
- Pilih Jelajahi Item Tabel.



Music

Actions ▼ Explore table items

- Di tampilan Item, pilih Buat item.

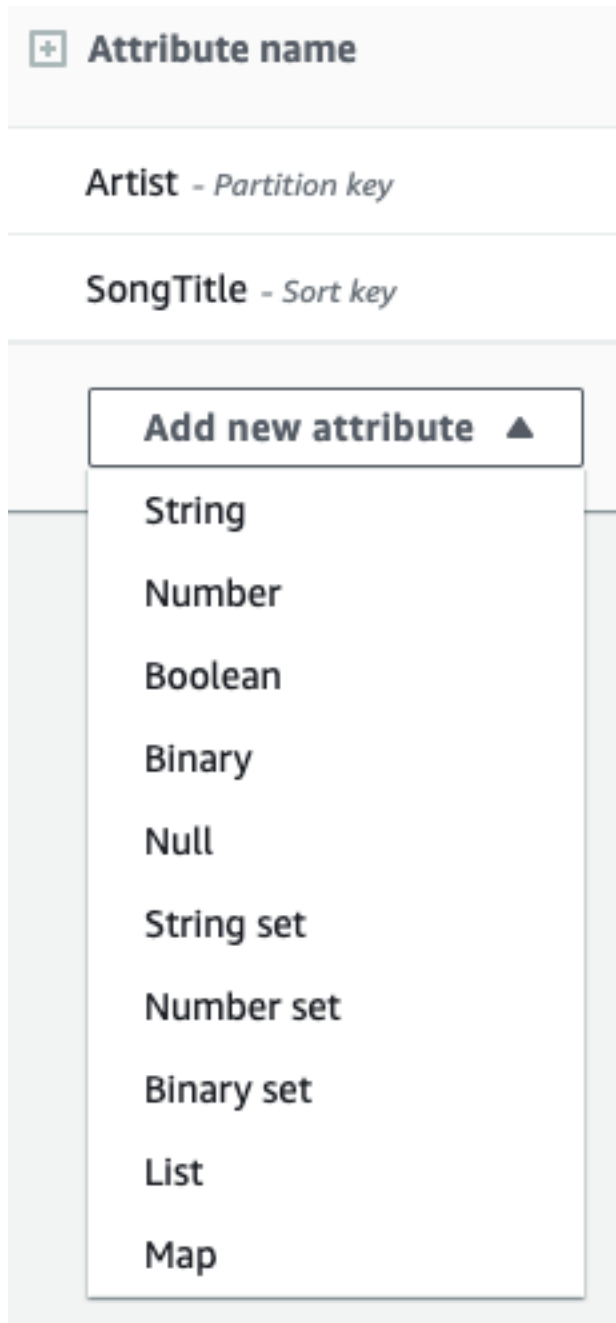


Items returned (0)

Actions ▼ Create item

Find Items < 1 > ⚙️

- Pilih Tambahkan atribut baru, lalu pilih Nomor. Beri nama bidang **Awards**.



7. Ulangi proses ini untuk membuat **AlbumTitle** jenis String.
8. Masukkan nilai berikut untuk item Anda:
 - a. Untuk Artist, masukkan **No One You Know** sebagai nilai.
 - b. Untuk SongTitle, masukkan **Call Me Today**.
 - c. Untuk AlbumTitle, masukkan **Somewhat Famous**.
 - d. Untuk Awards, masukkan **1**.

9. Pilih Buat item.

Create item

Form

JSON

Attributes

Attribute name	Value	Type	
Artist - <i>Partition key</i>	No One You Know	String	
SongTitle - <i>Sort key</i>	Call Me Today	String	
Awards	1	Number	<input type="button" value="Remove"/>
AlbumTitle	Somewhat Famous	String	<input type="button" value="Remove"/>
<input style="border: none; background: none; cursor: pointer; text-decoration: none; color: #0070c0; font-weight: normal; text-align: left; padding: 0 10px; border-bottom: 1px solid #ccc; border-right: 1px solid #ccc; border-left: 1px solid #ccc; border-top: 1px solid #ccc; width: 100%;" type="button" value="Add new attribute"/>			

Create Item

10. Ulangi proses ini dan buat item lain dengan nilai berikut:

- a. Untuk Artist, masukkan **Acme Band**.
- b. Untuk SongTitle masuk **Happy Day**.
- c. Untuk AlbumTitle, masukkan **Songs About Life**.
- d. Untuk Awards, masukkan **10**.

11. Lakukan ini sekali lagi untuk membuat item lain dengan Artist yang sama seperti pada langkah sebelumnya, tetapi nilai untuk atribut lainnya berbeda:

- a. Untuk Artist, masukkan **Acme Band**.
- b. Untuk SongTitle masuk **PartiQL Rocks**.
- c. Untuk AlbumTitle, masukkan **Another Album Title**.
- d. Untuk Awards, masukkan **8**.

AWS CLI

Contoh berikut AWS CLI membuat beberapa item baru di tabel `Music`. Anda dapat melakukan ini melalui API DynamoDB atau [PartiQL](#), bahasa kueri yang kompatibel dengan SQL untuk DynamoDB.

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'}"
```

Untuk informasi selengkapnya tentang menulis data dengan PartiQL, lihat [Pernyataan sisipkan PartiQL](#).

Untuk informasi selengkapnya tentang jenis data yang didukung di DynamoDB, lihat [Jenis data](#).

Untuk informasi selengkapnya tentang cara mewakili jenis data DynamoDB di JSON, lihat [Nilai atribut](#).

Setelah menulis data ke tabel Anda, lanjutkan ke [Langkah 3: Baca data dari Tabel](#).

Langkah 3: Baca data dari Tabel

Pada langkah ini, Anda akan membaca kembali item yang dibuat di [Langkah 2: Tulis data ke tabel menggunakan konsol atau AWS CLI](#). Anda dapat menggunakan konsol DynamoDB atau AWS CLI untuk membaca item dari tabel Music dengan menentukan Artist dan SongTitle.

Untuk informasi selengkapnya tentang operasi baca di DynamoDB, lihat [Membaca item](#).

AWS Management Console

Ikuti langkah-langkah ini untuk membaca data dari tabel Music menggunakan konsol DynamoDB.

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tabel Musik dari daftar tabel.
4. Pilih Jelajahi item tabel.
5. Di tab Item, lihat daftar item yang disimpan dalam tabel, yang diurutkan berdasarkan Artist dan SongTitle. Item pertama dalam daftar adalah item dengan Artist Acme Band dan SongTitle Happy Day.

DynamoDB > Items

Items Info

Autopreview

> **▼ Music** View table details

Query **Scan**

Table or index
Music

Artist (Partition key)
Enter partition key value

SongTitle (Sort key)
Equal to Sort descending

► Filters

Run **Reset**

✔ **Completed** Read capacity units consumed: 0.5

Items returned (2) Actions ▼ Create item

< 1 >

<input type="checkbox"/>	Artist ▼	SongTitle ▼	AlbumTitle ▼	Awards ▼
<input type="checkbox"/>	Acme Band	Happy Day	Songs Abou...	10
<input type="checkbox"/>	No One You...	Call Me Today	Somewhat ...	1

AWS CLI

Contoh AWS CLI berikut membaca item dari `Music`. Anda dapat melakukan ini melalui API DynamoDB atau [PartiQL](#), bahasa kueri yang kompatibel dengan SQL untuk DynamoDB.

DynamoDB API

Note

Perilaku default untuk DynamoDB adalah bacaan akhir konsisten. Parameter `consistent-read` digunakan di bawah ini untuk menunjukkan bacaan sangat konsisten.

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  
  \"/>
```

Menggunakan `get-item` akan menampilkan hasil sampel berikut.

```
{  
  "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'"
```

Menggunakan pernyataan Select PartiQL akan menampilkan hasil sampel berikut.

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

Untuk informasi selengkapnya tentang membaca data dengan PartiQL, lihat [Pernyataan pilih PartiQL](#).

Untuk memperbarui data di tabel Anda, lanjutkan ke [Langkah 4: Perbarui data di Tabel](#).

Langkah 4: Perbarui data di Tabel

Pada langkah ini, Anda akan memperbarui item yang Anda buat di [Langkah 2: Tulis data ke tabel menggunakan konsol atau AWS CLI](#). Anda dapat menggunakan konsol DynamoDB atau AWS CLI untuk memperbarui AlbumTitle dari item dalam tabel Music dengan menentukan Artist, SongTitle, dan AlbumTitle yang telah diperbarui.

Untuk informasi selengkapnya tentang operasi tulis, lihat [Menulis item](#).

AWS Management Console

Anda dapat menggunakan konsol DynamoDB untuk memperbarui data dalam tabel Music.

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tabel Musik dari daftar tabel.
4. Pilih Lihat item.
5. Pilih item yang nilai Artist-nya adalah Acme Band dan nilai SongTitle-nya adalah Happy Day.
6. Perbarui AlbumTitlenilainya **Updated Album Title**, lalu pilih Simpan.

Gambar berikut menunjukkan item yang diperbarui di konsol.

Attribute name	Value	Type
Artist - Partition key	Acme Band	String
SongTitle - Sort key	Happy Day	String
AlbumTitle	Updated Album Title	String
Awards	10	Number

AWS CLI

Contoh AWS CLI berikut memperbarui item dalam tabel Music. Anda dapat melakukan ini melalui API DynamoDB atau [PartiQL](#), bahasa kueri yang kompatibel dengan SQL untuk DynamoDB.

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  
  \"/>
```

Menggunakan `update-item` akan menampilkan hasil sampel berikut karena `return-values ALL_NEW` ditentukan.

```
{  
  "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
*"
```

Menggunakan pernyataan Update akan menampilkan hasil sampel berikut karena RETURNING ALL NEW * ditentukan.

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

Untuk informasi selengkapnya tentang memperbarui data dengan PartiQL, lihat [Pernyataan perbarui PartiQL](#).

Untuk mengkueri data dalam tabel `Music`, lanjutkan ke [Langkah 5: Cari data dalam tabel](#).

Langkah 5: Cari data dalam tabel

Pada langkah ini, Anda akan mengkueri data yang ditulis ke tabel `Music` di [the section called “Langkah 2: Tulis data”](#) dengan menentukan `Artist`. Ini akan menampilkan semua lagu yang terkait dengan kunci partisi: `Artist`.

Untuk informasi selengkapnya tentang operasi kueri, lihat [Operasi kueri di DynamoDB](#).

AWS Management Console

Ikuti langkah-langkah ini untuk menggunakan konsol DynamoDB guna mengkueri data di tabel `Music`.

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi di sisi kiri konsol, pilih `Tabel`.
3. Pilih tabel `Musik` dari daftar tabel.
4. Pilih `Jelajahi item tabel`.
5. Pilih `Kueri`.
6. Untuk kunci partisi, masukkan **Acme Band**, lalu pilih `Jalankan`.

DynamoDB > Items

Items Info

Autopreview

> **▼ Music** View table details

Query **Scan**

Table or index
Music ▼

Artist (Partition key)
Acme Band

SongTitle (Sort key)
Equal to ▼ Sort descending

► Filters

Run Reset

AWS CLI

Contoh AWS CLI berikut akan mengkueri item dalam tabel `Music`. Anda dapat melakukan ini melalui API DynamoDB atau [PartiQL](#), bahasa kueri yang kompatibel dengan SQL untuk DynamoDB.

DynamoDB API

Anda akan mengkueri item melalui API DynamoDB dengan menggunakan query dan menyediakan kunci partisi.

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\"}}"
```

Menggunakan query akan menampilkan semua lagu yang terkait dengan Artist ini.

```
{
  "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

Anda akan mengkueri item melalui PartiQL dengan menggunakan pernyataan Select dan menyediakan kunci partisi.

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'"
```

Menggunakan pernyataan Select dengan cara ini akan menampilkan semua lagu yang terkait dengan Artist ini.

```
{
  "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"
      }
    }
  ]
}
```

Untuk informasi selengkapnya tentang mengkueri data dengan PartiQL, lihat [Pernyataan pilih PartiQL](#).

Untuk membuat indeks sekunder global untuk tabel Anda, lanjutkan ke [Langkah 6: Buat indeks sekunder global](#).

Langkah 6: Buat indeks sekunder global

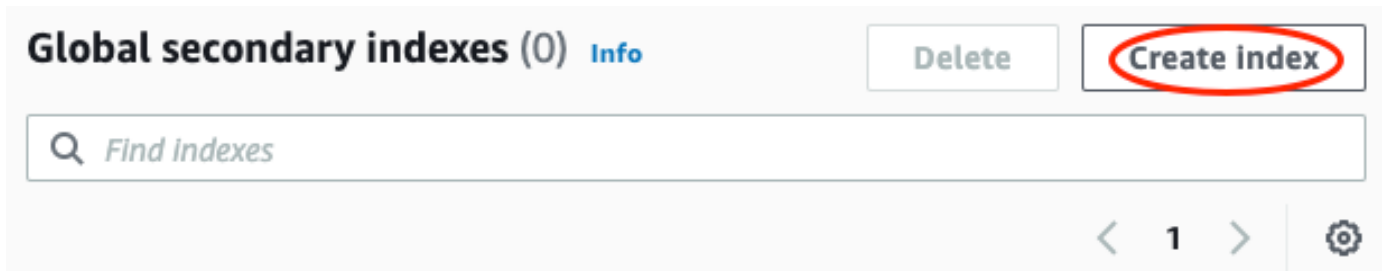
Pada langkah ini, Anda akan membuat indeks sekunder global untuk tabel `Music` yang Anda buat di [Langkah 2: Buat tabel](#).

Untuk informasi selengkapnya tentang indeks sekunder global, lihat [Menggunakan Indeks Sekunder Global di DynamoDB](#).

AWS Management Console

Untuk menggunakan konsol Amazon DynamoDB guna membuat indeks sekunder global `AlbumTitle-index` untuk tabel `Music`:

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi di sisi kiri konsol, pilih `Tabel`.
3. Pilih tabel `Musik` dari daftar tabel.
4. Pilih tab `Indeks` untuk tabel `Musik`.
5. Pilih `Buat indeks`.



6. Untuk Kunci partisi, masukkan **AlbumTitle**.
7. Untuk Nama indeks, masukkan **AlbumTitle-index**.
8. Biarkan pengaturan lain tetap default dan pilih Buat indeks.

Create global secondary index [Info](#)

Global secondary indexes allow you to perform queries on attributes that are not part of a table's primary key. Note that global secondary index read and write capacity settings are separate from those of the table, and they will incur additional costs.

Index details [Info](#)

Partition key

AlbumTitle

1 to 255 characters.

Data type

String

Sort key - optional

Enter the sort key name

1 to 255 characters.

Data type

String

Index name

AlbumTitle-index

Between 3 and 255 characters. Only A-Z, a-z, 0-9, underscore characters, hyphens, and periods allowed.

Index capacity [Info](#)

Read capacity

Read capacity settings

- Copy from base table
- Customize settings

Auto scaling [Info](#)

Dynamically adjusts provisioned throughput capacity on your behalf in response to actual traffic patterns.

- On
- Off

Minimum capacity units

1

Maximum capacity units

10

Target utilization (%)

70

Write capacity

Write capacity settings

- Copy from base table
- Customize settings

Auto scaling [Info](#)

Dynamically adjusts provisioned throughput capacity on your behalf in response to actual traffic patterns.

- On

AWS CLI

Contoh AWS CLI berikut akan membuat indeks sekunder global AlbumTitle-index untuk tabel Music menggunakan update-table.

Linux

```
aws dynamodb update-table \
  --table-name Music \
  --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
  --global-secondary-index-updates \
    [{"Create":{"IndexName": "AlbumTitle-index","KeySchema":
[{"AttributeName":"AlbumTitle","KeyType":"HASH"}], \
  "ProvisionedThroughput": {"ReadCapacityUnits": 10, "WriteCapacityUnits":
5},"Projection":{"ProjectionType":"ALL"}}}]"
```

Windows CMD

```
aws dynamodb update-table ^
  --table-name Music ^
  --attribute-definitions AttributeName=AlbumTitle,AttributeType=S ^
  --global-secondary-index-updates [{"Create":{"IndexName": "AlbumTitle-
index","KeySchema":[{"AttributeName":"AlbumTitle","KeyType":"HASH"}],
  "ProvisionedThroughput": {"ReadCapacityUnits": 10, "WriteCapacityUnits": 5},
  "Projection":{"ProjectionType":"ALL"}}}]"
```

Menggunakan update-table akan menampilkan hasil sampel berikut.

```
{
  "TableDescription": {
    "TableArn": "arn:aws:dynamodb:us-west-2:111122223333:table/Music",
    "AttributeDefinitions": [
      {
        "AttributeName": "AlbumTitle",
        "AttributeType": "S"
      },
      {
        "AttributeName": "Artist",
        "AttributeType": "S"
      },
      {
        "AttributeName": "SongTitle",
        "AttributeType": "S"
      }
    ]
  }
}
```

```
    }
  ],
  "GlobalSecondaryIndexes": [
    {
      "IndexSizeBytes": 0,
      "IndexName": "AlbumTitle-index",
      "Projection": {
        "ProjectionType": "ALL"
      },
      "ProvisionedThroughput": {
        "NumberOfDecreasesToday": 0,
        "WriteCapacityUnits": 5,
        "ReadCapacityUnits": 10
      },
      "IndexStatus": "CREATING",
      "Backfilling": false,
      "KeySchema": [
        {
          "KeyType": "HASH",
          "AttributeName": "AlbumTitle"
        }
      ],
      "IndexArn": "arn:aws:dynamodb:us-west-2:111122223333:table/Music/index/AlbumTitle-index",
      "ItemCount": 0
    }
  ],
  "ProvisionedThroughput": {
    "NumberOfDecreasesToday": 0,
    "WriteCapacityUnits": 5,
    "ReadCapacityUnits": 10
  },
  "TableSizeBytes": 0,
  "TableName": "Music",
  "TableStatus": "UPDATING",
  "TableId": "a04b7240-0a46-435b-a231-b54091ab1017",
  "KeySchema": [
    {
      "KeyType": "HASH",
      "AttributeName": "Artist"
    },
    {
      "KeyType": "RANGE",
      "AttributeName": "SongTitle"
    }
  ]
}
```

```
    }
  ],
  "ItemCount": 0,
  "CreationDateTime": 1558028402.69
}
}
```

Perhatikan bahwa nilai bidang `IndexStatus` diatur ke `CREATING`.

Untuk memverifikasi bahwa DynamoDB selesai membuat indeks sekunder global `AlbumTitle-index`, gunakan perintah `describe-table`.

Linux

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

Windows CMD

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

Perintah ini akan menampilkan hasil berikut. Indeks siap digunakan saat nilai bidang `IndexStatus` yang ditampilkan diatur ke `ACTIVE`.

```
"IndexStatus": "ACTIVE",
```

Selanjutnya, Anda dapat mengkueri indeks sekunder global. Lihat detailnya di [Langkah 7: Kueri indeks sekunder global](#).

Langkah 7: Kueri indeks sekunder global

Pada langkah ini, Anda mengkueri indeks sekunder global pada tabel `Music` menggunakan konsol Amazon DynamoDB atau AWS CLI.

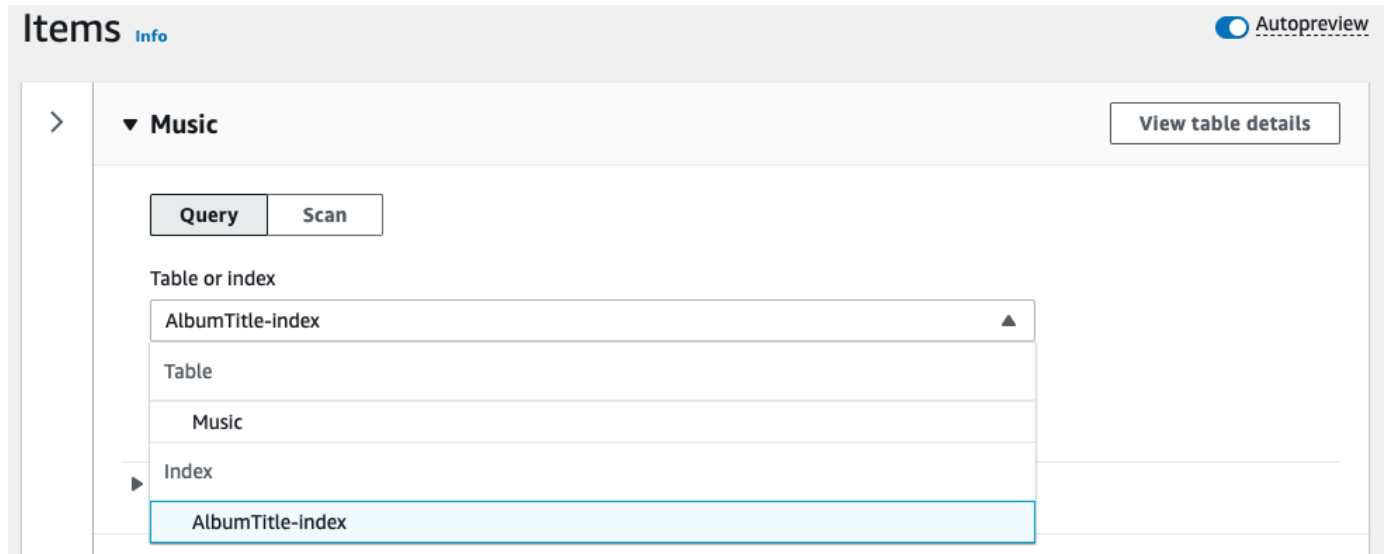
Untuk informasi selengkapnya tentang indeks sekunder global, lihat [Menggunakan Indeks Sekunder Global di DynamoDB](#).

AWS Management Console

Ikuti langkah-langkah ini untuk menggunakan konsol DynamoDB guna mengkueri data melalui indeks sekunder global `AlbumTitle-index`.

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tabel Musik dari daftar tabel.
4. Pilih Lihat item.
5. Pilih Kueri.
6. Dalam daftar drop-down di bawah Query, pilih AlbumTitle-index.

Untuk AlbumTitle, masukkan **Somewhat Famous**, lalu pilih Jalankan.



AWS CLI

Contoh berikut AWS CLI akan mengkueri indeks sekunder global AlbumTitle-index pada tabel Music. Anda dapat melakukan ini melalui API DynamoDB atau [PartiQL](#), bahasa kueri yang kompatibel dengan SQL untuk DynamoDB.

DynamoDB API

Anda akan mengkueri indeks sekunder global melalui API DynamoDB dengan menggunakan query dan memberikan nama indeks.

Linux

```
aws dynamodb query \  
  --table-name Music \  
  --index-name AlbumTitle-index \  
  --query 'SELECT * FROM AlbumTitle-index'
```

```
--key-condition-expression "AlbumTitle = :name" \  
--expression-attribute-values '{":name":{"S":"Somewhat Famous"}}'
```

Windows CMD

```
aws dynamodb query ^  
  --table-name Music ^  
  --index-name AlbumTitle-index ^  
  --key-condition-expression "AlbumTitle = :name" ^  
  --expression-attribute-values "{\":name\":{\":S\":\":Somewhat Famous\"}}"
```

Menggunakan query akan menampilkan hasil sampel berikut.

```
{  
  "Items": [  
    {  
      "AlbumTitle": {  
        "S": "Somewhat Famous"  
      },  
      "Awards": {  
        "S": "1"  
      },  
      "Artist": {  
        "S": "No One You Know"  
      },  
      "SongTitle": {  
        "S": "Call Me Today"  
      }  
    },  
    {  
      "AlbumTitle": {  
        "S": "Somewhat Famous"  
      },  
      "Awards": {  
        "N": "2"  
      },  
      "Artist": {  
        "S": "No One You Know"  
      },  
      "SongTitle": {  
        "S": "Howdy"  
      }  
    }  
  ]  
}
```



```
],  
  "Count": 2,  
  "ScannedCount": 2,  
  "ConsumedCapacity": null  
}
```

PartiQL for DynamoDB

Anda akan mengkueri indeks sekunder global melalui PartiQL dengan menggunakan pernyataan `Select` dan memberikan nama indeks.

Note

Anda harus menghindari tanda kutip ganda di sekitar `Music` dan `AlbumTitle-index` karena ini dilakukan melalui CLI.

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM \"Music\".\"AlbumTitle-  
index\" \  
                                     WHERE AlbumTitle='Somewhat Famous'"
```

Windows CMD

```
aws dynamodb execute-statement --statement "SELECT * FROM \"Music\".\"AlbumTitle-  
index\" WHERE AlbumTitle='Somewhat Famous'"
```

Menggunakan pernyataan `Select` dengan cara ini akan menampilkan hasil sampel berikut.

```
{  
  "Items": [  
    {  
      "AlbumTitle": {  
        "S": "Somewhat Famous"  
      },  
      "Awards": {  
        "S": "1"  
      },  
      "Artist": {  
        "S": "No One You Know"  
      }  
    }  
  ]  
}
```

```
    },
    "SongTitle": {
      "S": "Call Me Today"
    }
  },
  {
    "AlbumTitle": {
      "S": "Somewhat Famous"
    },
    "Awards": {
      "S": "2"
    },
    "Artist": {
      "S": "No One You Know"
    },
    "SongTitle": {
      "S": "Howdy"
    }
  }
]
}
```

Untuk informasi selengkapnya tentang mengkueri data dengan PartiQL, lihat [Pernyataan pilih PartiQL](#).

Langkah 8: (Opsional) hapus sumber daya

Jika tidak lagi memerlukan tabel Amazon DynamoDB yang dibuat untuk tutorial, Anda dapat menghapusnya. Langkah ini membantu memastikan bahwa Anda tidak akan dikenakan biaya untuk sumber daya yang tidak Anda gunakan. Anda dapat menggunakan konsol DynamoDB atau AWS CLI untuk menghapus tabel `Music` yang Anda buat di [Langkah 2: Buat tabel](#).

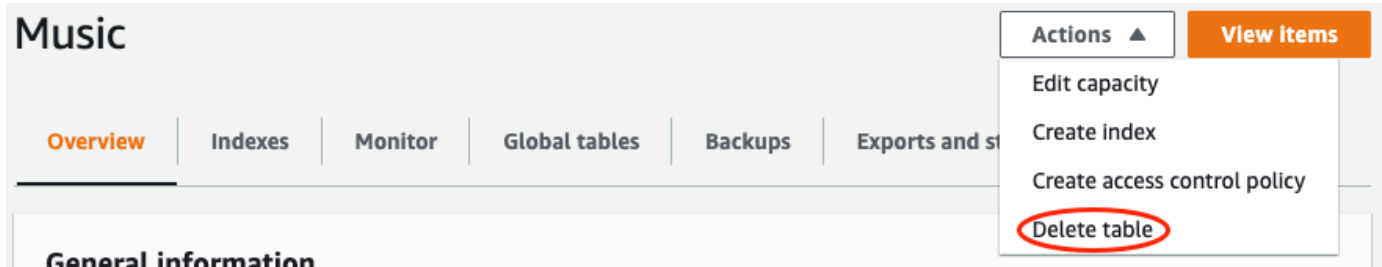
Untuk informasi selengkapnya tentang operasi tabel di DynamoDB, lihat [Bekerja dengan tabel dan data di DynamoDB](#).

AWS Management Console

Untuk menghapus tabel `Music` menggunakan konsol:

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi di sisi kiri konsol, pilih `Tabel`.

3. Pilih tabel Musik dari daftar tabel.
4. Pilih tab Indeks untuk tabel Music.
5. Pilih Hapus tabel.



AWS CLI

Contoh AWS CLI berikut akan menghapus tabel Music menggunakan delete-table.

```
aws dynamodb delete-table --table-name Music
```

Mulai menggunakan DynamoDB: Langkah selanjutnya

Untuk informasi selengkapnya tentang cara menggunakan Amazon DynamoDB, lihat topik berikut:

- [Bekerja dengan tabel dan data di DynamoDB](#)
- [Bekerja dengan item dan atribut](#)
- [Operasi kueri di DynamoDB](#)
- [Menggunakan Indeks Sekunder Global di DynamoDB](#)
- [Bekerja dengan transactions](#)
- [Akselerasi dalam memori dengan DynamoDB Accelerator \(DAX\)](#)
- [Memulai dengan DynamoDB dan SDK AWS](#)
- [Pemrograman dengan DynamoDB dan SDK AWS](#)

Memulai dengan DynamoDB dan SDK AWS

Gunakan tutorial langsung di bagian ini untuk memulai Amazon DynamoDB dan SDK. AWS Anda dapat menjalankan contoh kode pada versi DynamoDB yang dapat diunduh atau layanan web DynamoDB.

Topik

- [Membuat tabel DynamoDB](#)
- [Menulis item ke tabel DynamoDB](#)
- [Membaca item dari tabel DynamoDB](#)
- [Memperbarui item dalam tabel DynamoDB](#)
- [Menghapus item dalam tabel DynamoDB](#)
- [Mengkueri tabel DynamoDB](#)
- [Memindai tabel DynamoDB](#)
- [Menggunakan DynamoDB dengan SDK AWS](#)

Membuat tabel DynamoDB

Anda dapat membuat tabel menggunakan AWS Management Console, AWS CLI, atau AWS SDK. Untuk informasi selengkapnya tentang tabel ini, lihat [Komponen inti dari Amazon DynamoDB](#).

Membuat tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara membuat tabel DynamoDB menggunakan SDK AWS .

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
client, string tableName)
    {
        var response = await client.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,
                },
            },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 5,
```

```
        WriteCapacityUnits = 5,
    },
});

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

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

TableStatus status;

int sleepDuration = 2000;

do
{
    System.Threading.Thread.Sleep(sleepDuration);

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

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

return status == TableStatus.ACTIVE;
}
```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#####
# 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.
#   -p provisioned_throughput -- Provisioned throughput settings for the
#   table.
#
# Returns:
#   0 - If successful.
#   1 - If it fails.
#####
function dynamodb_create_table() {
    local table_name attribute_definitions key_schema provisioned_throughput
    response
    local option OPTARG # Required to use getopt command in a function.

#####
# Function usage explanation
#####
function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table."
    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 " -p provisioned_throughput -- Provisioned throughput settings for the
table."
    echo ""
}

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

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

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

response=$(aws dynamodb create-table \
  --table-name "$table_name" \
  --attribute-definitions file://"${attribute_definitions}" \
  --key-schema file://"${key_schema}" \
  --provisioned-throughput "$provisioned_throughput")

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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}
```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#!/ 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 outcome.IsSuccess();
}

```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Contoh 1: Untuk membuat tabel dengan tag

`create-table` Contoh berikut menggunakan atribut tertentu dan skema kunci untuk membuat tabel bernama `MusicCollection`. Tabel ini menggunakan throughput yang disediakan

dan dienkripsi saat istirahat menggunakan CMK yang dimiliki default. AWS Perintah ini juga menerapkan tag ke tabel, dengan kunci dari Owner dan nilai blueTeam.

```
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
```

Output:

```
{  
  "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"
  }
}
```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 2: Untuk membuat tabel dalam Mode On-Demand

Contoh berikut membuat tabel yang disebut MusicCollection menggunakan mode on-demand, bukan mode throughput yang disediakan. Ini berguna untuk tabel dengan beban kerja yang tidak terduga.

```
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
```

Output:

```
{
  "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"
}
}
}

```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 3: Untuk membuat tabel dan mengenkripsi dengan Customer Managed CMK

Contoh berikut membuat tabel bernama `MusicCollection` dan mengenkripsi menggunakan CMK yang dikelola pelanggan.

```

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

```

Output:

```
{
  "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"
    }
  }
}
```



```
}

```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 4: Untuk membuat tabel dengan Indeks Sekunder Lokal

Contoh berikut menggunakan atribut tertentu dan skema kunci untuk membuat tabel bernama `MusicCollection` dengan Indeks Sekunder Lokal bernama `AlbumTitleIndex`.

```
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 \
  --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\"]
        }
      }
    ]"
```

Output:

```
{
  "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"
    }
]
}
}

```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 5: Untuk membuat tabel dengan Indeks Sekunder Global

Contoh berikut membuat tabel bernama GameScores dengan Global Secondary Index disebut GameTitleIndex. Tabel dasar memiliki kunci partisi UserId dan kunci urutan GameTitle, sehingga Anda dapat menemukan skor terbaik pengguna individu untuk game tertentu secara efisien, sedangkan GSI memiliki kunci partisi GameTitle dan kunci urutan TopScore, memungkinkan Anda untuk cepat menemukan skor tertinggi secara keseluruhan untuk game tertentu.

```

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
    }
  }
]\"

```

Output:

```

{
  \"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"
  }
]
}
```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 6: Untuk membuat tabel dengan beberapa Indeks Sekunder Global sekaligus

Contoh berikut membuat tabel bernama GameScores dengan dua Global Secondary Indexes. Skema GSI diteruskan melalui file, bukan pada baris perintah.

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

Isi dari `gsi.json`:

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

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

Output:

```
{
  "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"
    }
]
}
```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 7: Untuk membuat tabel dengan Streams diaktifkan

Contoh berikut membuat tabel yang disebut GameScores dengan DynamoDB Streams diaktifkan. Baik gambar baru dan lama dari setiap item akan ditulis ke aliran.

```
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
```

Output:

```
{  
  "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"
  }
}

```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 8: Untuk membuat tabel dengan Keys-Only Stream diaktifkan

Contoh berikut membuat tabel yang disebut GameScores dengan DynamoDB Streams diaktifkan. Hanya atribut kunci dari item yang dimodifikasi yang ditulis ke aliran.

```

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

```

Output:

```

{
  "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
}
}
```

Untuk informasi selengkapnya, lihat [Mengubah pengambilan data untuk DynamoDB Streams di Panduan Pengembang Amazon DynamoDB](#).

Contoh 9: Untuk membuat tabel dengan kelas Standard Infrequent Access

Contoh berikut membuat tabel yang disebut GameScores dan menetapkan kelas tabel Standard-Infrequent Access (DynamoDB Standard-IA). Kelas tabel ini dioptimalkan untuk penyimpanan menjadi biaya dominan.

```
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
```

Output:

```
{  
  "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
  }
}
```

Untuk informasi selengkapnya, lihat [Kelas tabel](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 10: Membuat tabel dengan Hapus Perlindungan diaktifkan

Contoh berikut membuat tabel yang disebut GameScores dan memungkinkan perlindungan penghapusan.

```
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
```

Output:

```
{
  "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
}
}
```

Untuk informasi selengkapnya, lihat [Menggunakan perlindungan penghapusan](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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() (*types.TableDescription, error) {
    var tableDesc *types.TableDescription
    table, err := basics.DynamoDbClient.CreateTable(context.TODO(),
        &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),
            ProvisionedThroughput: &types.ProvisionedThroughput{
                ReadCapacityUnits:  aws.Int64(10),
                WriteCapacityUnits: aws.Int64(10),
            },
        })
    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(context.TODO(), &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
}
return tableDesc, err
}
```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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.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.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.
 *
 * For more information, see the following documentation topic:
 *
 */
```

```
* 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())
        .provisionedThroughput(ProvisionedThroughput.builder()
            .readCapacityUnits(10L)
            .writeCapacityUnits(10L)
            .build())
        .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 "";
}
}
```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

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

```
    return response;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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 provisionedVal = ProvisionedThroughput {
    readCapacityUnits = 10
    writeCapacityUnits = 10
}

val request = CreateTableRequest {
    attributeDefinitions = listOf(attDef)
    keySchema = listOf(keySchemaVal)
    provisionedThroughput = provisionedVal
    tableName = tableNameVal
}

DynamoDbClient { 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
}
}
```

- Untuk detail API, lihat [CreateTable](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat tabel.

```
$tableName = "ddb_demo_table_${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],
    ]);
}
```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat tabel untuk menyimpan data film.

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

    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
                    {"AttributeName": "title", "KeyType": "RANGE"}, # Sort key
                ],
                AttributeDefinitions=[
```

```
        {"AttributeName": "year", "AttributeType": "N"},
        {"AttributeName": "title", "AttributeType": "S"},
    ],
    ProvisionedThroughput={
        "ReadCapacityUnits": 10,
        "WriteCapacityUnits": 10,
    },
)
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
```

- Untuk detail API, lihat [CreateTable](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource
  attr_reader :table_name
  attr_reader :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"}
    ],
    provisioned_throughput: {read_capacity_units: 10, write_capacity_units:
10})
  @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
```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for Ruby API.

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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 pt = ProvisionedThroughput::builder()
        .read_capacity_units(10)
        .write_capacity_units(5)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .provisioned_throughput(pt)
```

```

        .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))
        }
    }
}

```

- Untuk detail API, lihat [CreateTable](#) referensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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.
```

- Untuk detail API, lihat [CreateTable](#) di AWS SDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
///
/// Create a movie table in the Amazon DynamoDB data store.
///
private func createTable() async throws {
    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),
        ],
        keySchema: [
            DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
            DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
        ],
        provisionedThroughput: DynamoDBClientTypes.ProvisionedThroughput(
            readCapacityUnits: 10,
            writeCapacityUnits: 10
        ),
        tableName: self.tableName
    )
    let output = try await client.createTable(input: input)
    if output.tableDescription == nil {
        throw MoviesError.TableNotFound
    }
}
```

- Untuk detail API, lihat referensi [CreateTableAWS](#) SDK untuk Swift API.

Untuk contoh DynamoDB lainnya, lihat [Contoh kode untuk DynamoDB menggunakan SDK AWS](#).

Menulis item ke tabel DynamoDB

Anda dapat menuliskan item pada tabel DynamoDB menggunakan AWS Management Console, AWS CLI, atau SDK AWS. Untuk informasi selengkapnya tentang item, lihat [Komponen inti dari Amazon DynamoDB](#).

Menulis item pada tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menulis item ke tabel DynamoDB menggunakan SDK AWS.

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <param name="newMovie">A Movie object containing information 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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
    Movie newMovie, string tableName)
    {
        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,
    };

    var response = await client.PutItemAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

#####
# 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 getopt 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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}
```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#!/ 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 outcome.IsSuccess();
}

```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Contoh 1: Untuk menambahkan item ke tabel

`put-item` Contoh berikut menambahkan item baru ke `MusicCollection` tabel.

```

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

```

Isi dari `item.json`:

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

Output:

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

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk menimpa item secara kondisional dalam tabel

`put-item` Contoh berikut menimpa item yang ada dalam `MusicCollection` tabel hanya jika item yang ada memiliki `AlbumTitle` atribut dengan nilai `Greatest Hits` Perintah mengembalikan nilai item sebelumnya.

```
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
```

Isi dari item.json:

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

Isi dari names.json:

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

Isi dari values.json:

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

Output:

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

Jika kunci sudah ada, Anda akan melihat output berikut:

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

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [PutItem](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(movie Movie) error {
    item, err := attributevalue.MarshalMap(movie)
    if err != nil {
        panic(err)
    }
    _, err = basics.DynamoDbClient.PutItem(context.TODO(), &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
}

// 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"])
}
```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menempatkan item ke dalam tabel menggunakan [DynamoDbClient](#).

```
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);
    }
}
```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [PutCommand](#).

```
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;
};
```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Masukkan item ke dalam tabel.

```
// 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);
  }
});
```

Masukkan item ke dalam tabel menggunakan klien dokumen DynamoDB.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

```

- Untuk detail API, lihat [PutItem](#) di AWSSDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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);
    }

```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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

    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
```

- Untuk detail API, lihat [PutItem](#) di AWSSDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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
end
```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for Ruby API.

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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,  
})  
}
```

- Untuk detail API, lihat [PutItem](#) referensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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.
```

- Untuk detail API, lihat [PutItem](#) di AWSSDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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 {
    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)
}

///
/// 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
}
```

- Untuk detail API, lihat referensi [PutItemAWSSDK](#) untuk Swift API.

Untuk contoh DynamoDB lainnya, lihat [Contoh kode untuk DynamoDB menggunakan SDK AWS](#).

Membaca item dari tabel DynamoDB

Anda dapat membaca item dari tabel DynamoDB menggunakan, AWS Management Console AWS CLI the, atau SDK. AWS Untuk informasi selengkapnya tentang item, lihat [Komponen inti dari Amazon DynamoDB](#).

Membaca item dari tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara membaca item dari tabel DynamoDB menggunakan SDK AWS .

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <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 static async Task<Dictionary<string, AttributeValue>>
    GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
    {
        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 client.GetItemAsync(request);
        return response.Item;
    }

```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

#####
# 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 getopt 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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#!/ 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();
}
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Contoh 1: Untuk membaca item dalam tabel

`get-item` Contoh berikut mengambil item dari `MusicCollection` tabel. Tabel memiliki kunci hash-and-range utama (`Artist` dan `SongTitle`), jadi Anda harus menentukan kedua atribut ini. Perintah tersebut juga meminta informasi tentang kapasitas baca yang dikonsumsi oleh operasi.

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

Isi dari `key.json`:

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

Output:

```
{
  "Item": {
    "AlbumTitle": {
      "S": "Songs About Life"
    },
    "SongTitle": {
      "S": "Happy Day"
    },
    "Artist": {
      "S": "Acme Band"
    }
  },
  "ConsumedCapacity": {
    "TableName": "MusicCollection",
    "CapacityUnits": 0.5
  }
}
```

Untuk informasi selengkapnya, lihat [Membaca Item](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk membaca item menggunakan pembacaan yang konsisten

Contoh berikut mengambil item dari MusicCollection tabel menggunakan pembacaan yang sangat konsisten.

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

Isi dari key.json:

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

Output:


```
{
  "Item": {
    "AlbumTitle": {
      "S": "Songs About Life"
    },
    "SongTitle": {
      "S": "Happy Day"
    },
    "Artist": {
      "S": "Acme Band"
    }
  },
  "ConsumedCapacity": {
    "TableName": "MusicCollection",
    "CapacityUnits": 1.0
  }
}
```

Untuk informasi selengkapnya, lihat [Membaca Item](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 3: Untuk mengambil atribut tertentu dari suatu item

Contoh berikut menggunakan ekspresi proyeksi untuk mengambil hanya tiga atribut dari item yang diinginkan.

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

Isi dari `names.json`:

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

Output:

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

Untuk informasi selengkapnya, lihat [Membaca Item](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [GetItem](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(title string, year int) (Movie, error) {
    movie := Movie{Title: title, Year: year}
    response, err := basics.DynamoDbClient.GetItem(context.TODO(),
        &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
}

// 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"])
}
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mendapat item dari tabel dengan menggunakan DynamoDbClient.

```
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);
}
}
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [GetCommand](#).

```
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;
};
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan item dari tabel.

```
// 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);
  }
});
```

Dapatkan item dari tabel menggunakan klien dokumen DynamoDB.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

- Untuk detail API, lihat [GetItem](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

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

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

    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"]
```

- Untuk detail API, lihat [GetItem](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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
end
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for Ruby API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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.
```

- Untuk detail API, lihat [GetItem](#) di AWS SDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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 {
    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
}
```

- Untuk detail API, lihat referensi [GetItemAWSSDK](#) untuk Swift API.

Untuk contoh DynamoDB lainnya, lihat [Contoh kode untuk DynamoDB menggunakan SDK AWS](#).

Memperbarui item dalam tabel DynamoDB

Anda dapat memperbarui item dari tabel DynamoDB menggunakan, AWS Management Console AWS CLI the, atau SDK. AWS Untuk informasi selengkapnya tentang item, lihat [Komponen inti dari Amazon DynamoDB](#).

Memperbarui item dalam tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara memperbarui item dalam tabel DynamoDB menggunakan SDK AWS .

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <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 static async Task<bool> UpdateItemAsync(
        AmazonDynamoDBClient client,
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
```

```
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,
};


var response = await client.UpdateItemAsync(request);

return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#####
# 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 getopt 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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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  
}
```

```
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#!/ 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 outcome.IsSuccess();
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Contoh 1: Untuk memperbarui item dalam tabel

Contoh `update-item` berikut memperbarui item dalam tabel `MusicCollection`. Ia menambahkan atribut baru (`Year`) dan memodifikasi `AlbumTitle` atribut. Semua atribut dalam item, seperti yang muncul setelah pembaruan, dikembalikan sebagai respons.

```
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
```

Isi dari `key.json`:

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

Isi dari `expression-attribute-names.json`:

```
{  
  "#Y": "Year", "#AT": "AlbumTitle"  
}
```

Isi dari `expression-attribute-values.json`:

```
{  
  ":y": {"N": "2015"},  
  ":t": {"S": "Louder Than Ever"}  
}
```

Output:

```
{
  "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
  ]
}
```

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk memperbarui item secara kondisional

Contoh berikut memperbarui item dalam `MusicCollection` tabel, tetapi hanya jika item yang ada belum memiliki `Year` atribut.


```
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)"
```

Isi dari `key.json`:

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

Isi dari `expression-attribute-names.json`:

```
{  
  "#Y": "Year",  
  "#AT": "AlbumTitle"  
}
```

Isi dari `expression-attribute-values.json`:

```
{  
  ":y": {"N": "2015"},  
  ":t": {"S": "Louder Than Ever"}  
}
```

Jika item sudah memiliki `Year` atribut, DynamoDB mengembalikan output berikut.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem  
operation: The conditional request failed
```

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(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(context.TODO(),
            &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
}

// 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"])
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Memperbarui item dalam tabel menggunakan [DynamoDbClient](#).

```
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!");
}
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [UpdateCommand](#).

```
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;
};
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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 { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- Untuk detail API, lihat [UpdateItem](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
    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
            ]
        ],
    ]);
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Perbarui item dengan menggunakan ekspresi pembaruan.

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

    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"]
```

Perbarui item dengan menggunakan ekspresi pembaruan yang menyertakan operasi aritmatika.

```
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"]
```

Perbarui item hanya jika memenuhi persyaratan tertentu.

```
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"]

```

- Untuk detail API, lihat [UpdateItem](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for Ruby API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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.
```

- Untuk detail API, lihat [UpdateItem](#) di AWS SDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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]? {
    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: String = "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
}
```

- Untuk detail API, lihat referensi [UpdateItemAWSSDK](#) untuk Swift API.

Untuk contoh DynamoDB lainnya, lihat [Contoh kode untuk DynamoDB menggunakan SDK AWS](#).

Menghapus item dalam tabel DynamoDB

Anda dapat menghapus item dari tabel DynamoDB menggunakan AWS Management Console, AWS CLI, atau SDK AWS. Untuk informasi selengkapnya tentang item, lihat [Komponen inti dari Amazon DynamoDB](#).

Menghapus item dalam tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menghapus item dalam tabel DynamoDB menggunakan SDK AWS.

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// <summary>
/// Deletes a single item from a DynamoDB table.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <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 static async Task<bool> DeleteItemAsync(
    AmazonDynamoDBClient client,
    string tableName,
    Movie movieToDelete)
{
    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,
    };

    var response = await client.DeleteItemAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

#####
# 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 getopt 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 getopt "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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}
```

- Untuk detail API, lihat [Deleteltemdi Referensi AWS CLI Perintah](#).

C++

SDK for C++

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#!/ 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 outcome.IsSuccess();  
}
```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Contoh 1: Untuk menghapus item

`delete-item` Contoh berikut menghapus item dari `MusicCollection` tabel dan meminta rincian tentang item yang telah dihapus dan kapasitas yang digunakan oleh permintaan.

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

Isi dari `key.json`:

```
{  
  "Artist": {"S": "No One You Know"},  
  "SongTitle": {"S": "Scared of My Shadow"}  
}
```

Output:

```
{  
  "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
    ]
  }
}

```

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk menghapus item secara kondisional

Contoh berikut menghapus item dari ProductCatalog tabel hanya jika salah satu Sporting Goods atau Gardening Supplies dan harganya antara 500 dan 600. ProductCategory la mengembalikan rincian tentang item yang telah dihapus.

```

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

```

Isi dari names.json:

```
{
```



```
"#P": "Price"
}
```

Isi dari `values.json`:

```
{
  ":cat1": {"S": "Sporting Goods"},
  ":cat2": {"S": "Gardening Supplies"},
  ":lo": {"N": "500"},
  ":hi": {"N": "600"}
}
```

Output:

```
{
  "Attributes": {
    "Id": {
      "N": "456"
    },
    "Price": {
      "N": "550"
    },
    "ProductCategory": {
      "S": "Sporting Goods"
    }
  }
}
```

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [Deleteltem](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(movie Movie) error {
    _, err := basics.DynamoDbClient.DeleteItem(context.TODO(),
        &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
}

// 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"])
}

```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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);
}
}
```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [DeleteCommand](#).

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus item dalam tabel.

```
// 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);
  }
});
```

```
}  
});
```

Hapus item dari tabel menggunakan klien dokumen DynamoDB.

```
// 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);  
  }  
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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 { region = "us-east-1" }.use { ddb ->
        ddb.deleteItem(request)
        println("Item with key matching $keyVal was deleted")
    }
}
```

- Untuk detail API, lihat [DeleteItem](#) di AWSSDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
$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,
    ]);
}
```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

    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
```

Anda dapat menentukan kondisi sehingga item dihapus hanya ketika memenuhi kriteria tertentu.

```
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
```

- Untuk detail API, lihat [DeleteItem](#) di AWSSDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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.
end
```

```
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
```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for Ruby API.

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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)),
  }
}
```

- Untuk detail API, lihat [DeleteItem](#) referensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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.
```

- Untuk detail API, lihat [DeleteItem](#) di AWSSDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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 {
    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)
}
```

- Untuk detail API, lihat referensi [DeleteItemAWSSDK](#) untuk Swift API.

Untuk contoh DynamoDB lainnya, lihat [Contoh kode untuk DynamoDB menggunakan SDK AWS](#).

Mengkueri tabel DynamoDB

Anda dapat melakukan kueri pada tabel DynamoDB menggunakan AWS Management Console, SDK, atau AWS CLI SDK. AWS Untuk informasi selengkapnya tentang kueri, lihat [Operasi kueri di DynamoDB](#).

Mengkueri tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara mengkueri tabel DynamoDB menggunakan SDK AWS .

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repository Contoh Kode AWS](#).

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient
client, string tableName, int year)
{
    var movieTable = Table.LoadTable(client, tableName);
    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;

    Search 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;
}

```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for .NET .

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

#####
# 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 getopt 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 getopt "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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}
```

- Untuk detail API, lihat [Query](#) in Referensi Perintah AWS CLI .

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
//! 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;
}
```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for C++ .

CLI

AWS CLI

Contoh 1: Untuk menanyakan tabel

queryContoh berikut query item dalam MusicCollection tabel. Tabel memiliki kunci hash-and-range utama (ArtistdanSongTitle), tetapi kueri ini hanya menentukan nilai kunci hash. Ini mengembalikan judul lagu oleh artis bernama “No One You Know”.

```
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
```

Isi dari expression-attributes.json:

```
{  
  ":v1": {"S": "No One You Know"}  
}
```

Output:

```
{  
  "Items": [  
    {  
      "SongTitle": {  
        "S": "Call Me Today"  
      },  
      "SongTitle": {  
        "S": "Scared of My Shadow"  
      }  
    }  
  ]  
}
```

```

    }
  ],
  "Count": 2,
  "ScannedCount": 2,
  "ConsumedCapacity": {
    "TableName": "MusicCollection",
    "CapacityUnits": 0.5
  }
}

```

Untuk informasi selengkapnya, lihat [Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

Contoh 2: Untuk menanyakan tabel menggunakan pembacaan yang sangat konsisten dan melintasi indeks dalam urutan menurun

Contoh berikut melakukan kueri yang sama dengan contoh pertama, tetapi mengembalikan hasil dalam urutan terbalik dan menggunakan pembacaan yang sangat konsisten.

```

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

```

Isi dari `expression-attributes.json`:

```

{
  ":v1": {"S": "No One You Know"}
}

```

Output:

```

{
  "Items": [
    {
      "SongTitle": {
        "S": "Scared of My Shadow"
      }
    }
  ]
}

```



```

    }
  },
  {
    "SongTitle": {
      "S": "Call Me Today"
    }
  }
],
"Count": 2,
"ScannedCount": 2,
"ConsumedCapacity": {
  "TableName": "MusicCollection",
  "CapacityUnits": 1.0
}
}

```

Untuk informasi selengkapnya, lihat [Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

Contoh 3: Untuk menyaring hasil tertentu

Contoh berikut query MusicCollection tetapi mengecualikan hasil dengan nilai-nilai tertentu dalam atribut. AlbumTitle Perhatikan bahwa ini tidak mempengaruhi ScannedCount atau ConsumedCapacity, karena filter diterapkan setelah item telah dibaca.

```

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

```

Isi dari values.json:

```

{
  ":v1": {"S": "No One You Know"},
  ":v2": {"S": "Blue Sky Blues"},
  ":v3": {"S": "Greatest Hits"}
}

```

Isi dari names.json:

```
{
  "#n1": "Artist",
  "#n2": "AlbumTitle"
}
```

Output:

```
{
  "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
  }
}
```

Untuk informasi selengkapnya, lihat [Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

Contoh 4: Untuk mengambil hanya jumlah item

Contoh berikut mengambil hitungan item yang cocok dengan query, tetapi tidak mengambil salah satu item itu sendiri.

```
aws dynamodb query \
  --table-name MusicCollection \
  --select COUNT \
  --key-condition-expression "Artist = :v1" \
  --expression-attribute-values file://expression-attributes.json
```

Isi dari `expression-attributes.json`:

```
{
  ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
  "Count": 2,
  "ScannedCount": 2,
  "ConsumedCapacity": null
}
```

Untuk informasi selengkapnya, lihat [Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

Contoh 5: Untuk menanyakan indeks

Contoh berikut query indeks `AlbumTitleIndex` sekunder lokal. Query mengembalikan semua atribut dari tabel dasar yang telah diproyeksikan ke indeks sekunder lokal. Perhatikan bahwa saat menanyakan indeks sekunder lokal atau indeks sekunder global, Anda juga harus memberikan nama tabel dasar menggunakan `table-name` parameter.

```
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
```

Isi dari `expression-attributes.json`:

```
{
  ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
```

```
"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
    }
  }
}
```

Untuk informasi selengkapnya, lihat [Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

- Untuk detail API, lihat [Query](#) in Referensi Perintah AWS CLI .

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(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(context.TODO())
        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
}

// 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"])
}
```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for Go .

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kueri tabel dengan menggunakan [DynamoDbClient](#).

```
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;
    }
}
```

Mengkueri tabel menggunakan DynamoDbClient dan indeks sekunder.

```
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);
        }
    }
}
```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [QueryCommand](#).

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for JavaScript .

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for JavaScript .

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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 { region = "us-east-1" }.use { ddb ->
    val response = ddb.query(request)
    return response.count
}
}

```

- Untuk detail API, lihat [Query](#) di Referensi API SDK for Kotlin AWS .

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

$birthKey = [
    'Key' => [
        'year' => [
            'N' => "$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);
    }

```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for PHP .

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kueri item menggunakan ekspresi kondisi kunci.

```

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

    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"]

```

Kueri dan proyeksikan item untuk mengembalikan subset data.

```

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"]
```

- Untuk detail API, lihat [Query](#) di Referensi API SDK for Python (Boto3) AWS .

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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
  end
end
```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for Ruby .

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Temukan film yang dibuat pada tahun tertentu.


```
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![])
    }
}
```

- Untuk detail API, lihat [Query](#) di Referensi API SDK for Rust AWS .

SAP ABAP

SDK untuk SAP ABAP

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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.
  ENDLLOOP.
  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.

```

- Untuk detail API, lihat [Query](#) di Referensi API SDK for SAP ABAP AWS .

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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] {
    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
    )
    let output = try await client.query(input: input)

    guard let items = output.items else {
        throw MoviesError.ItemNotFound
    }
}
```

```
    }

    // Convert the found movies into `Movie` objects and return an array
    // of them.

    var movieList: [Movie] = []
    for item in items {
        let movie = try Movie(withItem: item)
        movieList.append(movie)
    }
    return movieList
}
```

- Untuk detail API, lihat [Query](#) di Referensi API SDK for Swift AWS .

Untuk contoh DynamoDB lainnya, lihat [Contoh kode untuk DynamoDB menggunakan SDK AWS](#).

Memindai tabel DynamoDB

Anda dapat melakukan pemindaian pada tabel DynamoDB menggunakan, AWS Management Console AWS CLI the, atau SDK. AWS Untuk informasi selengkapnya tentang pemindaian, lihat [Bekerja dengan pemindaian di DynamoDB](#).

Memindai tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara memindai tabel DynamoDB menggunakan SDK AWS .

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
public static async Task<int> ScanTableAsync(
```

```
        AmazonDynamoDBClient client,
        string tableName,
        int startYear,
        int endYear)
    {
        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 client.ScanAsync(request);
            foundCount += response.Items.Count;
            response.Items.ForEach(i => DisplayItem(i));
            request.ExclusiveStartKey = response.LastEvaluatedKey;
        }
        while (response.LastEvaluatedKey.Count > 0);
        return foundCount;
    }
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for .NET .

Bash

AWS CLI dengan skrip Bash

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#####
# 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 getopt 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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```
#####
```

```
# 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
}
```

- Untuk detail API, lihat [Scan](#) in Referensi Perintah AWS CLI .

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
//! 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);

    // Perform scan on table.
    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();
```

```
    if (!items.empty()) {
        std::cout << "Number of items retrieved from scan: " << items.size()
                << std::endl;
        // Iterate each item and print.
        for (const Aws::Map<Aws::String,
            Aws::DynamoDB::Model::AttributeValue> &itemMap: 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 item found in table:" << tableName << std::endl;
    }
}
else {
    std::cerr << "Failed to Scan items: " << outcome.GetError().GetMessage()
                << std::endl;
}

return outcome.IsSuccess();
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for C++ .

CLI

AWS CLI

Untuk memindai tabel

scanContoh berikut memindai seluruh MusicCollection tabel, dan kemudian mempersempit hasilnya menjadi lagu-lagu oleh artis “No One You Know”. Untuk setiap item, hanya judul album dan judul lagu yang dikembalikan.

```
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
```

Isi dari `expression-attribute-names.json`:

```
{  
  "#ST": "SongTitle",  
  "#AT": "AlbumTitle"  
}
```

Isi dari `expression-attribute-values.json`:

```
{  
  ":a": {"S": "No One You Know"}  
}
```

Output:

```
{  
  "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  
}
```

Untuk informasi selengkapnya, lihat [Bekerja dengan Pemindaian di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

- Untuk detail API, lihat [Scan](#) in Referensi Perintah AWS CLI .

Go

SDK for Go V2

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(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(context.TODO())
    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
}

// 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"])
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for Go .

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Memindai tabel Amazon [DynamoDbClient](#)DynamoDB menggunakan.

```
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);
    }
}
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [ScanCommand](#).

```
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;
};
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for JavaScript .

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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 + ")"
      );
    });
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for JavaScript .

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
suspend fun scanItems(tableNameVal: String) {
    val request = ScanRequest {
        tableName = tableNameVal
    }

    DynamoDbClient { 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]}")
            }
        }
    }
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API SDK for Kotlin AWS .

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
$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);
    }

```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for PHP .

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

    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
```

- Untuk detail API, lihat [Scan](#) di Referensi API SDK for Python (Boto3) AWS .

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for Ruby .

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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(())
}

```

- Untuk detail API, lihat [Scan](#) di Referensi API SDK for Rust AWS .

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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.
```

- Untuk detail API, lihat [Scan](#) di Referensi API SDK for SAP ABAP AWS .

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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] {
    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 output = try await client.scan(input: input)

    guard let items = output.items else {
        return movieList
    }

    // Build an array of `Movie` objects for the returned items.

    for item in items {
        let movie = try Movie(withItem: item)
        movieList.append(movie)
    }

    // Call this function recursively to continue collecting matching
    // movies, if necessary.

    if output.lastEvaluatedKey != nil {
```

```
        let movies = try await self.getMovies(firstYear: firstYear, lastYear:
lastYear,
                                startKey: output.lastEvaluatedKey)
        movieList += movies
    }
    return movieList
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API SDK for Swift AWS .

Untuk contoh DynamoDB lainnya, lihat [Contoh kode untuk DynamoDB menggunakan SDK AWS](#).

Menggunakan DynamoDB dengan SDK AWS

AWSKit pengembangan perangkat lunak (SDK) tersedia untuk banyak bahasa pemrograman populer. Setiap SDK menyediakan API, contoh kode, dan dokumentasi yang memudahkan pengembang untuk membangun aplikasi dalam bahasa pilihan mereka.

Dokumentasi SDK	Contoh kode
AWS SDK for C++	AWS SDK for C++contoh kode
AWS SDK for Go	AWS SDK for Gocontoh kode
AWS SDK for Java	AWS SDK for Javacontoh kode
AWS SDK for JavaScript	AWS SDK for JavaScriptcontoh kode
AWS SDK for Kotlin	AWS SDK for Kotlincontoh kode
AWS SDK for .NET	AWS SDK for .NETcontoh kode
AWS SDK for PHP	AWS SDK for PHPcontoh kode
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3)contoh kode
AWS SDK for Ruby	AWS SDK for Rubycontoh kode
AWS SDK for Rust	AWS SDK for Rustcontoh kode

Dokumentasi SDK	Contoh kode
AWS SDK untuk SAP ABAP	AWS SDK untuk SAP ABAP contoh kode
AWS SDK for Swift	AWS SDK for Swift contoh kode

Untuk contoh khusus untuk DynamoDB, lihat. [Contoh kode untuk DynamoDB menggunakan SDK AWS](#)

 **Ketersediaan contoh**

Tidak dapat menemukan apa yang Anda butuhkan? Minta contoh kode dengan menggunakan tautan Berikan umpan balik di bagian bawah halaman ini.

Pemrograman dengan DynamoDB dan SDK AWS

Bagian ini mencakup topik terkait developer. Jika Anda ingin menjalankan contoh kode, lihat [Menjalankan contoh kode dalam Panduan Pengembang ini](#).

Note

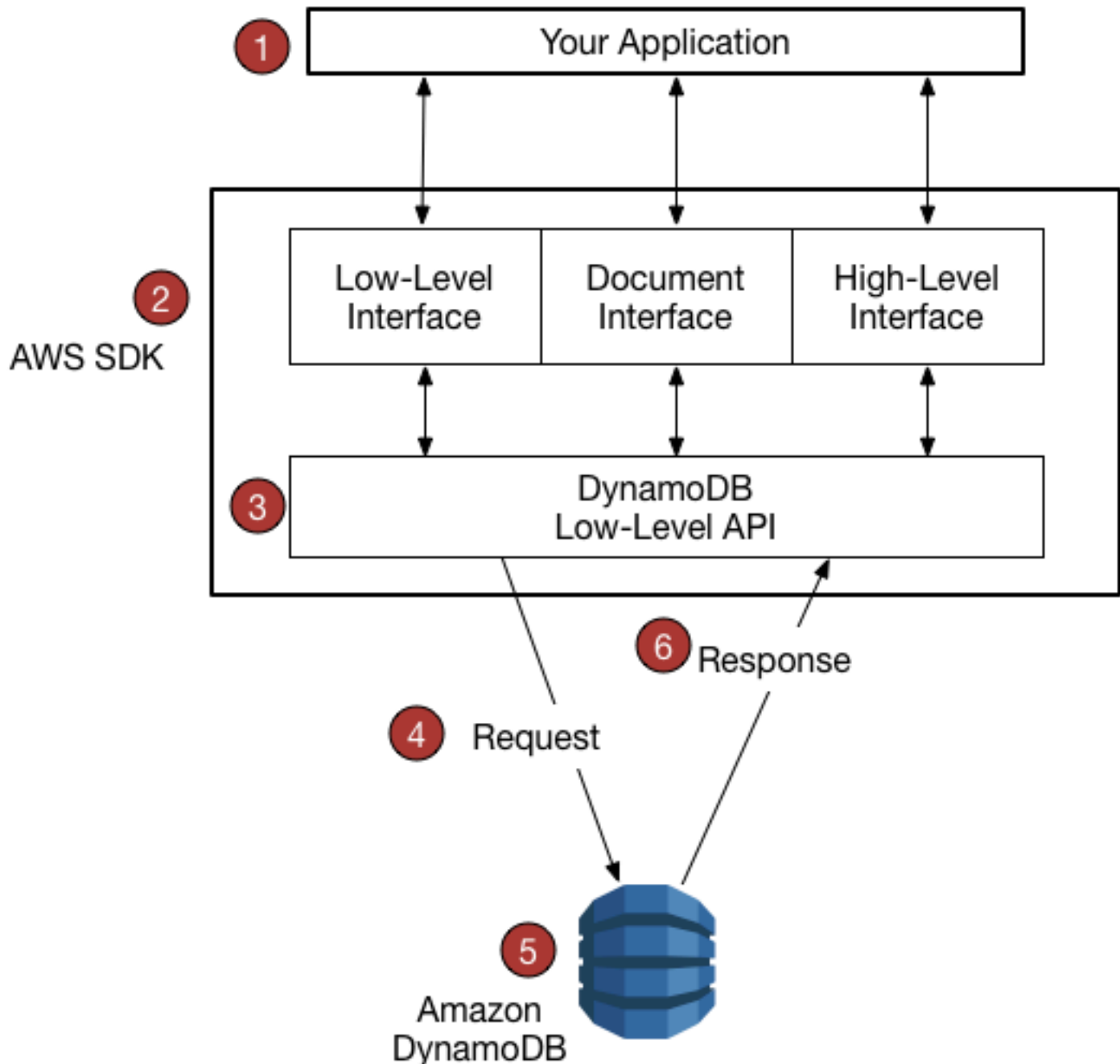
Pada bulan Desember 2017, AWS memulai proses migrasi semua titik akhir Amazon DynamoDB untuk menggunakan sertifikat aman yang dikeluarkan oleh Amazon Trust Services (ATS). Untuk informasi selengkapnya, lihat [Memecahkan masalah pembentukan koneksi SSL/TLS](#).

Topik

- [Ikhtisar dukungan AWS SDK untuk DynamoDB](#)
- [Antarmuka pemrograman tingkat tinggi untuk DynamoDB](#)
- [Menjalankan contoh kode dalam Panduan Pengembang ini](#)
- [Pemrograman Amazon DynamoDB dengan Python dan Boto3](#)
- [Pemrograman Amazon DynamoDB dengan JavaScript](#)
- [Pemrograman Amazon DynamoDB dengan AWS SDK for Java 2.x](#)

Ikhtisar dukungan AWS SDK untuk DynamoDB

Diagram berikut memberikan gambaran tingkat tinggi pemrograman aplikasi Amazon DynamoDB menggunakan SDK. AWS




1. Anda menulis aplikasi menggunakan AWS SDK untuk bahasa pemrograman Anda.
2. Setiap AWS SDK menyediakan satu atau lebih antarmuka terprogram untuk bekerja dengan DynamoDB. Antarmuka spesifik yang tersedia bergantung pada bahasa pemrograman dan AWS SDK yang Anda gunakan. Pilihannya meliputi:
 - [Antarmuka tingkat rendah](#)
 - [Antarmuka dokumen](#)
 - [Antarmuka persistensi objek](#)

- [Antarmuka Tingkat Tinggi](#)
3. AWS SDK membuat permintaan HTTP (S) untuk digunakan dengan API DynamoDB tingkat rendah.
 4. AWS SDK mengirimkan permintaan ke titik akhir DynamoDB.
 5. DynamoDB menjalankan permintaan. Jika permintaan berhasil, DynamoDB mengembalikan kode respons HTTP 200 (OK). Jika permintaan tidak berhasil, DynamoDB mengembalikan kode galat dan pesan kesalahan HTTP.
 6. AWS SDK memproses respons dan menyebarkannya kembali ke aplikasi Anda.

Masing-masing AWS SDK menyediakan layanan penting untuk aplikasi Anda, termasuk yang berikut ini:

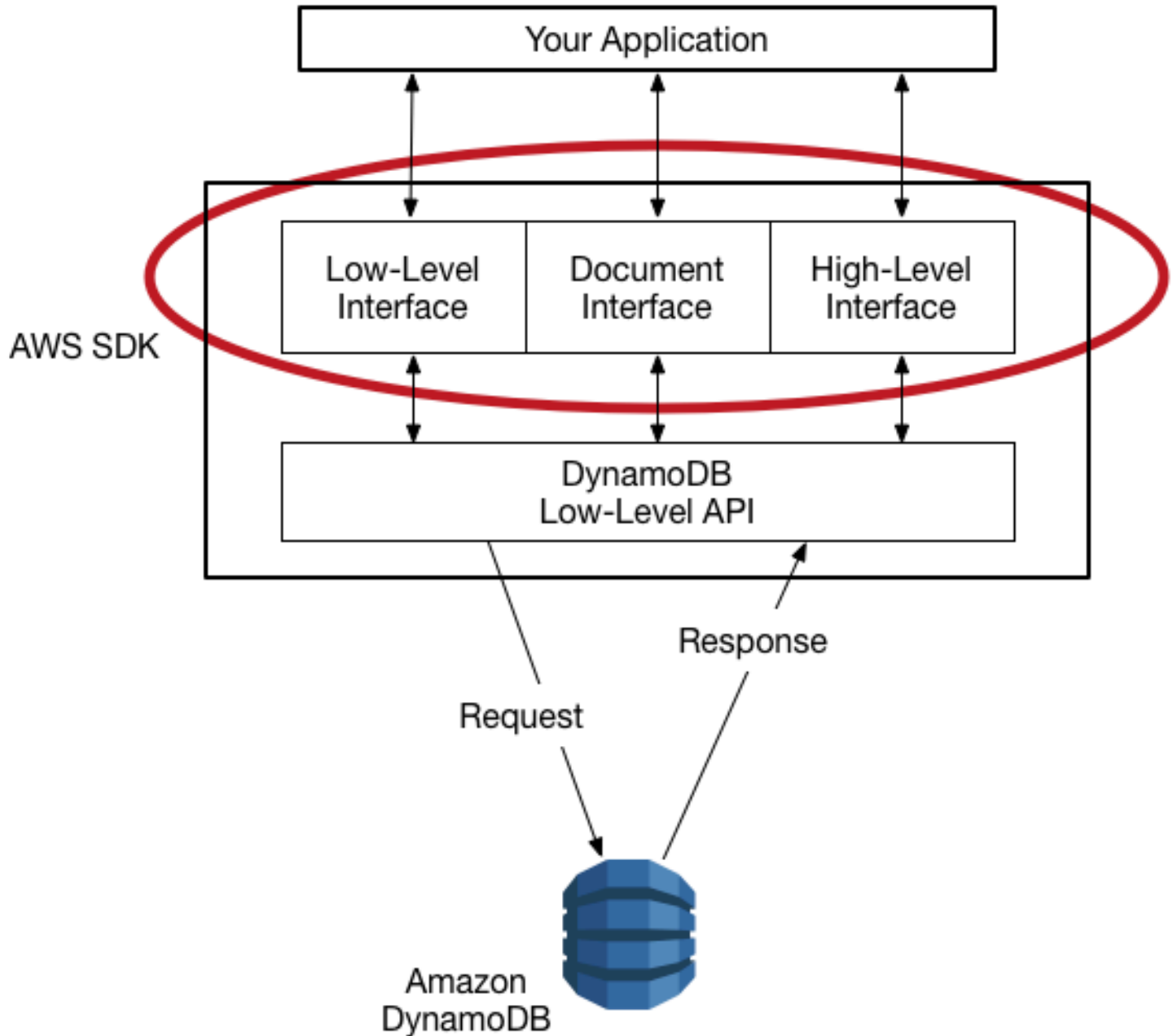
- Memformat permintaan HTTP(S) dan menyerialisasi parameter permintaan.
- Menghasilkan tanda tangan kriptografi untuk setiap permintaan.
- Meneruskan permintaan ke titik akhir DynamoDB dan menerima respons dari DynamoDB.
- Mengekstraksi hasil dari respons tersebut.
- Menerapkan logika coba lagi dasar dalam kasus kesalahan.

Anda tidak perlu menulis kode untuk semua tugas ini.

 Note

Untuk informasi selengkapnya tentang AWS SDK, termasuk petunjuk penginstalan dan dokumentasi, lihat [Alat untuk Amazon Web Services](#).

Antarmuka terprogram



Setiap [AWS SDK](#) menyediakan satu atau lebih antarmuka program untuk bekerja dengan Amazon DynamoDB. Antarmuka ini memiliki rentang dari pembungkus DynamoDB level rendah sederhana hingga lapisan persistensi berorientasi objek. Antarmuka yang tersedia bervariasi tergantung pada AWS SDK dan bahasa pemrograman yang Anda gunakan.

Bagian berikut menyoroti beberapa antarmuka yang tersedia, menggunakan AWS SDK for Java sebagai contoh. (Tidak semua antarmuka tersedia di semua AWS SDK.)

Topik

- [Antarmuka tingkat rendah](#)
- [Antarmuka dokumen](#)
- [Antarmuka persistensi objek](#)

Antarmuka tingkat rendah

Setiap AWS SDK khusus bahasa menyediakan antarmuka tingkat rendah untuk Amazon DynamoDB, dengan metode yang sangat mirip dengan permintaan API DynamoDB tingkat rendah.

Dalam beberapa kasus, Anda akan perlu untuk mengidentifikasi tipe data dari atribut-atribut tersebut menggunakan [Deskriptor tipe data](#), seperti S untuk string, atau N untuk angka.

Note

Antarmuka tingkat rendah tersedia dalam setiap AWS SDK khusus bahasa.

Program Java berikut menggunakan antarmuka AWS SDK for Java tingkat rendah.

```
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);
    }
}
```

Antarmuka dokumen

Banyak AWS SDK menyediakan antarmuka dokumen, memungkinkan Anda untuk melakukan operasi bidang data (membuat, membaca, memperbarui, menghapus) pada tabel dan indeks. Dengan antarmuka dokumen, Anda tidak perlu menentukan [Deskriptor tipe data](#). Jenis data yang tersirat oleh semantik data itu sendiri. AWS SDK ini juga menyediakan metode untuk dengan mudah mengonversi dokumen JSON ke dan dari tipe data Amazon DynamoDB asli.

Note

Antarmuka dokumen tersedia di AWS SDK untuk [Java](#), [.NET](#), [Node.js](#), dan [JavaScript di browser](#).

Program Java berikut menggunakan antarmuka dokumen AWS SDK for Java. Program ini menciptakan objek `Table` yang merepresentasikan tabel `Music`, lalu meminta objek tersebut

untuk menggunakan `GetItem` guna mengambil lagu. Program ini selanjutnya mencetak tahun lagu tersebut dirilis.

Kelas `com.amazonaws.services.dynamodbv2.document.DynamoDB` mengimplementasikan antarmuka dokumen DynamoDB. Perhatikan bagaimana DynamoDB berfungsi sebagai pembungkus di sekitar klien tingkat rendah (`AmazonDynamoDB`).

```
package com.amazonaws.codesamples.gsg;

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.GetItemOutcome;
import com.amazonaws.services.dynamodbv2.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);

    }
}
```

Antarmuka persistensi objek

Beberapa AWS SDK menyediakan antarmuka persistensi objek di mana Anda tidak melakukan operasi bidang data secara langsung. Sebagai gantinya, Anda akan membuat objek yang mewakili item dalam tabel dan indeks Amazon DynamoDB, dan hanya berinteraksi dengan objek tersebut. Ini memungkinkan Anda menulis kode yang berfokus pada objek, bukan kode yang berfokus pada basis data.

 Note

Antarmuka persistensi objek tersedia di AWS SDK untuk Java dan .NET. Untuk informasi selengkapnya, lihat [Antarmuka pemrograman tingkat tinggi untuk DynamoDB](#) DynamoDB.

```
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();

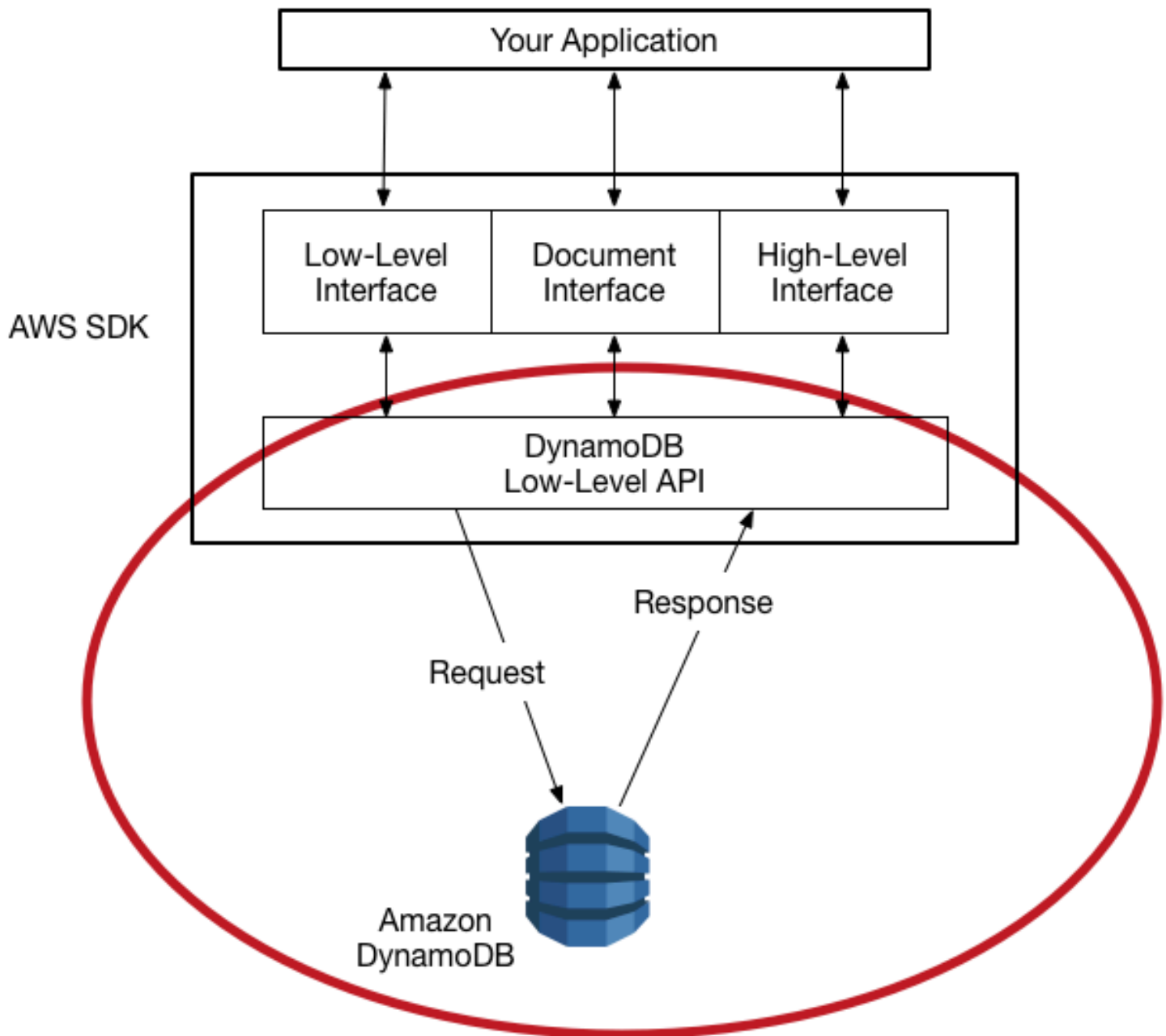
            // Get the item by using the key.
            result = table.getItem(
                (GetItemEnhancedRequest.Builder requestBuilder) ->
                requestBuilder.key(key));
            System.out.println("***** The description value is " +
                result.getCustName());

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

API tingkat rendah DynamoDB

Topik

- [Format permintaan](#)
- [Format respons](#)
- [Deskriptor tipe data](#)
- [Data numerik](#)
- [Data biner](#)



API tingkat rendah Amazon DynamoDB adalah antarmuka tingkat protokol untuk DynamoDB. Pada tingkat ini, setiap permintaan HTTP(S) harus diformat dengan benar dan membawa tanda tangan digital yang valid.

AWS SDK membuat permintaan API DynamoDB tingkat rendah atas nama Anda dan memproses respons dari DynamoDB. Ini memungkinkan Anda berfokus pada logika aplikasi Anda, bukan detail tingkat rendah. Namun, Anda masih bisa mendapatkan keuntungan dari pengetahuan dasar tentang bagaimana DynamoDB API tingkat rendah bekerja.

Untuk informasi selengkapnya tentang DynamoDB API, lihat [Referensi Amazon DynamoDB API](#).

Note

DynamoDB Streams memiliki API tingkat rendah sendiri, yang terpisah dari DynamoDB dan didukung penuh oleh SDK. AWS

Untuk informasi selengkapnya, lihat [Ubah tangkapan data DynamoDB Streams](#). Untuk DynamoDB Streams API tingkat rendah, lihat [Referensi Amazon DynamoDB Streams API](#).

DynamoDB API tingkat rendah JavaScript menggunakan Object Notation (JSON) sebagai format protokol kawat. JSON menyajikan data dalam hierarki sehingga nilai data dan struktur data disampaikan secara bersamaan. Pasangan nama-nilai ditentukan dalam format `name:value`. Hierarki data ditentukan oleh tanda kurung berisi pasangan nama-nilai.

DynamoDB menggunakan JSON hanya sebagai protokol transportasi, bukan sebagai format penyimpanan. AWS SDK menggunakan JSON untuk mengirim data ke DynamoDB, dan DynamoDB merespons dengan JSON. DynamoDB tidak menyimpan data terus-menerus dalam format JSON.

Note

Untuk informasi selengkapnya tentang JSON, lihat [Memperkenalkan JSON](#) pada situs web [JSON.org](#).

Format permintaan

API tingkat rendah DynamoDB menerima permintaan POST HTTP(S) sebagai masukan. AWS SDK membangun permintaan ini untuk Anda.

Misalkan Anda memiliki tabel bernama `Pets`, dengan skema kunci yang terdiri dari `AnimalType` (kunci partisi) dan `Name` (kunci urutan). Kedua atribut ini bertipe `string`. Untuk mengambil item dari `Pets`, AWS SDK membuat permintaan berikut.

```
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"}
  }
}
```

Perhatikan hal berikut tentang permintaan ini:

- Header `Authorization` berisi informasi yang diperlukan untuk DynamoDB guna mengautentikasi permintaan. Untuk informasi selengkapnya, lihat [Menandatangani permintaan AWS API](#) dan [proses penandatanganan Versi Tanda Tangan 4](#) di bagian Referensi Umum Amazon Web Services.
- Header `X-Amz-Target` berisi nama operasi DynamoDB: `GetItem`. (Ini juga disertai dengan versi API tingkat rendah, dalam hal ini `20120810`.)
- Muatan (isi) permintaan mengandung parameter untuk operasi, dalam format JSON. Untuk operasi `GetItem`, parameternya adalah `TableName` dan `Key`.

Format respons

Setelah menerima permintaan, DynamoDB memprosesnya dan mengembalikan respons. Untuk permintaan yang ditunjukkan sebelumnya, muatan respons HTTP(S) berisi hasil dari operasi, seperti yang ditunjukkan pada contoh berikut.

```
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"}
}
}
```

Pada titik ini, AWS SDK mengembalikan data respons ke aplikasi Anda untuk diproses lebih lanjut.

Note

Jika DynamoDB tidak dapat memproses permintaan, DynamoDB akan mengembalikan kode kesalahan HTTP dan pesan. AWS SDK menyebarkannya untuk aplikasi Anda dalam bentuk pengecualian. Untuk informasi selengkapnya, lihat [Penanganan kesalahan dengan DynamoDB](#).


Deskriptor tipe data

Protokol DynamoDB API tingkat rendah membutuhkan setiap atribut untuk disertai dengan deskriptor jenis data. Deskriptor tipe data adalah token yang memberi tahu DynamoDB cara menafsirkan setiap atribut.

Contoh di [Format permintaan](#) dan [Format respons](#) menunjukkan contoh bagaimana deskriptor tipe data digunakan. Permintaan `GetItem` menentukan S untuk atribut skema kunci Pets (`AnimalType` dan `Name`), yang bertipe `string`. Respons `GetItem` berisi item Pets dengan atribut jenis `string` (S), `number` (N), `map` (M), dan `list` (L).

Berikut ini adalah daftar lengkap deskriptor tipe daya DynamoDB:

- **S** – String
- **N** – Nomor
- **B** – Biner
- **BOOL** – Boolean
- **NULL** – Null
- **M** – Peta
- **L** – Daftar
- **SS** – Set String
- **NS** – Set Nomor
- **BS** – Set Biner

 Note

Untuk deskripsi mendetail tentang tipe data DynamoDB, lihat [Tipe Data](#).

Data numerik

Bahasa pemrograman yang berbeda menawarkan tingkat dukungan yang berbeda untuk JSON. Dalam beberapa kasus, Anda mungkin memutuskan untuk menggunakan pustaka pihak ketiga untuk memvalidasi dan menguraikan dokumen JSON.

Beberapa pustaka pihak ketiga membangun jenis angka JSON, menyediakan jenisnya sendiri seperti `int`, `long`, atau `double`. Namun, jenis data angka asli di DynamoDB tidak memetakan persis untuk jenis data lainnya, sehingga perbedaan jenis ini dapat menyebabkan konflik. Selain itu, banyak pustaka JSON tidak menangani nilai numerik presisi tetap, dan secara otomatis menyimpulkan jenis data ganda untuk urutan digit yang berisi titik desimal.

Untuk mengatasi masalah ini, DynamoDB menyediakan jenis numerik tunggal tanpa kehilangan data. Untuk menghindari konversi implisit yang tidak diinginkan ke nilai ganda, DynamoDB menggunakan string untuk transfer data nilai numerik. Pendekatan ini memberikan fleksibilitas untuk memperbarui nilai atribut sekaligus mempertahankan semantik pengurutan yang tepat, seperti menempatkan nilai "01", "2", dan "03" dalam urutan yang tepat.

Jika presisi angka penting untuk aplikasi, Anda harus mengonversi nilai numerik untuk string tersebut sebelum meneruskannya ke DynamoDB.

Data biner

DynamoDB mendukung atribut binari. Namun, JSON tidak secara native mendukung data biner pengodean. Untuk mengirim data biner dalam permintaan, Anda perlu untuk mengodekannya dalam format base64. Setelah menerima permintaan, DynamoDB mengodekan data base64 kembali ke biner.

Skema pengodean base64 yang digunakan oleh DynamoDB dijelaskan di [RFC 4648](#) di situs web Internet Engineering Task Force (IETF).

Penanganan kesalahan dengan DynamoDB

Bagian ini menjelaskan kesalahan runtime dan cara menanganinya. Kode dan pesan kesalahan yang khusus untuk Amazon DynamoDB juga dijelaskan dalam bagian ini. Untuk daftar kesalahan umum yang berlaku untuk semua layanan AWS, lihat [Manajemen Akses](#)

Topik

- [Komponen kesalahan](#)
- [Kesalahan transaksional](#)
- [Pesan dan kode kesalahan](#)
- [Penanganan kesalahan dalam aplikasi Anda](#)
- [Percobaan ulang kesalahan dan mundur eksponensial](#)
- [Operasi batch dan penanganan kesalahan](#)

Komponen kesalahan

Ketika program Anda mengirimkan permintaan, DynamoDB mencoba untuk memprosesnya. Jika permintaan berhasil, DynamoDB akan menghasilkan kode status sukses HTTP (200 OK), bersama dengan hasil dari operasi yang diminta.

Jika permintaan tidak berhasil, DynamoDB menampilkan kesalahan. Setiap kesalahan memiliki tiga komponen:

- Kode status HTTP (seperti 400).

- Nama pengecualian (seperti `ResourceNotFoundException`).
- Pesan kesalahan (seperti `Requested resource not found: Table: tablename not found`).

SDK AWS menangani kesalahan penyebaran ke aplikasi Anda sehingga Anda dapat mengambil tindakan yang tepat. Misalnya, dalam program Java, Anda bisa menulis logika try-catch untuk menangani `ResourceNotFoundException`.

Jika tidak menggunakan SDK AWS, Anda perlu mengurai konten respons tingkat rendah dari DynamoDB. Berikut ini adalah contoh respons tersebut.

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: LDM6CJP8RMQ1FHKSC1RBVJFPNVV4KQNS05AEMF66Q9ASUAAJG
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"}
```

Kesalahan transaksional

Untuk informasi tentang kesalahan transaksional, lihat [Penanganan konflik transaksi di DynamoDB](#)

Pesan dan kode kesalahan

Berikut ini adalah daftar pengecualian yang dikembalikan oleh DynamoDB, dikelompokkan berdasarkan kode status HTTP. Jika Boleh diulang? adalah Ya, Anda dapat mengirimkan permintaan yang sama lagi. Jika Boleh diulang? adalah Tidak, Anda harus memperbaiki masalah di sisi klien sebelum mengirimkan permintaan baru.

Kode status HTTP 400

Kode status `400` HTTP menunjukkan adanya masalah dengan permintaan Anda, seperti kegagalan autentikasi, parameter yang diperlukan tidak ada, atau melebihi throughput yang disediakan tabel. Anda harus memperbaiki masalah di aplikasi Anda sebelum mengirimkan permintaan lagi.

`AccessDeniedException`

Pesan: Akses ditolak.

Klien salah menandatangani permintaan. Jika Anda menggunakan SDK AWS, permintaan ditandatangani untuk Anda secara otomatis; jika tidak, buka [Proses penandatanganan Tanda Tangan versi 4](#) di Referensi Umum AWS.

Boleh diulang? Tidak

ConditionalCheckFailedException

Pesan: Permintaan bersyarat gagal.

Anda menentukan syarat yang dievaluasi ke false. Misalnya, Anda mungkin telah mencoba melakukan pembaruan bersyarat pada suatu item, tetapi nilai atribut sebenarnya tidak cocok dengan nilai yang diharapkan dalam kondisi yang dihadapi.

Boleh diulang? Tidak

IncompleteSignatureException

Pesan: Tanda tangan permintaan tidak sesuai dengan standar AWS.

Tanda tangan permintaan tidak menyertakan semua komponen yang diperlukan. Jika Anda menggunakan SDK AWS, permintaan ditandatangani untuk Anda secara otomatis; jika tidak, buka [Proses penandatanganan Tanda Tangan Versi 4](#) di Referensi Umum AWS.

Boleh diulang? Tidak

ItemCollectionSizeLimitExceededException

Pesan: Ukuran koleksi terlampaui.

Untuk tabel dengan indeks sekunder lokal, sekelompok item dengan nilai kunci partisi yang sama telah melampaui batas ukuran maksimum 10 GB. Untuk informasi selengkapnya tentang kumpulan item, lihat [Koleksi item dalam Indeks Sekunder Lokal](#).

Boleh diulang? Ya

LimitExceededException

Pesan: Terlalu banyak operasi untuk pelanggan tertentu.

Terlalu banyak operasi bidang kontrol secara bersamaan. Jumlah kumulatif tabel dan indeks dalam CREATING, DELETING, atau keadaan UPDATING tidak boleh melebihi 500.

Boleh diulang? Ya

MissingAuthenticationTokenException

Pesan: Permintaan harus berisi ID Kunci Akses AWS yang valid (terdaftar).

Permintaan tidak menyertakan header otorisasi yang diperlukan, atau formatnya salah. Lihat [API tingkat rendah DynamoDB](#).

Boleh diulang? Tidak

ProvisionedThroughputExceededException

Pesan: Anda melampaui throughput maksimum yang diizinkan untuk sebuah tabel atau untuk satu atau beberapa indeks sekunder global. [Untuk melihat metrik kinerja untuk throughput yang disediakan vs. throughput yang dikonsumsi, buka konsol Amazon CloudWatch](#)

Contoh: Tingkat permintaan Anda terlalu tinggi. SDK AWS untuk DynamoDB otomatis mengulangi permintaan yang menerima pengecualian ini. Permintaan Anda akhirnya berhasil, kecuali antrean percobaan ulang Anda terlalu besar untuk diselesaikan. Kurangi frekuensi permintaan menggunakan [Percobaan ulang kesalahan dan mundur eksponensial](#).

Boleh diulang? Ya

RequestLimitExceeded

Pesan: Throughput melebihi batas throughput saat ini untuk akun Anda. Untuk meminta kenaikan tingkat batas, hubungi Dukungan AWS di <https://aws.amazon.com/support>.

Contoh: Tingkat permintaan sesuai permintaan melebihi throughput akun yang diizinkan dan tabel tidak dapat diskalakan lebih lanjut.

Boleh diulang? Ya

ResourceInUseException

Pesan: Sumber daya yang Anda coba ubah sedang digunakan.

Contoh: Anda mencoba untuk membuat ulang tabel yang sudah ada, atau menghapus tabel yang saat ini dalam status CREATING.

Boleh diulang? Tidak

ResourceNotFoundException

Pesan: Sumber daya yang diminta tidak ditemukan.

Contoh: Tabel yang diminta tidak ada, atau terlalu dini dalam status CREATING.

Boleh diulang? Tidak

ThrottlingException

Pesan: Tingkat permintaan melebihi throughput yang diizinkan.

Pengecualian ini dikembalikan sebagai `AmazonServiceException` respons dengan kode status `THROTTLING_EXCEPTION`. Pengecualian ini mungkin dikembalikan jika Anda melakukan operasi API [bidang kontrol](#) terlalu cepat.

Untuk tabel yang menggunakan mode sesuai permintaan, pengecualian ini mungkin dikembalikan untuk setiap operasi API [bidang data](#) jika tingkat permintaan Anda terlalu tinggi. Untuk mempelajari selengkapnya tentang penskalaan sesuai permintaan, lihat [Properti lalu lintas puncak dan penskalaan](#)

Boleh diulang? Ya

UnrecognizedClientException

Pesan: ID Kunci Akses atau token keamanan tidak valid.

Tanda tangan permintaan tidak benar. Penyebab yang paling mungkin adalah kunci rahasia atau ID kunci akses AWS tidak valid.

Boleh diulang? Ya

ValidationException

Pesan: Bervariasi, tergantung pada kesalahan tertentu yang dialami

Kesalahan ini dapat terjadi karena beberapa alasan, seperti parameter yang diperlukan tidak ada, nilai di luar rentang, atau jenis data tidak cocok. kesalahan berisi detail tentang bagian tertentu dari permintaan yang menyebabkan kesalahan.

Boleh diulang? Tidak

Kode status HTTP 5xx

Kode status 5xx HTTP menunjukkan masalah yang harus diselesaikan oleh AWS. Kemungkinan ini adalah kesalahan sementara, sehingga Anda dapat mencoba lagi permintaan Anda hingga berhasil. Jika tidak, masuk ke [Service Health Dashboard AWS](#) untuk melihat apakah ada masalah operasional dengan layanan ini.

Untuk informasi selengkapnya, lihat [Bagaimana cara mengatasi kesalahan HTTP 5xx di Amazon DynamoDB?](#)

InternalServerError (HTTP 500)

DynamoDB tidak dapat memproses permintaan Anda.

Boleh diulang? Ya

Note

Anda mungkin mengalami kesalahan server internal saat menggunakan item. Ini diduga terjadi selama masa aktif tabel. Setiap permintaan yang gagal dapat segera dicoba kembali.

Ketika Anda menerima kode status 500 pada operasi tulis, operasi tersebut mungkin berhasil atau gagal. Jika operasi tulis adalah permintaan `TransactWriteItem`, maka boleh saja mencoba ulang operasi tersebut. Jika operasi tulis adalah permintaan tulis item tunggal seperti `PutItem`, `UpdateItem`, atau `DeleteItem`, maka aplikasi Anda akan membaca status item sebelum mencoba kembali operasi tersebut, dan/atau menggunakan [Ekspresi kondisi](#) untuk memastikan item tetap dalam status yang benar setelah mencoba ulang, terlepas dari apakah operasi sebelumnya berhasil atau gagal. Jika idempotensi adalah persyaratan untuk operasi tulis, gunakan [TransactWriteItem](#), yang mendukung permintaan idempoten dengan menentukan `ClientRequestToken` secara otomatis untuk membedakan beberapa upaya untuk melakukan tindakan yang sama.

ServiceUnavailable (HTTP 503)

DynamoDB saat ini tidak tersedia. (Ini harus merupakan status sementara.)

Boleh diulang? Ya

Penanganan kesalahan dalam aplikasi Anda

Agar aplikasi Anda berjalan lancar, Anda perlu menambahkan logika untuk menangkap dan merespons kesalahan. Pendekatan umum termasuk menggunakan blok `try-catch` atau pernyataan `if-then`.

SDK AWS melakukan percobaan ulang dan pemeriksaan kesalahannya sendiri. Jika Anda mengalami kesalahan saat menggunakan salah satu SDK AWS, kode kesalahan dan deskripsi dapat membantu mengatasinya.

Anda juga akan melihat `Request ID` dalam respons. `Request ID` akan membantu jika Anda perlu bekerja dengan Dukungan AWS untuk mendiagnosis suatu masalah.

Percobaan ulang kesalahan dan mundur eksponensial

Banyak komponen di jaringan, seperti server DNS, sakelar, penyeimbang beban, dan lainnya, dapat menghasilkan kesalahan di mana pun selama permintaan tertentu. Teknik biasa untuk menangani respons kesalahan ini dalam lingkungan jaringan adalah dengan menerapkan percobaan ulang dalam aplikasi klien. Teknik ini meningkatkan keandalan aplikasi.

Setiap SDK AWS mengimplementasikan logika percobaan ulang secara otomatis. Anda dapat mengubah parameter percobaan ulang sesuai kebutuhan Anda. Misalnya, pertimbangkan aplikasi Java yang memerlukan strategi gagal cepat (`fail fast`), dengan tidak memperbolehkan percobaan ulang jika terjadi kesalahan. Dengan AWS SDK for Java, Anda dapat menggunakan kelas `ClientConfiguration` dan menyediakan nilai `maxErrorRetry 0` untuk menonaktifkan percobaan ulang. Untuk informasi selengkapnya, lihat dokumentasi SDK AWS untuk bahasa pemrograman Anda.

Jika tidak menggunakan SDK AWS, Anda harus mencoba ulang permintaan asli yang menerima kesalahan server (`5xx`). Namun, kesalahan klien (`4xx`, selain `ThrottlingException` atau `ProvisionedThroughputExceededException`) menunjukkan bahwa Anda perlu merevisi permintaan itu sendiri untuk memperbaiki masalah sebelum melakukan percobaan ulang.

Selain percobaan ulang sederhana, masing-masing SDK AWS mengimplementasikan algoritma mundur eksponensial untuk kontrol aliran yang lebih baik. Konsep di balik mundur eksponensial adalah menggunakan waktu tunggu yang lebih lama di antara percobaan ulang untuk respon kesalahan berturut-turut. Sebagai contoh, hingga 50 milidetik sebelum percobaan ulang pertama, hingga 100 milidetik sebelum percobaan ulang kedua, hingga 200 milidetik sebelum percobaan ulang ketiga, dan seterusnya. Namun, setelah satu menit, jika permintaan tidak berhasil, masalahnya mungkin pada ukuran permintaan yang melebihi throughput yang disediakan, dan bukan pada tingkat permintaan. Tetapkan jumlah maksimum percobaan ulang untuk berhenti sekitar satu menit. Jika permintaan tidak berhasil, selidiki opsi throughput yang disediakan.

Note

SDK AWS menerapkan logika percobaan ulang otomatis dan mundur eksponensial.

Kebanyakan algoritma mundur eksponensial menggunakan jitter (penundaan acak) untuk mencegah tabrakan berturut-turut. Karena Anda tidak berusaha menghindari tabrakan seperti itu dalam kasus ini, Anda tidak perlu menggunakan nomor acak ini. Namun, jika Anda menggunakan klien secara bersamaan, jitter dapat membantu permintaan Anda berhasil lebih cepat. Untuk informasi selengkapnya, lihat postingan blog tentang [Mundur eksponensial dan jitter](#).

Operasi batch dan penanganan kesalahan

API tingkat rendah DynamoDB mendukung operasi batch untuk baca dan tulis. `BatchGetItem` membaca item dari satu atau beberapa tabel, dan `BatchWriteItem` memasukkan atau menghapus item dalam satu atau beberapa tabel. Operasi batch ini diimplementasikan sebagai pembungkus di sekitar operasi DynamoDB non-batch lainnya. Dengan kata lain, `BatchGetItem` menginvokasi `GetItem` sekali untuk setiap item dalam batch. Demikian pula, `BatchWriteItem` menginvokasi `DeleteItem` atau `PutItem`, jika sesuai, untuk setiap item dalam batch.

Operasi batch dapat menoleransi kegagalan permintaan individual dalam batch. Misalnya, pertimbangkan `BatchGetItem` untuk membaca lima item. Meskipun beberapa permintaan `GetItem` yang mendasarinya gagal, ini tidak menyebabkan seluruh operasi `BatchGetItem` gagal. Namun, jika kelima operasi baca gagal, maka seluruh `BatchGetItem` akan gagal pula.

Operasi batch mengembalikan informasi tentang masing-masing permintaan yang gagal sehingga Anda dapat mendiagnosis masalah dan mencoba kembali operasi tersebut. Untuk `BatchGetItem`, tabel dan kunci primer yang dimaksud dikembalikan dalam nilai `UnprocessedKeys` respons. Untuk `BatchWriteItem`, informasi serupa dikembalikan dalam `UnprocessedItems`.

Kemungkinan besar penyebab gagalnya baca atau tulis adalah throttling. Untuk `BatchGetItem`, satu atau beberapa tabel dalam permintaan batch tidak memiliki kapasitas baca yang memadai untuk mendukung operasi. Untuk `BatchWriteItem`, satu atau beberapa tabel tidak memiliki kapasitas tulis yang memadai.

Jika DynamoDB mengembalikan item yang belum diproses, Anda harus mencoba ulang operasi batch pada item tersebut. Namun, kami sangat merekomendasikan agar Anda menggunakan algoritma mundur eksponensial. Jika Anda segera mencoba ulang operasi batch, permintaan baca atau tulis yang mendasarinya masih bisa gagal karena throttling pada masing-masing tabel. Jika Anda menunda operasi batch menggunakan mundur eksponensial, masing-masing permintaan dalam batch kemungkinan besar akan berhasil.

Antarmuka pemrograman tingkat tinggi untuk DynamoDB

AWS SDK menyediakan aplikasi dengan antarmuka tingkat rendah untuk bekerja dengan Amazon DynamoDB. Metode dan kelas sisi klien ini berkaitan langsung dengan API DynamoDB tingkat rendah. Namun, banyak developer mengalami putusnya koneksi, atau ketidakcocokan impedansi, ketika mereka perlu memetakan jenis data yang kompleks untuk item dalam tabel basis data. Dengan antarmuka basis data tingkat rendah, developer harus menulis metode untuk membaca atau menulis data objek ke tabel basis data, dan sebaliknya. Jumlah kode tambahan yang diperlukan untuk setiap kombinasi jenis objek dan tabel basis data dapat terlihat sangat banyak.

Untuk menyederhanakan pengembangan, AWS SDK untuk Java dan .NET menyediakan antarmuka tambahan dengan tingkat abstraksi yang lebih tinggi. Antarmuka tingkat yang lebih tinggi untuk DynamoDB memungkinkan Anda menentukan hubungan di antara objek dalam program Anda dan tabel basis data yang menyimpan data objek tersebut. Setelah menentukan pemetaan ini, Anda memanggil metode objek sederhana seperti `save`, `load`, atau `delete`, dan operasi DynamoDB tingkat rendah yang mendasarinya akan dipanggil secara otomatis atas nama Anda. Ini memungkinkan Anda menulis kode yang berfokus pada objek, bukan kode yang berfokus pada basis data.

Antarmuka pemrograman tingkat tinggi untuk DynamoDB tersedia di SDK untuk Java dan .NET. AWS

Java

- [Java 1.x: DynamoDBMapper](#)
- [Java 2.x: DynamoDB Ditingkatkan Klien](#)

.NET

- [.NET: Model dokumen](#)
- [.NET: Model persistensi objek](#)

Java 1.x: DynamoDBMapper

AWS SDK for Java Ini menyediakan `DynamoDBMapper` kelas, memungkinkan Anda untuk memetakan kelas sisi klien Anda ke tabel Amazon DynamoDB. Untuk menggunakan `DynamoDBMapper`, tentukan hubungan antara item dalam tabel DynamoDB dan instans objek yang sesuai dalam kode Anda. Kelas `DynamoDBMapper` memungkinkan Anda melakukan berbagai operasi buat, baca, perbarui, dan hapus (CRUD) pada item, dan menjalankan kueri serta memindai tabel.

Topik

- [Jenis data yang didukung untuk DynamoDB Mapper untuk Java](#)
- [Anotasi Java untuk DynamoDB](#)
- [Kelas DynamoDBMapper](#)
- [Pengaturan konfigurasi opsional untuk DynamoDBMapper](#)
- [Penguncian Optimis dengan nomor versi](#)
- [Memetakan data arbitrer](#)
- [Contoh DynamoDBMapper](#)

Note

Kelas `DynamoDBMapper` tidak memungkinkan Anda membuat, memperbarui, atau menghapus tabel. Untuk melakukan tugas tersebut, gunakan antarmuka SDK for Java tingkat rendah. Untuk informasi selengkapnya, lihat [Bekerja dengan tabel DynamoDB di Java](#).

SDK for Java menyediakan serangkaian jenis anotasi agar Anda dapat memetakan kelas ke tabel. Misalnya, pertimbangkan tabel `ProductCatalog` yang memiliki `Id` sebagai kunci partisi.

```
ProductCatalog(Id, ...)
```

Anda dapat memetakan kelas di aplikasi klien ke tabel ProductCatalog seperti yang ditunjukkan dalam kode Java berikut. Kode ini menentukan plain old Java object (POJO) bernama CatalogItem, yang menggunakan anotasi untuk memetakan bidang objek ke nama atribut DynamoDB.

Example

```
package com.amazonaws.codesamples;

import java.util.Set;

import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIgnore;
import com.amazonaws.services.dynamodbv2.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; }
}
```

Dalam kode sebelumnya, anotasi `@DynamoDBTable` memetakan kelas `CatalogItem` ke tabel `ProductCatalog`. Anda dapat menyimpan masing-masing instans kelas sebagai item dalam tabel. Dalam definisi kelas, anotasi `@DynamoDBHashKey` memetakan properti `Id` ke kunci primer.

Secara default, properti kelas memetakan atribut nama yang sama dalam tabel. Properti `Title` dan `ISBN` memetakan atribut nama yang sama di tabel.

Anotasi `@DynamoDBAttribute` bersifat opsional saat nama atribut DynamoDB cocok dengan nama properti yang dinyatakan di kelas. Jika namanya berbeda, gunakan anotasi ini dengan parameter `attributeName` untuk menentukan atribut DynamoDB yang sesuai dengan properti ini.

Dalam contoh sebelumnya, anotasi `@DynamoDBAttribute` ditambahkan ke setiap properti untuk memastikan bahwa nama properti cocok persis dengan tabel yang dibuat di [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#), dan konsisten dengan nama atribut yang digunakan dalam contoh kode lain dalam panduan ini.

Definisi kelas Anda dapat memiliki properti yang tidak dipetakan ke setiap atribut dalam tabel. Anda mengidentifikasi properti ini dengan menambahkan anotasi `@DynamoDBIgnore`. Dalam contoh sebelumnya, properti `SomeProp` ditandai dengan anotasi `@DynamoDBIgnore`. Saat Anda mengunggah instans `CatalogItem` ke tabel, instans `DynamoDBMapper` tidak menyertakan properti `SomeProp`. Selain itu, pemeta tidak mengembalikan atribut ini ketika Anda mengambil item dari tabel.

Setelah menentukan kelas pemetaan, Anda dapat menggunakan metode `DynamoDBMapper` untuk menulis instans kelas tersebut untuk item yang sesuai dalam tabel `Catalog`. Contoh kode berikut mendemonstrasikan teknik ini.

```
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);
```

Contoh kode berikut menunjukkan cara mengambil item dan mengakses beberapa atributnya.

```
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());
}
```

DynamoDBMapper menawarkan cara natural yang intuitif untuk menggunakan data DynamoDB dalam Java. Ini juga menyediakan beberapa fitur bawaan, seperti penguncian optimis, transaksi ACID, nilai kunci urutan dan kunci partisi yang dibuat secara otomatis, serta objek versioning.


Jenis data yang didukung untuk DynamoDB Mapper untuk Java

Bagian ini menjelaskan jenis data Java primitif, koleksi, dan jenis data arbitrer yang didukung di Amazon DynamoDB.

Amazon DynamoDB mendukung jenis data Java primitif dan kelas pembungkus primitif berikut.

- String
- Boolean, boolean
- Byte, byte
- Date (sebagai string [ISO_8601](#) millisecond-precision, yang beralih ke UTC)
- Calendar (sebagai string [ISO_8601](#) millisecond-precision, yang beralih ke UTC)
- Long, long
- Integer, int
- Double, double
- Float, float
- BigDecimal

- BigInteger

 Note

- Untuk informasi selengkapnya tentang aturan penamaan DynamoDB dan berbagai jenis data yang didukung, lihat [Jenis data dan aturan penamaan yang didukung di Amazon DynamoDB](#).
- Nilai Biner kosong didukung oleh DynamoDBMapper.
- Nilai String kosong didukung oleh AWS SDK for Java 2.x.

Dalam AWS SDK for Java 1.x, DynamoDBMapper mendukung pembacaan nilai atribut String kosong, namun, itu tidak akan menulis nilai atribut String kosong karena atribut ini dijatuhkan dari permintaan.

DynamoDB mendukung jenis koleksi [Kumpulan](#), [Daftar](#), dan [Peta](#) Java. Tabel berikut merangkum bagaimana jenis Java ini dipetakan untuk jenis DynamoDB.

Jenis Java	Jenis DynamoDB
Semua jenis angka	N (jenis angka)
String	S (jenis string)
Boolean	BOOL (jenis Boolean), 0 atau 1.
ByteBuffer	B (jenis biner)
Tanggal	S (jenis string). Nilai data disimpan sebagai string berformat ISO-8601.
Jenis koleksi Kumpulan	Jenis SS (kumpulan biner), jenis NS (kumpulan angka), atau jenis BS (kumpulan biner)

Antarmuka `DynamoDBTypeConverter` memungkinkan Anda memetakan jenis data arbitrer Anda sendiri ke jenis data yang secara asli didukung oleh DynamoDB. Untuk informasi selengkapnya, lihat [Memetakan data arbitrer](#).

Anotasi Java untuk DynamoDB

Bagian ini menjelaskan anotasi yang tersedia untuk memetakan kelas dan properti ke tabel dan atribut di Amazon DynamoDB.

Untuk dokumentasi Javadoc yang sesuai, lihat [Ringkasan Jenis Anotasi](#) di [Referensi API AWS SDK for Java](#).

Note

Dalam anotasi berikut, hanya `DynamoDBTable` dan `DynamoDBHashKey` yang diperlukan.

Topik

- [DynamoDBAttribute](#)
- [DynamoDB AutoGeneratedKey](#)
- [DynamoDB AutoGeneratedTimestamp](#)
- [DynamoDBDocument](#)
- [DynamoDB HashKey](#)
- [DynamoDBIgnore](#)
- [DynamoDB IndexHashKey](#)
- [DynamoDB IndexRangeKey](#)
- [DynamoDB RangeKey](#)
- [DynamoDBTable](#)
- [DynamoDB TypeConverted](#)
- [DynamoDBTyped](#)
- [DynamoDB VersionAttribute](#)

DynamoDBAttribute

Memetakan properti ke atribut tabel. Secara default, setiap properti kelas dipetakan ke atribut item dengan nama yang sama. Namun, jika nama tidak sama, Anda dapat menggunakan anotasi ini untuk memetakan properti ke atribut. Dalam cuplikan Java berikut, `DynamoDBAttribute` memetakan properti `BookAuthors` ke nama atribut `Authors` dalam tabel.

```
@DynamoDBAttribute(attributeName = "Authors")
```

```
public List<String> getBookAuthors() { return BookAuthors; }
public void setBookAuthors(List<String> BookAuthors) { this.BookAuthors =
    BookAuthors; }
```

DynamoDBMapper menggunakan Authors sebagai nama atribut saat menyimpan objek ke tabel.

DynamoDB AutoGeneratedKey

Menandai kunci partisi atau properti kunci urutan sebagai dibuat otomatis. DynamoDBMapper membuat [UUID](#) acak saat menyimpan atribut ini. Hanya properti String yang dapat ditandai sebagai kunci yang dibuat otomatis.

Contoh berikut mendemonstrasikan menggunakan kunci yang dibuat otomatis.

```
@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());
    }
}
```

DynamoDB AutoGeneratedTimestamp

Membuat stempel waktu secara otomatis.

```
@DynamoDBAutoGeneratedTimestamp(strategy=DynamoDBAutoGenerateStrategy.ALWAYS)
public Date getLastUpdatedDate() { return lastUpdatedDate; }
public void setLastUpdatedDate(Date lastUpdatedDate) { this.lastUpdatedDate =
    lastUpdatedDate; }
```

Secara opsional, strategi pembuatan otomatis dapat didefinisikan dengan menyediakan atribut strategi. Nilai default-nya ALWAYS.

DynamoDBDocument

Menunjukkan bahwa kelas dapat diserialisasi sebagai dokumen Amazon DynamoDB.

Misalnya, katakanlah Anda ingin memetakan dokumen JSON ke atribut DynamoDB untuk jenis Peta (M). Contoh kode berikut mendefinisikan item yang berisi atribut bersarang (Gambar) untuk jenis Peta.

```
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; }

}
}
```

Kemudian, Anda dapat menyimpan item ProductCatalog, dengan Pictures, seperti yang ditunjukkan dalam contoh berikut.

```
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);
```

Item ProductCatalog yang dihasilkan akan tampak seperti berikut (dalam format JSON).

```
{
  "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"
  }
}
```

DynamoDB HashKey

Memetakan properti kelas ke kunci partisi tabel. Properti harus berupa salah satu string skalar, angka, atau jenis biner. Properti tidak dapat berupa jenis koleksi.

Asumsikan bahwa Anda memiliki tabel, `ProductCatalog`, yang memiliki `Id` sebagai kunci primer. Kode Java berikut mendefinisikan kelas `CatalogItem` dan memetakan properti `Id`-nya ke kunci primer dari tabel `ProductCatalog` menggunakan tanda `@DynamoDBHashKey`.

```
@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

Menunjukkan ke instans `DynamoDBMapper` bahwa properti terkait harus diabaikan. Saat menyimpan data ke tabel, `DynamoDBMapper` tidak menyimpan properti ini ke tabel.

Diterapkan pada metode getter atau bidang kelas untuk properti yang tidak dimodelkan. Jika anotasi diterapkan langsung ke bidang kelas, getter dan setter yang sesuai harus dinyatakan di kelas yang sama.

DynamoDB IndexHashKey

Memetakan properti kelas ke kunci partisi untuk indeks sekunder global. Properti harus berupa salah satu string skalar, angka, atau jenis biner. Properti tidak dapat berupa jenis koleksi.

Gunakan anotasi ini jika Anda perlu Query indeks sekunder global. Anda harus menentukan nama indeks (`globalSecondaryIndexName`). Jika nama properti kelas berbeda dari kunci partisi indeks, Anda juga harus menentukan nama atribut indeks (`attributeName`).

DynamoDB IndexRangeKey

Memetakan properti kelas untuk kunci urutan indeks sekunder global atau indeks sekunder lokal. Properti harus berupa salah satu string skalar, angka, atau jenis biner. Properti tidak dapat berupa jenis koleksi.

Gunakan anotasi ini jika anda perlu Query indeks sekunder lokal atau indeks sekunder global dan ingin menyempurnakan hasil menggunakan kunci urutan indeks. Anda harus menentukan nama indeks (baik `globalSecondaryIndexName` maupun `localSecondaryIndexName`). Jika nama properti kelas berbeda dari kunci urutan indeks, Anda juga harus menentukan nama atribut indeks (`attributeName`).

DynamoDB RangeKey

Memetakan properti kelas untuk kunci urutan tabel. Properti harus berupa salah satu string skalar, angka, atau jenis biner. Properti ini tidak boleh berupa jenis koleksi.

Jika kunci primer adalah komposit (kunci partisi dan kunci urutan), Anda dapat menggunakan tanda ini untuk memetakan bidang kelas Anda ke kunci urutan. Misalnya, Anda memiliki tabel `Reply` yang menyimpan balasan untuk utas forum. Setiap utas dapat berisi banyak balasan. Jadi, kunci primer tabel ini adalah `ThreadId` dan `ReplyDateTime`. `ThreadId` adalah kunci partisinya, dan `ReplyDateTime` adalah kunci urutannya.

Kode Java berikut akan mendefinisikan `Reply` dan memetakannya ke tabel `Reply`. Kode tersebut menggunakan tanda `@DynamoDBHashKey` dan `@DynamoDBRangeKey` untuk mengidentifikasi properti kelas yang dipetakan ke kunci primer.

```
@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

Mengidentifikasi tabel target di DynamoDB. Misalnya, kode Java berikut menentukan kelas `Developer` dan memetakannya ke tabel `People` di DynamoDB.

```
@DynamoDBTable(tableName="People")
public class Developer { ...}
```

Anotasi `@DynamoDBTable` dapat diwariskan. Setiap kelas baru yang diwariskan dari kelas `Developer` juga dipetakan ke tabel `People` yang sama. Misalnya, Anda membuat kelas `Lead` yang diwariskan dari kelas `Developer`. Karena Anda memetakan kelas `Developer` ke tabel `People`, objek kelas `Lead` juga disimpan dalam tabel yang sama.

`@DynamoDBTable` juga dapat diganti. Setiap kelas baru yang diwariskan dari kelas `Developer` secara default dipetakan ke tabel `People` yang sama. Namun, Anda dapat mengganti pemetaan default ini. Misalnya, jika Anda membuat kelas yang diwariskan dari kelas `Developer`, Anda dapat secara eksplisit memetakannya ke tabel lain dengan menambahkan anotasi `@DynamoDBTable` seperti yang ditunjukkan dalam contoh kode Java berikut.

```
@DynamoDBTable(tableName="Managers")
public class Manager extends Developer { ...}
```

DynamoDB TypeConverted

Anotasi untuk menandai properti sebagai menggunakan konverter jenis khusus. Dapat dianotasikan pada anotasi yang ditetapkan pengguna untuk meneruskan properti tambahan ke `DynamoDBTypeConverter`.

Antarmuka `DynamoDBTypeConverter` memungkinkan Anda memetakan jenis data arbitrer Anda sendiri ke jenis data yang secara asli didukung oleh DynamoDB. Untuk informasi selengkapnya, lihat [Memetakan data arbitrer](#).

DynamoDBTyped

Anotasi untuk menimpa pengikatan jenis atribut standar. Jenis standar tidak memerlukan anotasi jika menerapkan pengikatan atribut default untuk jenis tersebut.

DynamoDB VersionAttribute

Mengidentifikasi properti kelas untuk menyimpan nomor versi penguncian optimis. `DynamoDBMapper` menetapkan nomor versi untuk properti ini ketika menyimpan item baru, dan menambahkannya setiap kali Anda memperbarui item. Hanya jenis skalar nomor yang didukung. Untuk informasi selengkapnya tentang jenis data, lihat [Tipe Data](#). Untuk informasi selengkapnya tentang versioning, lihat [Penguncian Optimis dengan nomor versi](#).

Kelas DynamoDBMapper

Kelas `DynamoDBMapper` adalah titik masuk ke Amazon DynamoDB. Kelas ini menyediakan akses ke titik akhir DynamoDB dan memungkinkan Anda mengakses data Anda dalam berbagai tabel. Kelas ini juga memungkinkan Anda melakukan berbagai operasi buat, baca, perbarui, dan hapus (CRUD) pada item, dan menjalankan query serta memindai tabel. Kelas ini menyediakan metode berikut agar berfungsi dengan DynamoDB.

Untuk dokumentasi Javadoc yang sesuai, lihat [DynamoDBMapper](#) di Referensi API AWS SDK for Java .

Topik

- [save](#)
- [load](#)
- [delete](#)
- [query](#)
- [queryPage](#)
- [scan](#)
- [scanPage](#)
- [parallelScan](#)
- [batchSave](#)
- [batchLoad](#)
- [batchDelete](#)
- [batchWrite](#)
- [transactionWrite](#)
- [transactionLoad](#)
- [count](#)
- [generateCreateTablePermintaan](#)
- [createS3Link](#)
- [GetS3 ClientCache](#)

save

Menyimpan objek tertentu ke tabel. Objek yang ingin Anda simpan adalah satu-satunya parameter yang diperlukan untuk metode ini. Anda dapat memberikan parameter konfigurasi opsional menggunakan objek `DynamoDBMapperConfig`.

Jika item yang memiliki kunci primer yang sama tidak ada, metode ini akan membuat item baru dalam tabel. Jika item yang memiliki kunci primer yang sama ada, item yang ada akan diperbarui. Jika kunci partisi dan kunci urutan berupa String jenis dan dianotasikan dengan `@DynamoDBAutoGeneratedKey`, kunci tersebut diberi pengidentifikasi unik universal acak (UUID) jika tidak diinisialisasi. Bidang versi yang diberi anotasi dengan `@DynamoDBVersionAttribute` bertambah satu. Selain itu, jika bidang versi diperbarui atau kunci dibuat, objek yang diteruskan akan diperbarui sebagai hasil operasi.

Secara default, hanya atribut yang sesuai dengan properti kelas yang dipetakan yang akan diperbarui. Atribut tambahan yang ada pada item tidak akan terpengaruh. Namun, jika menentukan `SaveBehavior.CLOBBER`, Anda dapat memaksa item untuk ditimpa sepenuhnya.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER).build();

mapper.save(item, config);
```

Jika versioning diaktifkan, versi item sisi klien dan sisi server harus cocok. Namun, versi tidak harus cocok jika opsi `SaveBehavior.CLOBBER` digunakan. Untuk informasi selengkapnya tentang versioning, lihat [Penguncian Optimis dengan nomor versi](#).

load

Mengambil item dari tabel. Anda harus menyediakan kunci item primer yang ingin Anda ambil. Anda dapat memberikan parameter konfigurasi opsional menggunakan objek `DynamoDBMapperConfig`. Misalnya, Anda secara opsional dapat meminta bacaan sangat konsisten untuk memastikan bahwa metode ini hanya mengambil nilai item terbaru seperti yang ditunjukkan dalam pernyataan Java berikut.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT).build();

CatalogItem item = mapper.load(CatalogItem.class, item.getId(), config);
```

Secara default, DynamoDB akan mengembalikan item yang memiliki nilai yang akhirnya konsisten. Untuk informasi tentang model konsistensi akhir DynamoDB, lihat [Konsistensi baca](#).

delete

Menghapus item dari tabel. Anda harus meneruskan instans objek dari kelas yang dipetakan.

Jika versioning diaktifkan, versi item sisi klien dan sisi server harus cocok. Namun, versi tidak harus cocok jika opsi `SaveBehavior.CLOBBER` digunakan. Untuk informasi selengkapnya tentang versioning, lihat [Penguncian Optimis dengan nomor versi](#).

query

Mengkueri tabel atau indeks sekunder. Anda dapat mengkueri tabel atau indeks hanya jika memiliki kunci primer komposit (kunci partisi dan kunci urutan). Metode ini mengharuskan Anda untuk memberikan nilai kunci partisi dan filter kueri yang diterapkan pada kunci urutan. Ekspresi filter menyertakan syarat dan nilai.

Misalnya, Anda memiliki tabel, `Reply`, yang menyimpan balasan utas forum. Setiap subjek utas dapat berisi nol atau beberapa balasan. Kunci primer tabel `Reply` terdiri dari bidang `Id` dan `ReplyDateTime`, dengan `Id` sebagai kunci partisi dan `ReplyDateTime` sebagai kunci urutan kunci primer.

```
Reply ( Id, ReplyDateTime, ... )
```

Misalnya, Anda membuat pemetaan antara kelas `Reply` dan tabel `Reply` yang sesuai di DynamoDB. Kode Java berikut menggunakan `DynamoDBMapper` untuk menemukan semua balasan dalam dua minggu terakhir untuk subjek utas tertentu.

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);
```

Kueri ini mengembalikan koleksi objek Reply.

Secara default, metode query mengembalikan koleksi "lazy-loaded". Metode ini awalnya hanya mengembalikan satu halaman hasil, lalu membuat panggilan layanan untuk halaman berikutnya jika diperlukan. Untuk mendapatkan semua item yang cocok, iterasikan ke koleksi latestReplies.

Perhatikan bahwa memanggil metode size() pada koleksi akan memuat setiap hasil untuk memberikan jumlah yang akurat. Hal ini dapat mengakibatkan penggunaan banyak throughput yang disediakan, dan bahkan bisa menghabiskan semua memori di JVM Anda pada tabel yang sangat besar.

Untuk mencari indeks, Anda harus membuat model indeks sebagai kelas pemeta terlebih dahulu. Misalkan Reply tabel memiliki indeks sekunder global bernama PostedBy-Message-Index. Kunci partisi untuk indeks ini adalah PostedBy, dan kunci urutannya adalah Message. Definisi kelas untuk item dalam indeks akan tampak seperti berikut ini.

```
@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.
```



```
}
```

Anotasi `@DynamoDBTable` menunjukkan bahwa indeks ini terkait dengan tabel `Reply`. `@DynamoDBIndexHashKey` Anotasi menunjukkan kunci partisi (`PostedBy`) dari indeks, dan `@DynamoDBIndexRangeKey` menunjukkan kunci sortir (`Pesan`) dari indeks.

Sekarang, Anda dapat menggunakan `DynamoDBMapper` untuk mengkueri indeks, mengambil subset pesan yang diposting oleh pengguna tertentu. Anda tidak perlu menentukan nama indeks jika Anda tidak memiliki pemetaan yang bertentangan di seluruh tabel dan indeks serta pemetaan sudah dibuat di mapper. Pemeta akan menyimpulkan berdasarkan kunci primer dan kunci urutan. Kode berikut mengkueri indeks sekunder global. Karena indeks sekunder global mendukung bacaan akhir konsisten tetapi bukan bacaan sangat konsisten, Anda harus menentukan `withConsistentRead(false)`.

```
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);
```

Kueri ini mengembalikan koleksi objek `PostedByMessage`.

queryPage

Mengkueri tabel atau indeks sekunder dan mengembalikan satu halaman hasil yang cocok. Seperti metode `query`, Anda harus menentukan nilai kunci partisi dan filter kueri yang diterapkan pada atribut kunci urutan. Namun, `queryPage` hanya mengembalikan “halaman” pertama data, yaitu jumlah data yang sesuai dalam ukuran 1 MB

scan

Memindai seluruh tabel atau indeks sekunder. Anda dapat menentukan `FilterExpression` untuk memfilter kumpulan hasil secara opsional.

Misalnya, Anda memiliki tabel, Reply, yang menyimpan balasan utas forum. Setiap subjek utas dapat berisi nol atau beberapa balasan. Kunci primer tabel Reply terdiri dari bidang Id dan ReplyDateTime, dengan Id sebagai kunci partisi dan ReplyDateTime sebagai kunci urutan kunci primer.

```
Reply ( Id, ReplyDateTime, ... )
```

Jika memetakan kelas Java ke tabel Reply, Anda dapat menggunakan DynamoDBMapper untuk memindai tabel. Misalnya, kode Java berikut memindai seluruh tabel Reply, sehingga hanya mengembalikan balasan untuk tahun tertentu.

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);
```

Secara default, metode scan mengembalikan koleksi "lazy-loaded". Metode ini awalnya hanya mengembalikan satu halaman hasil, lalu membuat panggilan layanan untuk halaman berikutnya jika diperlukan. Untuk mendapatkan semua item yang cocok, iterasikan ke koleksi replies.

Perhatikan bahwa memanggil metode size() pada koleksi akan memuat setiap hasil untuk memberikan jumlah yang akurat. Hal ini dapat mengakibatkan penggunaan banyak throughput yang disediakan, dan bahkan bisa menghabiskan semua memori di JVM Anda pada tabel yang sangat besar.

Untuk memindai indeks, Anda harus membuat model indeks sebagai kelas pemeta terlebih dahulu. Misalkan tabel Reply memiliki indeks sekunder global bernama PostedBy-Message-Index. Kunci partisi untuk indeks ini adalah PostedBy, dan kunci urutannya adalah Message. Kelas pemeta untuk indeks ini ditampilkan di bagian [query](#). Kelas ini menggunakan anotasi @DynamoDBIndexHashKey dan @DynamoDBIndexRangeKey untuk menentukan kunci partisi indeks dan kunci urutan.

Contoh kode berikut memindai PostedBy-Message-Index. Contoh kode ini tidak menggunakan filter pemindaian, sehingga semua item dalam indeks dikembalikan kepada Anda.

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false);

List<PostedByMessage> iList = mapper.scan(PostedByMessage.class, scanExpression);
Iterator<PostedByMessage> indexItems = iList.iterator();
```

scanPage

Memindai tabel atau indeks sekunder dan mengembalikan satu halaman hasil yang cocok. Seperti halnya metode `scan`, Anda dapat menentukan `FilterExpression` untuk memfilter kumpulan hasil secara opsional. Namun, `scanPage` hanya mengembalikan “halaman” pertama data, yaitu jumlah data yang sesuai dalam ukuran 1 MB.

parallelScan

Melakukan pemindaian paralel dari seluruh tabel atau indeks sekunder. Anda menentukan sejumlah segmen logis untuk tabel, beserta ekspresi pemindaian untuk memfilter hasil. `parallelScan` membagi tugas pemindaian antara beberapa pekerja, satu tugas untuk setiap segmen logis; pekerja akan memproses data secara paralel dan mengembalikan hasilnya.

Contoh kode Java berikut melakukan pemindaian paralel pada tabel `Product`.

```
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);
```

Untuk contoh kode Java yang menggambarkan penggunaan `parallelScan`, lihat [Operasi kueri dan pindai DynamoDBMapper](#).

batchSave

Menyimpan objek ke satu atau beberapa tabel menggunakan satu atau beberapa panggilan ke metode `AmazonDynamoDB.batchWriteItem`. Metode ini tidak memberikan jaminan transaksi.

Kode Java berikut menyimpan dua item (buku) ke tabel ProductCatalog.

```
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

Mengambil beberapa item dari satu atau beberapa tabel menggunakan kunci primernya.

Kode Java berikut akan mengambil dua item dari dua tabel yang berbeda.

```
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

Menghapus objek dari satu atau beberapa tabel menggunakan satu atau beberapa panggilan ke metode `AmazonDynamoDB.batchWriteItem`. Metode ini tidak memberikan jaminan transaksi.

Kode Java berikut akan menghapus dua item (buku) dari tabel ProductCatalog.

```
Book book1 = mapper.load(Book.class, 901);
Book book2 = mapper.load(Book.class, 902);
```

```
mapper.batchDelete(Arrays.asList(book1, book2));
```

batchWrite

Menyimpan objek ke dan menghapus objek dari satu atau beberapa tabel menggunakan satu atau beberapa panggilan ke metode AmazonDynamoDB.`batchWriteItem`. Metode ini tidak memberikan jaminan transaksi atau versioning dukungan (penempatan atau penghapusan bersyarat).

Kode Java berikut menulis item baru ke tabel Forum, menulis item baru ke tabel Thread, dan menghapus item dari tabel ProductCatalog.

```
// 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

Menyimpan objek ke dan menghapus objek dari satu atau beberapa tabel menggunakan satu panggilan ke metode AmazonDynamoDB.`transactWriteItems`.

[Untuk daftar pengecualian khusus transaksi, lihat kesalahan. `TransactWriteItems`](#)

Untuk informasi selengkapnya tentang transaksi DynamoDB dan jaminan atomisitas, konsistensi, isolasi, dan durabilitas atau atomicity, consistency, isolation, and durability (ACID), lihat [Amazon DynamoDB Transactions](#).

Note

Metode ini tidak mendukung hal berikut:

- [DynamoDBMapperConfig. SaveBehavior.](#)

Kode Java berikut menulis item baru ke setiap tabel Forum dan Thread, secara transaksional.

```
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

Memuat objek dari satu atau beberapa tabel menggunakan satu panggilan ke metode `AmazonDynamoDB.transactGetItems`.

[Untuk daftar pengecualian khusus transaksi, lihat kesalahan. TransactGetItems](#)

Untuk informasi selengkapnya tentang transaksi DynamoDB dan jaminan atomisitas, konsistensi, isolasi, dan durabilitas atau atomicity, consistency, isolation, and durability (ACID), lihat [Amazon DynamoDB Transactions](#).

Kode Java berikut akan memuat satu item dari setiap tabel Forum dan Thread, secara transaksional.

```
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

Mengevaluasi ekspresi pemindaian yang ditentukan dan mengembalikan jumlah item yang cocok. Tidak ada data item yang dikembalikan.

generateCreateTablePermintaan

Mengurai kelas POJO yang mewakili tabel DynamoDB, dan mengembalikan `CreateTableRequest` untuk tabel tersebut.

createS3Link

Membuat tautan ke objek di Amazon S3. Anda harus menentukan nama bucket dan nama kunci, yang secara unik mengidentifikasi objek di bucket.

Untuk menggunakan `createS3Link`, kelas pemeta Anda harus menentukan metode getter dan setter. Contoh kode berikut mengilustrasikan hal ini dengan menambahkan atribut baru dan metode getter/setter untuk kelas `CatalogItem`.

```
@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;  
    }  
  
    ...  
}
```

Kode Java berikut mendefinisikan item baru yang akan ditulis ke tabel Product. Item tersebut termasuk link ke gambar produk; data gambar diunggah ke Amazon S3.

```
CatalogItem item = new CatalogItem();

item.setId(150);
item.setTitle("Book 150 Title");

String myS3Bucket = "myS3bucket";
String myS3Key = "productImages/book_150_cover.jpg";
item.setProductImage(mapper.createS3Link(myS3Bucket, myS3Key));

item.getProductImage().uploadFrom(new File("/file/path/book_150_cover.jpg"));

mapper.save(item);
```

Kelas S3Link menyediakan banyak metode lain untuk memanipulasi objek di Amazon S3. Untuk informasi selengkapnya, lihat [Javadocs untuk S3Link](#).

GetS3 ClientCache

Mengembalikan S3ClientCache yang mendasarinya untuk mengakses Amazon S3.

S3ClientCache adalah Peta cerdas untuk objek AmazonS3Client. Jika Anda memiliki banyak klien, an S3ClientCache dapat membantu Anda menjaga klien tetap terorganisir berdasarkan AWS Wilayah, dan dapat membuat klien Amazon S3 baru sesuai permintaan.

Pengaturan konfigurasi opsional untuk DynamoDBMapper

Saat Anda membuat instans DynamoDBMapper, instans tersebut memiliki perilaku default tertentu; Anda dapat menimpa pengaturan default ini menggunakan kelas DynamoDBMapperConfig.

Cuplikan kode berikut akan membuat DynamoDBMapper dengan pengaturan kustom:

```
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);
```

[Untuk informasi selengkapnya, lihat MapperConfigDynamoDB di AWS SDK for Java Referensi API.](#)

Anda dapat menggunakan argumen berikut untuk instans `DynamoDBMapperConfig`:

- Nilai enumerasi `DynamoDBMapperConfig.ConsistentReads`:
 - `EVENTUAL`—instans pemeta menggunakan permintaan bacaan akhir konsisten.
 - `CONSISTENT`—instans pemeta menggunakan permintaan bacaan sangat konsisten. Anda dapat menggunakan pengaturan opsional ini dengan operasi `load`, `query`, atau `scan`. Bacaan sangat konsisten memiliki implikasi terhadap performa dan penagihan; lihat [halaman detail produk DynamoDB](#) untuk informasi selengkapnya.

Jika Anda tidak menentukan pengaturan konsistensi baca untuk instans pemeta Anda, pengaturan defaultnya adalah `EVENTUAL`.

 Note

Nilai ini diterapkan dalam operasi `query`, `querypage`, `load`, dan `batch load` `DynamoDBMapper`.

- Nilai enumerasi `DynamoDBMapperConfig.PaginationLoadingStrategy`—Mengontrol cara instans pemeta memproses daftar data yang diberi nomor halaman, seperti hasil dari `query` atau `scan`:
 - `LAZY_LOADING`—instans pemeta memuat data jika memungkinkan, dan menyimpan semua hasil yang dimuat dalam memori.
 - `EAGER_LOADING`—instans pemeta memuat data segera setelah daftar diinisialisasi.
 - `ITERATION_ONLY`—Anda hanya dapat menggunakan `Iterator` untuk membaca dari daftar. Selama iterasi, daftar akan menghapus semua hasil lama sebelum memuat halaman berikutnya, sehingga daftar akan menyimpan maksimal satu halaman dari hasil yang dimuat dalam memori. Hal ini juga berarti bahwa daftar hanya dapat diiterasi satu kali. Strategi ini direkomendasikan saat menangani item besar, guna mengurangi overhead memori.

Jika Anda tidak menentukan strategi pemuatan pemberian nomor halaman untuk instans pemeta Anda, pengaturan defaultnya adalah `LAZY_LOADING`.

- Nilai enumerasi `DynamoDBMapperConfig.SaveBehavior` - Menentukan cara instans pemeta akan menangani atribut selama operasi simpan:
 - `UPDATE`—selama operasi simpan, semua atribut model diperbarui, dan atribut yang tidak dibuat modelnya tidak terpengaruh. Jenis bilangan primitif (byte, int, long) diatur ke 0. Jenis objek diatur ke nol.
 - `CLOBBER`—menghapus dan mengganti semua atribut, termasuk yang tidak dibuat modelnya, selama operasi simpan. Hal ini dilakukan dengan menghapus item lalu membuatnya kembali. Batasan bidang dengan versi juga diabaikan.

Jika Anda tidak menentukan perilaku simpan untuk instans pemeta, pengaturan defaultnya adalah `UPDATE`.

Note

Operasi transaksional `DynamoDBMapper` tidak mendukung enumerasi `DynamoDBMapperConfig.SaveBehavior`.

- Objek `DynamoDBMapperConfig.TableNameOverride`—Menginstruksikan instans pemeta untuk mengabaikan nama tabel yang ditentukan oleh anotasi `DynamoDBTable` kelas, dan menggunakan nama tabel lain yang Anda berikan. Ini berguna saat mempartisi data Anda menjadi beberapa tabel saat runtime.

Anda dapat mengganti objek konfigurasi default untuk `DynamoDBMapper` per operasi, jika diperlukan.

Penguncian Optimis dengan nomor versi

Penguncian optimis adalah strategi untuk memastikan bahwa item sisi klien yang Anda perbarui (atau hapus) sama dengan item di Amazon DynamoDB. Jika Anda menggunakan strategi ini, penulisan basis data Anda dilindungi agar tidak ditimpa oleh penulisan orang lain, dan sebaliknya.

Dengan penguncian optimis, setiap item memiliki atribut yang berfungsi sebagai nomor versi. Jika Anda mengambil item dari tabel, aplikasi akan mencatat nomor versi item tersebut. Anda dapat memperbarui item, tetapi hanya jika nomor versi di sisi server tidak berubah. Jika ada ketidakcocokan versi, artinya orang lain telah mengubah item sebelum Anda melakukannya. Upaya pembaruan gagal, karena versi item Anda sudah usang. Jika ini terjadi, coba lagi dengan mengambil item dan kemudian mencoba memperbaruinya. Penguncian optimis mencegah Anda menimpa perubahan

secara tidak sengaja yang dibuat oleh orang lain. Penguncian optimis juga mencegah orang lain menimpa perubahan Anda secara tidak sengaja.

Meskipun Anda dapat menerapkan strategi penguncian optimis Anda sendiri, AWS SDK for Java memberikan `@DynamoDBVersionAttribute` anotasi. Di kelas pemetaan untuk tabel, Anda menetapkan satu properti untuk menyimpan nomor versi, dan menandainya menggunakan anotasi ini. Saat Anda menyimpan objek, item terkait dalam tabel DynamoDB akan memiliki atribut yang menyimpan nomor versi. `DynamoDBMapper` menetapkan nomor versi saat Anda pertama kali menyimpan objek, dan otomatis menambahkan nomor versi setiap kali Anda memperbarui item. Permintaan perbarui atau hapus Anda hanya akan berhasil jika versi objek sisi klien cocok dengan nomor versi item yang sesuai dalam tabel DynamoDB.

`ConditionalCheckFailedException` ditampilkan jika:

- Anda menggunakan penguncian optimis dengan `@DynamoDBVersionAttribute` dan nilai versi pada server yang berbeda dari nilai pada sisi klien.
- Anda menentukan batasan bersyarat Anda sendiri saat menyimpan data dengan menggunakan `DynamoDBMapper` dengan `DynamoDBSaveExpression` dan batasan ini gagal.

Note

- Tabel global DynamoDB menggunakan rekonsiliasi “penulis terakhir menang” antara pembaruan bersamaan. Jika Anda menggunakan tabel global, kebijakan penulis terakhir akan menang. Jadi dalam hal ini, strategi penguncian tidak berfungsi seperti yang diharapkan.
- Operasi tulis transaksional `DynamoDBMapper` tidak mendukung ekspresi syarat dan anotasi `@DynamoDBVersionAttribute` pada objek yang sama. Jika sebuah objek dalam penulisan transaksional dianotasi dengan `@DynamoDBVersionAttribute` dan juga memiliki ekspresi kondisi, maka akan dilemparkan. `SdkClientException`

Misalnya, kode Java berikut mendefinisikan kelas `CatalogItem` yang memiliki beberapa properti. Properti `Version` ditandai dengan anotasi `@DynamoDBVersionAttribute`.

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;}
}
```

Anda dapat menerapkan anotasi `@DynamoDBVersionAttribute` untuk jenis yang dapat dinihilkan yang diberikan oleh kelas pembungkus primitif yang menyediakan jenis yang dapat dinihilkan, seperti `Long` dan `Integer`.

Penguncian optimis memiliki dampak sebagai berikut pada metode `DynamoDBMapper` ini:

- `save` — Untuk item baru, `DynamoDBMapper` menetapkan nomor versi awal 1. Jika Anda mengambil item, memperbarui satu atau beberapa propertinya, dan mencoba untuk menyimpan

perubahan, operasi simpan hanya akan berhasil jika nomor versi di sisi klien dan sisi server cocok. `DynamoDBMapper` menambah nomor versi secara otomatis.

- `delete` — Metode `delete` mengambil sebuah objek sebagai parameter, dan `DynamoDBMapper` melakukan pemeriksaan versi sebelum menghapus item. Pemeriksaan versi dapat dinonaktifkan jika `DynamoDBMapperConfig.SaveBehavior.CLOBBER` ditentukan dalam permintaan.

Implementasi internal penguncian optimis dalam `DynamoDBMapper` menggunakan pembaruan bersyarat dan dukungan penghapusan bersyarat yang disediakan oleh DynamoDB.

- `transactionWrite` —
 - `Put` — Untuk item baru, `DynamoDBMapper` menetapkan nomor versi awal 1. Jika Anda mengambil item, memperbarui satu atau beberapa propertinya, dan mencoba untuk menyimpan perubahan, operasi masukkan hanya akan berhasil jika nomor versi di sisi klien dan sisi server cocok. `DynamoDBMapper` menambah nomor versi secara otomatis.
 - `Update` — Untuk item baru, `DynamoDBMapper` menetapkan nomor versi awal 1. Jika Anda mengambil item, memperbarui satu atau beberapa propertinya, dan mencoba untuk menyimpan perubahan, operasi perbarui hanya akan berhasil jika nomor versi di sisi klien dan sisi server cocok. `DynamoDBMapper` menambah nomor versi secara otomatis.
 - `Delete` — `DynamoDBMapper` melakukan pemeriksaan versi sebelum menghapus item. Operasi hapus hanya akan berhasil jika nomor versi pada sisi klien dan sisi server cocok.
 - `ConditionCheck` — Anotasi `@DynamoDBVersionAttribute` tidak didukung untuk operasi `ConditionCheck`. An `SdkClientException` akan dilemparkan ketika `ConditionCheck` item dianotasi dengan `@DynamoDBVersionAttribute`

Menonaktifkan penguncian optimis

Untuk menonaktifkan penguncian optimis, Anda dapat mengubah nilai enumerasi `DynamoDBMapperConfig.SaveBehavior` dari `UPDATE` menjadi `CLOBBER`. Anda dapat melakukannya dengan membuat instans `DynamoDBMapperConfig` yang melewati pemeriksaan versi dan menggunakan instans ini untuk semua permintaan Anda. Untuk informasi tentang `DynamoDBMapperConfig.SaveBehavior` dan parameter `DynamoDBMapper` opsional lainnya, lihat [Pengaturan konfigurasi opsional untuk DynamoDBMapper](#).

Anda juga dapat mengatur perilaku penguncian untuk operasi tertentu saja. Misalnya, cuplikan Java berikut menggunakan `DynamoDBMapper` untuk menyimpan item katalog. Cuplikan ini menentukan `DynamoDBMapperConfig.SaveBehavior` dengan menambahkan parameter `DynamoDBMapperConfig` opsional untuk metode `save`.

Note

Metode `TransactionWrite` tidak mendukung `DynamoDB.MapperConfig.SaveBehavior` konfigurasi. Menonaktifkan penguncian optimis untuk `transactionWrite` tidak didukung.

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));
```

Memetakan data arbitrer

Selain jenis Java yang didukung (lihat [Jenis data yang didukung untuk DynamoDB Mapper untuk Java](#)), Anda dapat menggunakan jenis di aplikasi Anda yang tidak memiliki pemetaan langsung ke jenis Amazon DynamoDB. Untuk memetakan jenis ini, Anda harus menyediakan implementasi yang mengonversi jenis kompleks Anda ke jenis yang didukung DynamoDB dan sebaliknya, serta menganotasi metode penilai jenis kompleks menggunakan anotasi `@DynamoDBTypeConverted`. Kode konverter mengubah data ketika objek disimpan atau dimuat. Kode ini juga digunakan untuk semua operasi yang mengonsumsi jenis kompleks. Perhatikan bahwa saat membandingkan data selama operasi kueri dan pindai, perbandingan pada data yang disimpan dalam DynamoDB akan dilakukan.

Misalnya, pertimbangkan kelas `CatalogItem` berikut yang mendefinisikan properti, `Dimension`, yaitu `DimensionType`. Properti ini menyimpan dimensi item sebagai tinggi, lebar, dan ketebalan. Misalnya Anda memutuskan untuk menyimpan dimensi item ini sebagai string (seperti `8.5x11x.05`) dalam DynamoDB. Contoh berikut memberikan kode konverter yang mengonversi objek `DimensionType` menjadi string dan string menjadi `DimensionType`.

Note

Contoh kode ini mengasumsikan bahwa Anda telah memuat data ke DynamoDB untuk akun Anda dengan mengikuti petunjuk di bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihat [Contoh kode Java](#).

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());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
        itemDimension.setHeight(data[1].trim());
        itemDimension.setThickness(data[2].trim());
    }
} catch (Exception e) {
    e.printStackTrace();
}

return itemDimension;
}
}
```

Contoh DynamoDBMapper

Contoh-contoh kode Java berikut ini menunjukkan cara melakukan berbagai operasi dengan kelas `DynamoDBMapper`. Anda dapat menggunakan contoh-contoh ini untuk melakukan operasi CRUD, kueri, pindai, batch, dan transaksi.

Topik

- [Operasi CRUD DynamoDBMapper](#)
- [Operasi kueri dan pindai DynamoDBMapper](#)
- [Operasi batch DynamoDBMapper](#)
- [Operasi Transaksi DynamoDBMapper](#)

Operasi CRUD DynamoDBMapper

Contoh kode Java berikut menyatakan kelas `CatalogItem` yang memiliki properti `Id`, `Title`, `ISBN`, dan `Authors`. Contoh ini menggunakan anotasi untuk memetakan properti ini ke tabel `ProductCatalog` di DynamoDB. Contoh kemudian menggunakan `DynamoDBMapper` untuk menyimpan objek buku, mengambilnya, memperbaruinya, lalu menghapus item buku.

Note

Contoh kode ini mengasumsikan bahwa Anda telah memuat data ke DynamoDB untuk akun Anda dengan mengikuti petunjuk di bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihat [Contoh kode Java](#).

Impor

```
import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapperConfig;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
```

Kode

```
public class DynamoDBMapperCRUExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

    public static void main(String[] args) throws IOException {
        testCRUDOperations();
        System.out.println("Example complete!");
    }

    @DynamoDBTable(tableName = "ProductCatalog")
    public static class CatalogItem {
        private Integer id;
        private String title;
        private String ISBN;
        private Set<String> bookAuthors;

        // Partition key
        @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;
}

@Override
public String toString() {
    return "Book [ISBN=" + ISBN + ", bookAuthors=" + bookAuthors + ", id=" + id
+ ", title=" + title + "]";
}

private static void testCRUDOperations() {

    CatalogItem item = new CatalogItem();
    item.setId(601);
    item.setTitle("Book 601");
    item.setISBN("611-1111111111");
    item.setBookAuthors(new HashSet<String>(Arrays.asList("Author1", "Author2")));

    // Save the item (book).
    DynamoDBMapper mapper = new DynamoDBMapper(client);
```

```
mapper.save(item);

// Retrieve the item.
CatalogItem itemRetrieved = mapper.load(CatalogItem.class, 601);
System.out.println("Item retrieved:");
System.out.println(itemRetrieved);

// Update the item.
itemRetrieved.setISBN("622-2222222222");
itemRetrieved.setBookAuthors(new HashSet<String>(Arrays.asList("Author1",
"Author3")));
mapper.save(itemRetrieved);
System.out.println("Item updated:");
System.out.println(itemRetrieved);

// Retrieve the updated item.
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT)
    .build();
CatalogItem updatedItem = mapper.load(CatalogItem.class, 601, config);
System.out.println("Retrieved the previously updated item:");
System.out.println(updatedItem);

// Delete the item.
mapper.delete(updatedItem);

// Try to retrieve deleted item.
CatalogItem deletedItem = mapper.load(CatalogItem.class, updatedItem.getId(),
config);
if (deletedItem == null) {
    System.out.println("Done - Sample item is deleted.");
}
}
}
```

Operasi kueri dan pindai DynamoDBMapper

Contoh Java di bagian ini mendefinisikan kelas berikut dan memetakannya ke tabel di Amazon DynamoDB. Untuk informasi selengkapnya tentang membuat tabel sampel, lihat [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

- Kelas Book dipetakan ke tabel ProductCatalog

- Kelas Forum, Thread, dan Reply dipetakan ke tabel dengan nama yang sama.

Contoh ini kemudian menjalankan operasi kueri dan pindai berikut menggunakan instans DynamoDBMapper.

- Dapatkan buku dengan Id.

Tabel ProductCatalog memiliki Id sebagai kunci primernya. Tabel tersebut tidak memiliki kunci urutan sebagai bagian dari kunci primernya. Oleh karena itu, Anda tidak dapat mengkueri tabel. Anda bisa mendapatkan item menggunakan nilai Id-nya.

- Jalankan kueri berikut terhadap tabel Reply.

Kunci primer tabel Reply terdiri dari atribut Id dan ReplyDateTime. ReplyDateTime adalah kunci urutan. Oleh karena itu, Anda dapat mengkueri tabel ini.

- Temukan balasan untuk utas forum yang diposting dalam 15 hari terakhir.
- Temukan balasan untuk utas forum yang diposting dalam rentang tanggal tertentu.
- Pindai tabel ProductCatalog untuk menemukan buku yang harganya kurang dari nilai yang ditentukan.

Untuk alasan performa, Anda harus menggunakan operasi kueri, bukan operasi pindai. Namun, Anda mungkin perlu memindai tabel. Misalkan ada kesalahan entri data dan salah satu harga buku diatur ke kurang dari 0. Contoh ini memindai tabel ProductCategory untuk menemukan item buku (ProductCategory adalah buku) yang harganya kurang dari 0.

- Lakukan pemindaian tabel ProductCatalog paralel untuk menemukan sepeda dari jenis tertentu.

Note

Contoh kode ini mengasumsikan bahwa Anda telah memuat data ke DynamoDB untuk akun Anda dengan mengikuti petunjuk di bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihat [Contoh kode Java](#).

Impor

```
import java.text.SimpleDateFormat;
import java.util.Date;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TimeZone;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBQueryExpression;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBRangeKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBScanExpression;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
```

Kode

```
public class DynamoDBMapperQueryScanExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

    public static void main(String[] args) throws Exception {
        try {

            DynamoDBMapper mapper = new DynamoDBMapper(client);

            // Get a book - Id=101
            GetBook(mapper, 101);
            // Sample forum and thread to test queries.
            String forumName = "Amazon DynamoDB";
            String threadSubject = "DynamoDB Thread 1";
            // Sample queries.
            FindRepliesInLast15Days(mapper, forumName, threadSubject);
            FindRepliesPostedWithinTimePeriod(mapper, forumName, threadSubject);

            // Scan a table and find book items priced less than specified
            // value.
            FindBooksPricedLessThanSpecifiedValue(mapper, "20");

            // Scan a table with multiple threads and find bicycle items with a
            // specified bicycle type
```



```
        int numberOfThreads = 16;
        FindBicyclesOfSpecificTypeWithMultipleThreads(mapper, numberOfThreads,
"Road");

        System.out.println("Example complete!");

    } catch (Throwable t) {
        System.err.println("Error running the DynamoDBMapperQueryScanExample: " +
t);
        t.printStackTrace();
    }
}

private static void GetBook(DynamoDBMapper mapper, int id) throws Exception {
    System.out.println("GetBook: Get book Id='101' ");
    System.out.println("Book table has no sort key. You can do GetItem, but not
Query.");
    Book book = mapper.load(Book.class, id);
    System.out.format("Id = %s Title = %s, ISBN = %s %n", book.getId(),
book.getTitle(), book.getISBN());
}

private static void FindRepliesInLast15Days(DynamoDBMapper mapper, String
forumName, String threadSubject)
    throws Exception {
    System.out.println("FindRepliesInLast15Days: Replies within last 15 days.");

    String partitionKey = forumName + "#" + threadSubject;

    long twoWeeksAgoMilli = (new Date()).getTime() - (15L * 24L * 60L * 60L *
1000L);
    Date twoWeeksAgo = new Date();
    twoWeeksAgo.setTime(twoWeeksAgoMilli);
    SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");
    dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));
    String twoWeeksAgoStr = dateFormatter.format(twoWeeksAgo);

    Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
    eav.put(":val1", new AttributeValue().withS(partitionKey));
    eav.put(":val2", new AttributeValue().withS(twoWeeksAgoStr.toString()));

    DynamoDBQueryExpression<Reply> queryExpression = new
DynamoDBQueryExpression<Reply>()
```

```
        .withKeyConditionExpression("Id = :val1 and ReplyDateTime  
> :val2").withExpressionAttributeValues(eav);  
  
        List<Reply> latestReplies = mapper.query(Reply.class, queryExpression);  
  
        for (Reply reply : latestReplies) {  
            System.out.format("Id=%s, Message=%s, PostedBy=%s %n, ReplyDateTime=%s %n",  
reply.getId(),  
                reply.getMessage(), reply.getPostedBy(), reply.getReplyDateTime());  
        }  
    }  
  
    private static void FindRepliesPostedWithinTimePeriod(DynamoDBMapper mapper, String  
forumName, String threadSubject)  
        throws Exception {  
        String partitionKey = forumName + "#" + threadSubject;  
  
        System.out.println(  
            "FindRepliesPostedWithinTimePeriod: Find replies for thread Message =  
'DynamoDB Thread 2' posted within a period.");  
        long startDateMilli = (new Date()).getTime() - (14L * 24L * 60L * 60L *  
1000L); // Two  
  
// weeks  
  
// ago.  
        long endDateMilli = (new Date()).getTime() - (7L * 24L * 60L * 60L * 1000L); //  
One  
                                                                    //  
week  
                                                                    //  
ago.  
        SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-  
dd'T'HH:mm:ss.SSS'Z'");  
        dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));  
        String startDate = dateFormatter.format(startDateMilli);  
        String endDate = dateFormatter.format(endDateMilli);  
  
        Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();  
        eav.put(":val1", new AttributeValue().withS(partitionKey));  
        eav.put(":val2", new AttributeValue().withS(startDate));  
        eav.put(":val3", new AttributeValue().withS(endDate));  
    }  
}
```

```
        DynamoDBQueryExpression<Reply> queryExpression = new
DynamoDBQueryExpression<Reply>()
            .withKeyConditionExpression("Id = :val1 and ReplyDateTime between :val2
and :val3")
            .withExpressionAttributeValues(eav);

        List<Reply> betweenReplies = mapper.query(Reply.class, queryExpression);

        for (Reply reply : betweenReplies) {
            System.out.format("Id=%s, Message=%s, PostedBy=%s %n, PostedDateTime=%s
%n", reply.getId(),
                reply.getMessage(), reply.getPostedBy(), reply.getReplyDateTime());
        }
    }

    private static void FindBooksPricedLessThanSpecifiedValue(DynamoDBMapper mapper,
String value) throws Exception {

        System.out.println("FindBooksPricedLessThanSpecifiedValue: Scan
ProductCatalog.");

        Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
        eav.put(":val1", new AttributeValue().withN(value));
        eav.put(":val2", new AttributeValue().withS("Book"));

        DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
            .withFilterExpression("Price < :val1 and ProductCategory
= :val2").withExpressionAttributeValues(eav);

        List<Book> scanResult = mapper.scan(Book.class, scanExpression);

        for (Book book : scanResult) {
            System.out.println(book);
        }
    }

    private static void FindBicyclesOfSpecificTypeWithMultipleThreads(DynamoDBMapper
mapper, int numberOfThreads,
String bicycleType) throws Exception {

        System.out.println("FindBicyclesOfSpecificTypeWithMultipleThreads: Scan
ProductCatalog With Multiple Threads.");
        Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
```

```
eav.put(":val1", new AttributeValue().withS("Bicycle"));
eav.put(":val2", new AttributeValue().withS(bicycleType));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("ProductCategory = :val1 and BicycleType
= :val2")
    .withExpressionAttributeValues(eav);

List<Bicycle> scanResult = mapper.parallelScan(Bicycle.class, scanExpression,
numberOfThreads);
for (Bicycle bicycle : scanResult) {
    System.out.println(bicycle);
}

}

@DynamoDBTable(tableName = "ProductCatalog")
public static class Book {
    private int id;
    private String title;
    private String ISBN;
    private int price;
    private int pageCount;
    private String productCategory;
    private boolean inPublication;

    @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 = "Price")
public int getPrice() {
    return price;
}

public void setPrice(int price) {
    this.price = price;
}

@DynamoDBAttribute(attributeName = "PageCount")
public int getPageCount() {
    return pageCount;
}

public void setPageCount(int pageCount) {
    this.pageCount = pageCount;
}

@DynamoDBAttribute(attributeName = "ProductCategory")
public String getProductCategory() {
    return productCategory;
}

public void setProductCategory(String productCategory) {
    this.productCategory = productCategory;
}

@DynamoDBAttribute(attributeName = "InPublication")
public boolean getInPublication() {
    return inPublication;
}

public void setInPublication(boolean inPublication) {
    this.inPublication = inPublication;
}
```

```
    @Override
    public String toString() {
        return "Book [ISBN=" + ISBN + ", price=" + price + ", product category=" +
productCategory + ", id=" + id
        + ", title=" + title + "]";
    }

}

@DynamoDBTable(tableName = "ProductCatalog")
public static class Bicycle {
    private int id;
    private String title;
    private String description;
    private String bicycleType;
    private String brand;
    private int price;
    private List<String> color;
    private String productCategory;

    @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 = "Description")
    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
```

```
        this.description = description;
    }

    @DynamoDBAttribute(attributeName = "BicycleType")
    public String getBicycleType() {
        return bicycleType;
    }

    public void setBicycleType(String bicycleType) {
        this.bicycleType = bicycleType;
    }

    @DynamoDBAttribute(attributeName = "Brand")
    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    @DynamoDBAttribute(attributeName = "Price")
    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    @DynamoDBAttribute(attributeName = "Color")
    public List<String> getColor() {
        return color;
    }

    public void setColor(List<String> color) {
        this.color = color;
    }

    @DynamoDBAttribute(attributeName = "ProductCategory")
    public String getProductCategory() {
        return productCategory;
    }
}
```

```
    public void setProductCategory(String productCategory) {
        this.productCategory = productCategory;
    }

    @Override
    public String toString() {
        return "Bicycle [Type=" + bicycleType + ", color=" + color + ", price=" +
price + ", product category="
        + productCategory + ", id=" + id + ", title=" + title + "]);
    }

}

@DynamoDBTable(tableName = "Reply")
public static class Reply {
    private String id;
    private String replyDateTime;
    private String message;
    private String postedBy;

    // Partition key
    @DynamoDBHashKey(attributeName = "Id")
    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    // Range key
    @DynamoDBRangeKey(attributeName = "ReplyDateTime")
    public String getReplyDateTime() {
        return replyDateTime;
    }

    public void setReplyDateTime(String replyDateTime) {
        this.replyDateTime = replyDateTime;
    }

    @DynamoDBAttribute(attributeName = "Message")
    public String getMessage() {
        return message;
    }
}
```



```
    public void setMessage(String message) {
        this.message = message;
    }

    @DynamoDBAttribute(attributeName = "PostedBy")
    public String getPostedBy() {
        return postedBy;
    }

    public void setPostedBy(String postedBy) {
        this.postedBy = postedBy;
    }
}

@dynamoDBTable(tableName = "Thread")
public static class Thread {
    private String forumName;
    private String subject;
    private String message;
    private String lastPostedDateTime;
    private String lastPostedBy;
    private Set<String> tags;
    private int answered;
    private int views;
    private int replies;

    // Partition key
    @DynamoDBHashKey(attributeName = "ForumName")
    public String getForumName() {
        return forumName;
    }

    public void setForumName(String forumName) {
        this.forumName = forumName;
    }

    // Range key
    @DynamoDBRangeKey(attributeName = "Subject")
    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
```

```
        this.subject = subject;
    }

    @DynamoDBAttribute(attributeName = "Message")
    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    @DynamoDBAttribute(attributeName = "LastPostedDateTime")
    public String getLastPostedDateTime() {
        return lastPostedDateTime;
    }

    public void setLastPostedDateTime(String lastPostedDateTime) {
        this.lastPostedDateTime = lastPostedDateTime;
    }

    @DynamoDBAttribute(attributeName = "LastPostedBy")
    public String getLastPostedBy() {
        return lastPostedBy;
    }

    public void setLastPostedBy(String lastPostedBy) {
        this.lastPostedBy = lastPostedBy;
    }

    @DynamoDBAttribute(attributeName = "Tags")
    public Set<String> getTags() {
        return tags;
    }

    public void setTags(Set<String> tags) {
        this.tags = tags;
    }

    @DynamoDBAttribute(attributeName = "Answered")
    public int getAnswered() {
        return answered;
    }
}
```

```
    public void setAnswered(int answered) {
        this.answered = answered;
    }

    @DynamoDBAttribute(attributeName = "Views")
    public int getViews() {
        return views;
    }

    public void setViews(int views) {
        this.views = views;
    }

    @DynamoDBAttribute(attributeName = "Replies")
    public int getReplies() {
        return replies;
    }

    public void setReplies(int replies) {
        this.replies = replies;
    }
}

@DynamoDBTable(tableName = "Forum")
public static class Forum {
    private String name;
    private String category;
    private int threads;

    @DynamoDBHashKey(attributeName = "Name")
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @DynamoDBAttribute(attributeName = "Category")
    public String getCategory() {
        return category;
    }
}
```

```
    public void setCategory(String category) {
        this.category = category;
    }

    @DynamoDBAttribute(attributeName = "Threads")
    public int getThreads() {
        return threads;
    }

    public void setThreads(int threads) {
        this.threads = threads;
    }
}
}
```

Operasi batch DynamoDBMapper

Contoh kode Java berikut menyatakan kelas Book, Forum, Thread, dan Reply serta memetakannya ke tabel Amazon DynamoDB menggunakan kelas DynamoDBMapper.

Kode menggambarkan operasi tulis batch berikut:

- `batchSave` untuk memasukkan item buku di tabel `ProductCatalog`.
- `batchDelete` untuk menghapus item dari tabel `ProductCatalog`.
- `batchWrite` untuk memasukkan dan menghapus item dari tabel `Forum` dan `Thread`.

Untuk informasi selengkapnya tentang tabel yang digunakan dalam contoh ini, lihat [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#). Untuk step-by-step instruksi untuk menguji contoh berikut, lihat [Contoh kode Java](#).

Impor

```
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
```

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapperConfig;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBRangeKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
```

Kode

```
public class DynamoDBMapperBatchWriteExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

    public static void main(String[] args) throws Exception {
        try {

            DynamoDBMapper mapper = new DynamoDBMapper(client);

            testBatchSave(mapper);
            testBatchDelete(mapper);
            testBatchWrite(mapper);

            System.out.println("Example complete!");

        } catch (Throwable t) {
            System.err.println("Error running the DynamoDBMapperBatchWriteExample: " +
t);
            t.printStackTrace();
        }
    }

    private static void testBatchSave(DynamoDBMapper mapper) {

        Book book1 = new Book();
        book1.setId(901);
        book1.setInPublication(true);
        book1.setISBN("902-11-11-1111");
        book1.setPageCount(100);
        book1.setPrice(10);
        book1.setProductCategory("Book");
        book1.setTitle("My book created in batch write");
```

```
    Book book2 = new Book();
    book2.setId(902);
    book2.setInPublication(true);
    book2.setISBN("902-11-12-1111");
    book2.setPageCount(200);
    book2.setPrice(20);
    book2.setProductCategory("Book");
    book2.setTitle("My second book created in batch write");

    Book book3 = new Book();
    book3.setId(903);
    book3.setInPublication(false);
    book3.setISBN("902-11-13-1111");
    book3.setPageCount(300);
    book3.setPrice(25);
    book3.setProductCategory("Book");
    book3.setTitle("My third book created in batch write");

    System.out.println("Adding three books to ProductCatalog table.");
    mapper.batchSave(Arrays.asList(book1, book2, book3));
}

private static void testBatchDelete(DynamoDBMapper mapper) {

    Book book1 = mapper.load(Book.class, 901);
    Book book2 = mapper.load(Book.class, 902);
    System.out.println("Deleting two books from the ProductCatalog table.");
    mapper.batchDelete(Arrays.asList(book1, book2));
}

private static void testBatchWrite(DynamoDBMapper mapper) {

    // Create Forum item to save
    Forum forumItem = new Forum();
    forumItem.setName("Test BatchWrite Forum");
    forumItem.setThreads(0);
    forumItem.setCategory("Amazon Web Services");

    // Create Thread item to save
    Thread threadItem = new Thread();
    threadItem.setForumName("AmazonDynamoDB");
    threadItem.setSubject("My sample question");
    threadItem.setMessage("BatchWrite message");
    List<String> tags = new ArrayList<String>();
```

```
tags.add("batch operations");
tags.add("write");
threadItem.setTags(new HashSet<String>(tags));

// Load ProductCatalog item to delete
Book book3 = mapper.load(Book.class, 903);

List<Object> objectsToWrite = Arrays.asList(forumItem, threadItem);
List<Book> objectsToDelete = Arrays.asList(book3);

DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER)
    .build();

mapper.batchWrite(objectsToWrite, objectsToDelete, config);
}

@DynamoDBTable(tableName = "ProductCatalog")
public static class Book {
    private int id;
    private String title;
    private String ISBN;
    private int price;
    private int pageCount;
    private String productCategory;
    private boolean inPublication;

    // 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 = "Price")
    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    @DynamoDBAttribute(attributeName = "PageCount")
    public int getPageCount() {
        return pageCount;
    }

    public void setPageCount(int pageCount) {
        this.pageCount = pageCount;
    }

    @DynamoDBAttribute(attributeName = "ProductCategory")
    public String getProductCategory() {
        return productCategory;
    }

    public void setProductCategory(String productCategory) {
        this.productCategory = productCategory;
    }

    @DynamoDBAttribute(attributeName = "InPublication")
    public boolean getInPublication() {
        return inPublication;
    }

    public void setInPublication(boolean inPublication) {
```



```
        this.inPublication = inPublication;
    }

    @Override
    public String toString() {
        return "Book [ISBN=" + ISBN + ", price=" + price + ", product category=" +
productCategory + ", id=" + id
            + ", title=" + title + "]";
    }

}

@DynamoDBTable(tableName = "Reply")
public static class Reply {
    private String id;
    private String replyDateTime;
    private String message;
    private String postedBy;

    // Partition key
    @DynamoDBHashKey(attributeName = "Id")
    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    // Sort key
    @DynamoDBRangeKey(attributeName = "ReplyDateTime")
    public String getReplyDateTime() {
        return replyDateTime;
    }

    public void setReplyDateTime(String replyDateTime) {
        this.replyDateTime = replyDateTime;
    }

    @DynamoDBAttribute(attributeName = "Message")
    public String getMessage() {
        return message;
    }
}
```

```
    public void setMessage(String message) {
        this.message = message;
    }

    @DynamoDBAttribute(attributeName = "PostedBy")
    public String getPostedBy() {
        return postedBy;
    }

    public void setPostedBy(String postedBy) {
        this.postedBy = postedBy;
    }
}

@DynamoDBTable(tableName = "Thread")
public static class Thread {
    private String forumName;
    private String subject;
    private String message;
    private String lastPostedDateTime;
    private String lastPostedBy;
    private Set<String> tags;
    private int answered;
    private int views;
    private int replies;

    // Partition key
    @DynamoDBHashKey(attributeName = "ForumName")
    public String getForumName() {
        return forumName;
    }

    public void setForumName(String forumName) {
        this.forumName = forumName;
    }

    // Sort key
    @DynamoDBRangeKey(attributeName = "Subject")
    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }
}
```

```
}

    @DynamoDBAttribute(attributeName = "Message")
    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    @DynamoDBAttribute(attributeName = "LastPostedDateTime")
    public String getLastPostedDateTime() {
        return lastPostedDateTime;
    }

    public void setLastPostedDateTime(String lastPostedDateTime) {
        this.lastPostedDateTime = lastPostedDateTime;
    }

    @DynamoDBAttribute(attributeName = "LastPostedBy")
    public String getLastPostedBy() {
        return lastPostedBy;
    }

    public void setLastPostedBy(String lastPostedBy) {
        this.lastPostedBy = lastPostedBy;
    }

    @DynamoDBAttribute(attributeName = "Tags")
    public Set<String> getTags() {
        return tags;
    }

    public void setTags(Set<String> tags) {
        this.tags = tags;
    }

    @DynamoDBAttribute(attributeName = "Answered")
    public int getAnswered() {
        return answered;
    }

    public void setAnswered(int answered) {
```

```
        this.answered = answered;
    }

    @DynamoDBAttribute(attributeName = "Views")
    public int getViews() {
        return views;
    }

    public void setViews(int views) {
        this.views = views;
    }

    @DynamoDBAttribute(attributeName = "Replies")
    public int getReplies() {
        return replies;
    }

    public void setReplies(int replies) {
        this.replies = replies;
    }
}

@dynamoDBTable(tableName = "Forum")
public static class Forum {
    private String name;
    private String category;
    private int threads;

    // Partition key
    @DynamoDBHashKey(attributeName = "Name")
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @DynamoDBAttribute(attributeName = "Category")
    public String getCategory() {
        return category;
    }
}
```

```
    public void setCategory(String category) {
        this.category = category;
    }

    @DynamoDBAttribute(attributeName = "Threads")
    public int getThreads() {
        return threads;
    }

    public void setThreads(int threads) {
        this.threads = threads;
    }
}
}
```

Operasi Transaksi DynamoDBMapper

Contoh kode Java berikut menyatakan kelas Forum dan Thread, serta memetakannya ke tabel DynamoDB menggunakan kelas DynamoDBMapper.

Kode tersebut menggambarkan operasi transaksional berikut:

- `transactionWrite` untuk menambahkan, memperbarui, dan menghapus beberapa item dari satu atau beberapa tabel dalam satu transaksi.
- `transactionLoad` untuk mengambil beberapa item dari satu atau beberapa tabel dalam satu transaksi.

Impor

```
import java.util.ArrayList;
import java.util.List;
import java.util.Set;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMappingException;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBRangeKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
```

```
import
  com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTransactionLoadExpression;
import
  com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTransactionWriteExpression;
import com.amazonaws.services.dynamodbv2.datamodeling.TransactionLoadRequest;
import com.amazonaws.services.dynamodbv2.datamodeling.TransactionWriteRequest;
import com.amazonaws.services.dynamodbv2.model.InternalServerErrorException;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import com.amazonaws.services.dynamodbv2.model.TransactionCanceledException;
```

Kode

```
public class DynamoDBMapperTransactionExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDBMapper mapper;

    public static void main(String[] args) throws Exception {
        try {

            mapper = new DynamoDBMapper(client);

            testPutAndUpdateInTransactionWrite();
            testPutWithConditionalUpdateInTransactionWrite();
            testPutWithConditionCheckInTransactionWrite();
            testMixedOperationsInTransactionWrite();
            testTransactionLoadWithSave();
            testTransactionLoadWithTransactionWrite();
            System.out.println("Example complete");

        } catch (Throwable t) {
            System.err.println("Error running the
DynamoDBMapperTransactionWriteExample: " + t);
            t.printStackTrace();
        }
    }

    private static void testTransactionLoadWithSave() {
        // Create new Forum item for DynamoDB using save
        Forum dynamodbForum = new Forum();
        dynamodbForum.setName("DynamoDB Forum");
        dynamodbForum.setCategory("Amazon Web Services");
        dynamodbForum.setThreads(0);
    }
}
```

```
mapper.save(dynamodbForum);

// Add a thread to DynamoDB Forum
Thread dynamodbForumThread = new Thread();
dynamodbForumThread.setForumName("DynamoDB Forum");
dynamodbForumThread.setSubject("Sample Subject 1");
dynamodbForumThread.setMessage("Sample Question 1");
mapper.save(dynamodbForumThread);

// Update DynamoDB Forum to reflect updated thread count
dynamodbForum.setThreads(1);
mapper.save(dynamodbForum);

// Read DynamoDB Forum item and Thread item at the same time in a serializable
// manner
TransactionLoadRequest transactionLoadRequest = new TransactionLoadRequest();

// Read entire item for DynamoDB Forum
transactionLoadRequest.addLoad(dynamodbForum);

// Only read subject and message attributes from Thread item
DynamoDBTransactionLoadExpression loadExpressionForThread = new
DynamoDBTransactionLoadExpression()
    .withProjectionExpression("Subject, Message");
transactionLoadRequest.addLoad(dynamodbForumThread, loadExpressionForThread);

// Loaded objects are guaranteed to be in same order as the order in which they
// are
// added to TransactionLoadRequest
List<Object> loadedObjects = executeTransactionLoad(transactionLoadRequest);
Forum loadedDynamoDBForum = (Forum) loadedObjects.get(0);
System.out.println("Forum: " + loadedDynamoDBForum.getName());
System.out.println("Threads: " + loadedDynamoDBForum.getThreads());
Thread loadedDynamodbForumThread = (Thread) loadedObjects.get(1);
System.out.println("Subject: " + loadedDynamodbForumThread.getSubject());
System.out.println("Message: " + loadedDynamodbForumThread.getMessage());
}

private static void testTransactionLoadWithTransactionWrite() {
    // Create new Forum item for DynamoDB using save
    Forum dynamodbForum = new Forum();
    dynamodbForum.setName("DynamoDB New Forum");
    dynamodbForum.setCategory("Amazon Web Services");
    dynamodbForum.setThreads(0);
}
```

```
mapper.save(dynamodbForum);

// Update Forum item for DynamoDB and add a thread to DynamoDB Forum, in
// an ACID manner using transactionWrite

dynamodbForum.setThreads(1);
Thread dynamodbForumThread = new Thread();
dynamodbForumThread.setForumName("DynamoDB New Forum");
dynamodbForumThread.setSubject("Sample Subject 2");
dynamodbForumThread.setMessage("Sample Question 2");
TransactionWriteRequest transactionWriteRequest = new
TransactionWriteRequest();
transactionWriteRequest.addPut(dynamodbForumThread);
transactionWriteRequest.addUpdate(dynamodbForum);
executeTransactionWrite(transactionWriteRequest);

// Read DynamoDB Forum item and Thread item at the same time in a serializable
// manner
TransactionLoadRequest transactionLoadRequest = new TransactionLoadRequest();

// Read entire item for DynamoDB Forum
transactionLoadRequest.addLoad(dynamodbForum);

// Only read subject and message attributes from Thread item
DynamoDBTransactionLoadExpression loadExpressionForThread = new
DynamoDBTransactionLoadExpression()
    .withProjectionExpression("Subject, Message");
transactionLoadRequest.addLoad(dynamodbForumThread, loadExpressionForThread);

// Loaded objects are guaranteed to be in same order as the order in which they
// are
// added to TransactionLoadRequest
List<Object> loadedObjects = executeTransactionLoad(transactionLoadRequest);
Forum loadedDynamoDBForum = (Forum) loadedObjects.get(0);
System.out.println("Forum: " + loadedDynamoDBForum.getName());
System.out.println("Threads: " + loadedDynamoDBForum.getThreads());
Thread loadedDynamodbForumThread = (Thread) loadedObjects.get(1);
System.out.println("Subject: " + loadedDynamodbForumThread.getSubject());
System.out.println("Message: " + loadedDynamodbForumThread.getMessage());
}

private static void testPutAndUpdateInTransactionWrite() {
    // Create new Forum item for S3 using save
    Forum s3Forum = new Forum();
```



```
s3Forum.setName("S3 Forum");
s3Forum.setCategory("Core Amazon Web Services");
s3Forum.setThreads(0);
mapper.save(s3Forum);

// Update Forum item for S3 and Create new Forum item for DynamoDB using
// transactionWrite
s3Forum.setCategory("Amazon Web Services");
Forum dynamodbForum = new Forum();
dynamodbForum.setName("DynamoDB Forum");
dynamodbForum.setCategory("Amazon Web Services");
dynamodbForum.setThreads(0);
TransactionWriteRequest transactionWriteRequest = new
TransactionWriteRequest();
transactionWriteRequest.addUpdate(s3Forum);
transactionWriteRequest.addPut(dynamodbForum);
executeTransactionWrite(transactionWriteRequest);
}

private static void testPutWithConditionalUpdateInTransactionWrite() {
    // Create new Thread item for DynamoDB forum and update thread count in
DynamoDB
    // forum
    // if the DynamoDB Forum exists
    Thread dynamodbForumThread = new Thread();
    dynamodbForumThread.setForumName("DynamoDB Forum");
    dynamodbForumThread.setSubject("Sample Subject 1");
    dynamodbForumThread.setMessage("Sample Question 1");

    Forum dynamodbForum = new Forum();
    dynamodbForum.setName("DynamoDB Forum");
    dynamodbForum.setCategory("Amazon Web Services");
    dynamodbForum.setThreads(1);

    DynamoDBTransactionWriteExpression transactionWriteExpression = new
DynamoDBTransactionWriteExpression()
        .withConditionExpression("attribute_exists(Category)");

    TransactionWriteRequest transactionWriteRequest = new
TransactionWriteRequest();
    transactionWriteRequest.addPut(dynamodbForumThread);
    transactionWriteRequest.addUpdate(dynamodbForum, transactionWriteExpression);
    executeTransactionWrite(transactionWriteRequest);
}
```

```
private static void testPutWithConditionCheckInTransactionWrite() {
    // Create new Thread item for DynamoDB forum and update thread count in
DynamoDB
    // forum if a thread already exists
    Thread dynamodbForumThread2 = new Thread();
    dynamodbForumThread2.setForumName("DynamoDB Forum");
    dynamodbForumThread2.setSubject("Sample Subject 2");
    dynamodbForumThread2.setMessage("Sample Question 2");

    Thread dynamodbForumThread1 = new Thread();
    dynamodbForumThread1.setForumName("DynamoDB Forum");
    dynamodbForumThread1.setSubject("Sample Subject 1");
    DynamoDBTransactionWriteExpression conditionExpressionForConditionCheck = new
DynamoDBTransactionWriteExpression()
        .withConditionExpression("attribute_exists(Subject)");

    Forum dynamodbForum = new Forum();
    dynamodbForum.setName("DynamoDB Forum");
    dynamodbForum.setCategory("Amazon Web Services");
    dynamodbForum.setThreads(2);

    TransactionWriteRequest transactionWriteRequest = new
TransactionWriteRequest();
    transactionWriteRequest.addPut(dynamodbForumThread2);
    transactionWriteRequest.addConditionCheck(dynamodbForumThread1,
conditionExpressionForConditionCheck);
    transactionWriteRequest.addUpdate(dynamodbForum);
    executeTransactionWrite(transactionWriteRequest);
}

private static void testMixedOperationsInTransactionWrite() {
    // Create new Thread item for S3 forum and delete "Sample Subject 1" Thread
from
    // DynamoDB forum if
    // "Sample Subject 2" Thread exists in DynamoDB forum
    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);

Thread dynamodbForumThread1 = new Thread();
dynamodbForumThread1.setForumName("DynamoDB Forum");
dynamodbForumThread1.setSubject("Sample Subject 1");

Thread dynamodbForumThread2 = new Thread();
dynamodbForumThread2.setForumName("DynamoDB Forum");
dynamodbForumThread2.setSubject("Sample Subject 2");
DynamoDBTransactionWriteExpression conditionExpressionForConditionCheck = new
DynamoDBTransactionWriteExpression()
    .withConditionExpression("attribute_exists(Subject)");

Forum dynamodbForum = new Forum();
dynamodbForum.setName("DynamoDB Forum");
dynamodbForum.setCategory("Amazon Web Services");
dynamodbForum.setThreads(1);

TransactionWriteRequest transactionWriteRequest = new
TransactionWriteRequest();
transactionWriteRequest.addPut(s3ForumThread);
transactionWriteRequest.addUpdate(s3Forum);
transactionWriteRequest.addDelete(dynamodbForumThread1);
transactionWriteRequest.addConditionCheck(dynamodbForumThread2,
conditionExpressionForConditionCheck);
transactionWriteRequest.addUpdate(dynamodbForum);
executeTransactionWrite(transactionWriteRequest);
}

private static List<Object> executeTransactionLoad(TransactionLoadRequest
transactionLoadRequest) {
    List<Object> loadedObjects = new ArrayList<Object>();
    try {
        loadedObjects = mapper.transactionLoad(transactionLoadRequest);
    } catch (DynamoDBMappingException ddbme) {
        System.err.println("Client side error in Mapper, fix before retrying.
Error: " + ddbme.getMessage());
    } catch (ResourceNotFoundException rnf) {
        System.err.println("One of the tables was not found, verify table exists
before retrying. Error: "
            + rnf.getMessage());
    } catch (InternalServerErrorException ise) {
        System.err.println(
```

```

                "Internal Server Error, generally safe to retry with back-off.
Error: " + ise.getMessage());
    } catch (TransactionCanceledException tce) {
        System.err.println(
            "Transaction Canceled, implies a client issue, fix before retrying.
Error: " + tce.getMessage());
    } catch (Exception ex) {
        System.err.println(
            "An exception occurred, investigate and configure retry strategy.
Error: " + ex.getMessage());
    }
    return loadedObjects;
}

private static void executeTransactionWrite(TransactionWriteRequest
transactionWriteRequest) {
    try {
        mapper.transactionWrite(transactionWriteRequest);
    } catch (DynamoDBMappingException ddbme) {
        System.err.println("Client side error in Mapper, fix before retrying.
Error: " + ddbme.getMessage());
    } catch (ResourceNotFoundException rnfe) {
        System.err.println("One of the tables was not found, verify table exists
before retrying. Error: "
            + rnfe.getMessage());
    } catch (InternalServerErrorException ise) {
        System.err.println(
            "Internal Server Error, generally safe to retry with back-off.
Error: " + ise.getMessage());
    } catch (TransactionCanceledException tce) {
        System.err.println(
            "Transaction Canceled, implies a client issue, fix before retrying.
Error: " + tce.getMessage());
    } catch (Exception ex) {
        System.err.println(
            "An exception occurred, investigate and configure retry strategy.
Error: " + ex.getMessage());
    }
}

@DynamoDBTable(tableName = "Thread")
public static class Thread {
    private String forumName;
    private String subject;

```

```
private String message;
private String lastPostedDateTime;
private String lastPostedBy;
private Set<String> tags;
private int answered;
private int views;
private int replies;

// Partition key
@DynamoDBHashKey(attributeName = "ForumName")
public String getForumName() {
    return forumName;
}

public void setForumName(String forumName) {
    this.forumName = forumName;
}

// Sort key
@DynamoDBRangeKey(attributeName = "Subject")
public String getSubject() {
    return subject;
}

public void setSubject(String subject) {
    this.subject = subject;
}

@DynamoDBAttribute(attributeName = "Message")
public String getMessage() {
    return message;
}

public void setMessage(String message) {
    this.message = message;
}

@DynamoDBAttribute(attributeName = "LastPostedDateTime")
public String getLastPostedDateTime() {
    return lastPostedDateTime;
}

public void setLastPostedDateTime(String lastPostedDateTime) {
    this.lastPostedDateTime = lastPostedDateTime;
}
```

```
    }

    @DynamoDBAttribute(attributeName = "LastPostedBy")
    public String getLastPostedBy() {
        return lastPostedBy;
    }

    public void setLastPostedBy(String lastPostedBy) {
        this.lastPostedBy = lastPostedBy;
    }

    @DynamoDBAttribute(attributeName = "Tags")
    public Set<String> getTags() {
        return tags;
    }

    public void setTags(Set<String> tags) {
        this.tags = tags;
    }

    @DynamoDBAttribute(attributeName = "Answered")
    public int getAnswered() {
        return answered;
    }

    public void setAnswered(int answered) {
        this.answered = answered;
    }

    @DynamoDBAttribute(attributeName = "Views")
    public int getViews() {
        return views;
    }

    public void setViews(int views) {
        this.views = views;
    }

    @DynamoDBAttribute(attributeName = "Replies")
    public int getReplies() {
        return replies;
    }

    public void setReplies(int replies) {
```

```
        this.replies = replies;
    }

}

@DynamoDBTable(tableName = "Forum")
public static class Forum {
    private String name;
    private String category;
    private int threads;

    // Partition key
    @DynamoDBHashKey(attributeName = "Name")
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @DynamoDBAttribute(attributeName = "Category")
    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    @DynamoDBAttribute(attributeName = "Threads")
    public int getThreads() {
        return threads;
    }

    public void setThreads(int threads) {
        this.threads = threads;
    }
}
}
```

Java 2.x: DynamoDB Ditingkatkan Klien

DynamoDB ditingkatkan klien adalah perpustakaan tingkat tinggi yang merupakan bagian dari AWS SDK for Java versi 2 (v2). Kelas ini menawarkan cara mudah untuk memetakan kelas DynamoDB. Anda menentukan hubungan antara tabel dan kelas model yang sesuai dalam kode Anda. Setelah Anda menentukan hubungan tersebut, Anda dapat melakukan berbagai operasi buat, baca, perbarui, atau hapus (CRUD) pada tabel atau item di tabel atau item di tabel atau item di tabel atau item di tabel atau item di tabel atau item di tabel atau item di tabel atau item di tabel atau item di tabel atau item di tabel atau item

Untuk informasi selengkapnya tentang bagaimana Anda dapat menggunakan klien yang disempurnakan dengan DynamoDB, lihat [Menggunakan DynamoDB Enhanced Client di 2.x](#). AWS SDK for Java

.NET: Model dokumen

AWS SDK for .NET menyediakan kelas model dokumen yang membungkus beberapa operasi Amazon DynamoDB tingkat rendah, yang kemudian memudahkan pengodean Anda. Dalam model dokumen, kelas utamanya adalah `Table` dan `Document`. Kelas `Table` menyediakan metode operasi data seperti `PutItem`, `GetItem`, dan `DeleteItem`. Kelas ini juga menyediakan metode `Query` dan `Scan`. Kelas `Document` mewakili satu item dalam tabel.

Kelas model dokumen sebelumnya tersedia di namespace `Amazon.DynamoDBv2.DocumentModel`.

Note

Anda tidak dapat menggunakan kelas model dokumen untuk membuat, memperbarui, dan menghapus tabel. Namun, model dokumen mendukung operasi data yang paling umum.

Topik

- [Jenis data yang didukung](#)
- [Menggunakan item di DynamoDB dengan model dokumen AWS SDK for .NET](#)
- [Contoh: Operasi CRUD menggunakan model dokumen AWS SDK for .NET](#)
- [Contoh: Operasi batch menggunakan API model dokumen AWS SDK for .NET](#)
- [Menggunakan tabel di DynamoDB dengan model dokumen AWS SDK for .NET](#)

Jenis data yang didukung

Model dokumen mendukung serangkaian jenis data .NET primitif dan jenis data koleksi. Model ini mendukung jenis data primitif berikut.

- bool
- byte
- char
- DateTime
- decimal
- double
- float
- Guid
- Int16
- Int32
- Int64
- SByte
- string
- UInt16
- UInt32
- UInt64

Tabel berikut merangkum pemetaan jenis .NET sebelumnya untuk jenis DynamoDB.

Jenis .NET primitif	Jenis DynamoDB
Semua jenis angka	N (jenis angka)
Semua jenis string	S (jenis string)
MemoryStream, byte []	B (jenis biner)
bool	N (jenis angka). 0 mewakili false dan 1 mewakili true.

Jenis .NET primitif	Jenis DynamoDB
DateTime	S (jenis string). Nilai DateTime disimpan sebagai string berformat ISO-8601.
Guid	S (jenis string).
Jenis koleksi (Daftar, HashSet, dan array)	Jenis BS (set biner), jenis SS (set string), dan jenis NS (set angka)

AWS SDK for .NET mendefinisikan jenis untuk memetakan jenis Boolean, null, daftar dan peta DynamoDB untuk API model dokumen .NET:

- Gunakan `DynamoDBBool` untuk jenis Boolean.
- Gunakan `DynamoDBNull` untuk jenis null.
- Gunakan `DynamoDBList` untuk jenis daftar.
- Gunakan `Document` untuk jenis peta.

Note

- Nilai biner kosong didukung.
- Pembacaan nilai string kosong didukung. Nilai atribut string kosong didukung dalam nilai atribut string dari jenis Set saat menulis ke DynamoDB. Nilai atribut string kosong dari jenis string dan nilai string kosong dalam jenis Daftar atau Peta dihilangkan dari permintaan tulis

Menggunakan item di DynamoDB dengan model dokumen AWS SDK for .NET

Contoh kode Java berikut menunjukkan cara melakukan berbagai operasi dengan model dokumen AWS SDK for .NET. Anda dapat menggunakan contoh-contoh ini untuk melakukan operasi CRUD, batch, dan transaksi.

Topik

- [Menempatkan item - Tabel. PutItem metode](#)
- [Menentukan parameter opsional](#)

- [Mendapatkan item - Tabel. GetItem](#)
- [Menghapus item - Tabel. DeleteItem](#)
- [Memperbarui item - Tabel. UpdateItem](#)
- [Penulisan batch - menempatkan dan menghapus beberapa item](#)

Untuk melakukan operasi data menggunakan model dokumen, Anda harus memanggil metode `Table.LoadTable` terlebih dahulu, yang akan membuat instans dari kelas `Table` yang mewakili tabel tertentu. Contoh C# berikut membuat objek `Table` yang mewakili tabel `ProductCatalog` di Amazon DynamoDB.

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");
```

Note

Secara umum, Anda menggunakan metode `LoadTable` satu kali di awal aplikasi karena metode tersebut membuat panggilan `DescribeTable` yang akan menambah bolak-balik ke DynamoDB.

Kemudian, Anda dapat menggunakan objek `Table` untuk melakukan berbagai operasi data. Setiap operasi data memiliki dua jenis beban berlebih: Jenis pertama mengambil parameter minimum yang diperlukan dan jenis kedua mengambil informasi konfigurasi opsional khusus operasi. Misalnya, untuk mengambil item, Anda harus memberikan nilai kunci primer tabel, dalam hal ini Anda dapat menggunakan beban berlebih `GetItem` berikut.

Example

```
// Get the item from a table that has a primary key that is composed of only a
partition key.
Table.GetItem(Primitive partitionKey);
// Get the item from a table whose primary key is composed of both a partition key and
sort key.
Table.GetItem(Primitive partitionKey, Primitive sortKey);
```

Anda juga dapat meneruskan parameter opsional untuk metode ini. Misalnya, `GetItem` sebelumnya mengembalikan seluruh item termasuk semua atributnya. Anda dapat menentukan daftar atribut yang

akan diambil secara opsional. Dalam hal ini, Anda menggunakan beban berlebih `GetItem` berikut yang mengambil parameter objek konfigurasi khusus operasi.

Example

```
// Configuration object that specifies optional parameters.
GetItemOperationConfig config = new GetItemOperationConfig()
{
    AttributesToGet = new List<string>() { "Id", "Title" },
};
// Pass in the configuration to the GetItem method.
// 1. Table that has only a partition key as primary key.
Table.GetItem(Primitive partitionKey, GetItemOperationConfig config);
// 2. Table that has both a partition key and a sort key.
Table.GetItem(Primitive partitionKey, Primitive sortKey, GetItemOperationConfig
    config);
```

Anda dapat menggunakan objek konfigurasi untuk menentukan beberapa parameter opsional seperti meminta daftar atribut tertentu atau menentukan ukuran halaman (jumlah item per halaman). Setiap metode operasi data memiliki kelas konfigurasinya sendiri. Misalnya, Anda dapat menggunakan kelas `GetItemOperationConfig` untuk menyediakan opsi bagi operasi `GetItem`. Anda dapat menggunakan kelas `PutItemOperationConfig` untuk menyediakan parameter opsional bagi operasi `PutItem`.

Bagian berikut membahas setiap operasi data yang didukung oleh kelas `Table`.

Menempatkan item - Tabel. `PutItem` metode

Metode `PutItem` mengunggah instans `Document` input ke tabel. Jika item dengan kunci primer yang ditentukan dalam `Document` input ada dalam tabel, operasi `PutItem` akan menggantikan seluruh item yang ada. Item baru identik dengan objek `Document` yang Anda berikan untuk metode `PutItem`. Jika item asli Anda memiliki atribut tambahan, item tersebut tidak lagi berada di item baru.

Berikut adalah langkah-langkah untuk menempatkan item baru ke dalam tabel menggunakan model dokumen AWS SDK for .NET.

1. Jalankan metode `Table.LoadTable` yang menyediakan nama tabel tempat Anda ingin meletakkan item.
2. Buat objek `Document` yang berisi daftar nama atribut dan nilainya.

3. Jalankan `Table.PutItem` dengan menyediakan instans `Document` sebagai parameter.

Contoh kode #C berikut mendemonstrasikan tugas sebelumnya. Contoh tersebut mengunggah item ke tabel `ProductCatalog`.

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

var book = new Document();
book["Id"] = 101;
book["Title"] = "Book 101 Title";
book["ISBN"] = "11-11-11-11";
book["Authors"] = new List<string> { "Author 1", "Author 2" };
book["InStock"] = new DynamoDBBool(true);
book["QuantityOnHand"] = new DynamoDBNull();

table.PutItem(book);
```

Dalam contoh sebelumnya, `Document` instance membuat item yang memiliki `Number`, `String`, `String`, `Set Boolean`, dan `Null` atribut. (`Null` digunakan untuk menunjukkan bahwa `QuantityOnHand` untuk produk ini tidak diketahui.) Untuk `Boolean` dan `Null`, gunakan metode konstruktor `DynamoDBBool` dan `DynamoDBNull`.

Dalam `DynamoDB`, jenis data `List` dan `Map` dapat berisi elemen yang terdiri dari jenis data lainnya. Berikut adalah cara memetakan jenis data tersebut ke API model dokumen:

- Daftar — gunakan konstruktor `DynamoDBList`.
- Peta — gunakan konstruktor `Document`.

Anda dapat mengubah contoh sebelumnya untuk menambahkan atribut `List` ke item. Untuk melakukannya, gunakan konstruktor `DynamoDBList`, seperti yang ditunjukkan dalam contoh kode berikut.

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

var book = new Document();
```

```
book["Id"] = 101;

/*other attributes omitted for brevity...*/

var relatedItems = new DynamoDBList();
relatedItems.Add(341);
relatedItems.Add(472);
relatedItems.Add(649);
book.Add("RelatedItems", relatedItems);

table.PutItem(book);
```

Untuk menambahkan atribut Map ke buku, tentukan Document lainnya. Contoh kode berikut menunjukkan cara melakukannya.

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

var book = new Document();
book["Id"] = 101;

/*other attributes omitted for brevity...*/

var pictures = new Document();
pictures.Add("FrontView", "http://example.com/products/101_front.jpg" );
pictures.Add("RearView", "http://example.com/products/101_rear.jpg" );

book.Add("Pictures", pictures);

table.PutItem(book);
```

Contoh ini didasarkan pada item yang ditampilkan dalam [Menentukan atribut item saat menggunakan ekspresi](#). Model dokumen memungkinkan Anda membuat atribut bersarang kompleks, seperti atribut `ProductReviews` yang ditampilkan dalam studi kasus.

Menentukan parameter opsional

Anda dapat mengonfigurasi parameter opsional untuk operasi `PutItem` dengan menambahkan parameter `PutItemOperationConfig`. Untuk daftar lengkap parameter opsional, lihat [PutItem](#). Contoh kode C# berikut menempatkan item dalam tabel `ProductCatalog`. Contoh tersebut menentukan parameter opsional berikut:

- Parameter `ConditionalExpression` untuk menjadikannya sebagai permintaan penempatan bersyarat. Contoh ini membuat ekspresi yang menentukan atribut ISBN harus memiliki nilai tertentu yang harus ada dalam item yang diganti.

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

var book = new Document();
book["Id"] = 555;
book["Title"] = "Book 555 Title";
book["Price"] = "25.00";
book["ISBN"] = "55-55-55-55";
book["Name"] = "Item 1 updated";
book["Authors"] = new List<string> { "Author x", "Author y" };
book["InStock"] = new DynamoDBBool(true);
book["QuantityOnHand"] = new DynamoDBNull();

// Create a condition expression for the optional conditional put operation.
Expression expr = new Expression();
expr.ExpressionStatement = "ISBN = :val";
expr.ExpressionAttributeValues[":val"] = "55-55-55-55";

PutItemOperationConfig config = new PutItemOperationConfig()
{
    // Optional parameter.
    ConditionalExpression = expr
};

table.PutItem(book, config);
```

Mendapatkan item - Tabel. GetItem

Operasi `GetItem` mengambil item sebagai instans `Document`. Anda harus memberikan kunci primer item yang ingin diambil seperti yang ditunjukkan dalam kode contoh C# berikut.

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");
Document document = table.GetItem(101); // Primary key 101.
```

Operasi `GetItem` mengembalikan semua atribut item dan melakukan bacaan akhir konsisten (lihat [Konsistensi baca](#)) secara default.

Menentukan parameter opsional

Anda dapat mengonfigurasi opsi tambahan untuk operasi `GetItem` dengan menambahkan parameter `GetItemOperationConfig`. Untuk daftar lengkap parameter opsional, lihat [GetItem](#). Contoh kode C# berikut mengambil item dari tabel `ProductCatalog`. Contoh ini menentukan `GetItemOperationConfig` untuk menyediakan parameter opsional berikut:

- Parameter `AttributesToGet` hanya akan mengambil atribut yang ditentukan.
- Parameter `ConsistentRead` akan meminta nilai terbaru untuk semua atribut yang ditentukan. Untuk mempelajari selengkapnya tentang konsistensi data, lihat [Konsistensi baca](#).

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

GetItemOperationConfig config = new GetItemOperationConfig()
{
    AttributesToGet = new List<string>() { "Id", "Title", "Authors", "InStock",
    "QuantityOnHand" },
    ConsistentRead = true
};
Document doc = table.GetItem(101, config);
```

Ketika mengambil item menggunakan API model dokumen, Anda dapat mengakses elemen individu dalam objek `Document` yang ditampilkan, seperti ditunjukkan dalam contoh berikut.

Example

```
int id = doc["Id"].AsInt();
string title = doc["Title"].AsString();
List<string> authors = doc["Authors"].AsListOfString();
bool inStock = doc["InStock"].AsBoolean();
DynamoDBNull quantityOnHand = doc["QuantityOnHand"].AsDynamoDBNull();
```

Untuk atribut yang berupa jenis `List` atau `Map`, berikut adalah cara memetakan atribut ke API model dokumen:

- **List** — Gunakan metode `AsDynamoDBList`.
- **Map** — Gunakan metode `AsDocument`.

Contoh kode berikut menunjukkan bagaimana untuk mengambil `List` (`RelatedItems`) dan `Map` (`Gambar`) dari `Document` objek:

Example

```
DynamoDBList relatedItems = doc["RelatedItems"].AsDynamoDBList();  
  
Document pictures = doc["Pictures"].AsDocument();
```

Menghapus item - Tabel. `DeleteItem`

Operasi `DeleteItem` menghapus item dari tabel. Anda dapat meneruskan kunci primer item sebagai parameter. Jika sudah membaca item dan memiliki objek `Document` yang sesuai, Anda dapat meneruskannya sebagai parameter ke metode `DeleteItem`, seperti yang ditunjukkan dalam contoh kode C# berikut.

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");  
  
// Retrieve a book (a Document instance)  
Document document = table.GetItem(111);  
  
// 1) Delete using the Document instance.  
table.DeleteItem(document);  
  
// 2) Delete using the primary key.  
int partitionKey = 222;  
table.DeleteItem(partitionKey)
```

Menentukan parameter opsional

Anda dapat mengonfigurasi opsi tambahan untuk operasi `Delete` dengan menambahkan parameter `DeleteItemOperationConfig`. Untuk daftar lengkap parameter opsional, lihat [DeleteTable](#). Contoh kode C# berikut menentukan dua parameter opsional berikut:

- Parameter `ConditionalExpression` untuk memastikan bahwa item buku yang dihapus memiliki nilai khusus untuk atribut `ISBN`.

- Parameter `ReturnValues` untuk meminta bahwa metode `Delete` menampilkan item yang dihapus.

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");
int partitionKey = 111;

Expression expr = new Expression();
expr.ExpressionStatement = "ISBN = :val";
expr.ExpressionAttributeValues[":val"] = "11-11-11-11";

// Specify optional parameters for Delete operation.
DeleteItemOperationConfig config = new DeleteItemOperationConfig
{
    ConditionalExpression = expr,
    ReturnValues = ReturnValues.AllOldAttributes // This is the only supported value
    when using the document model.
};

// Delete the book.
Document d = table.DeleteItem(partitionKey, config);
```

Memperbarui item - Tabel. UpdateItem

Operasi `UpdateItem` memperbarui item yang ada jika ada. Jika item dengan kunci primer yang ditentukan tidak ditemukan, operasi `UpdateItem` akan menambahkan item baru.

Anda dapat menggunakan operasi `UpdateItem` untuk memperbarui nilai atribut yang ada, menambahkan atribut baru ke koleksi yang ada, atau menghapus atribut dari koleksi yang ada. Anda menyediakan pembaruan ini dengan membuat instans `Document` yang menjelaskan pembaruan yang ingin Anda lakukan.

Tindakan `UpdateItem` menggunakan panduan berikut:

- Jika item tidak ada, `UpdateItem` menambahkan item baru menggunakan kunci primer yang ditentukan dalam input.
- Jika item ada, `UpdateItem` menerapkan pembaruan sebagai berikut:
 - Menggantikan nilai atribut yang ada dengan nilai dalam pembaruan.

- Jika atribut yang Anda berikan pada input tidak ada, atribut baru akan ditambahkan ke item tersebut.
- Jika atribut input adalah null, atribut tersebut akan dihapus, jika ada.

Note

`UpdateItem` operasi tingkat menengah ini tidak mendukung `Add` tindakan (lihat [UpdateItem](#)) yang didukung oleh operasi DynamoDB yang mendasarinya.

Note

Operasi `PutItem` ([Menempatkan item - Tabel. PutItem metode](#)) juga dapat melakukan pembaruan. Jika Anda memanggil `PutItem` untuk mengunggah item dan kunci primer ada, operasi `PutItem` menggantikan seluruh item. Jika terdapat atribut dalam item yang ada yang tidak ditentukan dalam input `Document` yang ditempatkan, operasi `PutItem` akan menghapus atribut tersebut. Namun, `UpdateItem` hanya memperbarui atribut input yang ditentukan. Atribut lain item tersebut yang ada tidak akan berubah.

Berikut adalah langkah-langkah untuk memperbarui item menggunakan model dokumen AWS SDK for .NET:

1. Jalankan metode `Table.LoadTable` dengan memberikan nama tabel tempat Anda ingin melakukan operasi pembaruan.
2. Buat instans `Document` dengan menyediakan semua pembaruan yang ingin Anda lakukan.

Untuk menghapus atribut yang ada, tentukan nilai atribut sebagai null.

3. Panggil metode `Table.UpdateItem` dan berikan instans `Document` sebagai parameter input.

Anda harus memberikan kunci primer di instans `Document` maupun secara eksplisit sebagai parameter.

Contoh kode #C berikut mendemonstrasikan tugas sebelumnya. Contoh kode tersebut memperbarui item dalam tabel `Book`. Operasi `UpdateItem` memperbarui atribut `Authors` yang ada, menghapus

atribut PageCount, dan menambahkan atribut XYZ baru. Instans Document menyertakan kunci primer dari buku yang akan diperbarui.

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

var book = new Document();

// Set the attributes that you wish to update.
book["Id"] = 111; // Primary key.
// Replace the authors attribute.
book["Authors"] = new List<string> { "Author x", "Author y" };
// Add a new attribute.
book["XYZ"] = 12345;
// Delete the existing PageCount attribute.
book["PageCount"] = null;

table.Update(book);
```

Menentukan parameter opsional

Anda dapat mengonfigurasi opsi tambahan untuk operasi UpdateItem dengan menambahkan parameter UpdateItemOperationConfig. Untuk daftar lengkap parameter opsional, lihat [UpdateItem](#).

Contoh kode C# berikut memperbarui harga item buku menjadi 25. Contoh tersebut menentukan dua parameter opsional berikut:

- Parameter ConditionalExpression yang mengidentifikasi atribut Price dengan nilai 20 yang diharapkan ada.
- Parameter ReturnValues akan meminta operasi UpdateItem guna menampilkan item yang diperbarui.

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");
string partitionKey = "111";

var book = new Document();
```

```
book["Id"] = partitionKey;
book["Price"] = 25;

Expression expr = new Expression();
expr.ExpressionStatement = "Price = :val";
expr.ExpressionAttributeValues[":val"] = "20";

UpdateItemOperationConfig config = new UpdateItemOperationConfig()
{
    ConditionalExpression = expr,
    ReturnValues = ReturnValues.AllOldAttributes
};

Document d1 = table.Update(book, config);
```

Penulisan batch - menempatkan dan menghapus beberapa item

Penulisan batch merujuk pada penempatan dan penghapusan beberapa item dalam satu batch. Operasi memungkinkan Anda untuk menempatkan dan menghapus beberapa item dari satu atau beberapa tabel dalam satu panggilan. Berikut langkah-langkah untuk menempatkan atau menghapus beberapa item menggunakan API model dokumen AWS SDK for .NET.

1. Buat objek `Table` dengan menjalankan metode `Table.LoadTable` dengan memberikan nama tabel tempat Anda ingin melakukan operasi batch.
2. Jalankan metode `createBatchWrite` pada instans tabel yang Anda buat di langkah sebelumnya dan buat objek `DocumentBatchWrite`.
3. Gunakan metode objek `DocumentBatchWrite` untuk menentukan dokumen yang ingin Anda unggah atau hapus.
4. Panggil metode `DocumentBatchWrite.Execute` untuk menjalankan operasi batch.

Saat menggunakan API model dokumen, Anda dapat menentukan sejumlah operasi dalam batch. Namun, DynamoDB membatasi jumlah operasi dalam batch dan ukuran total batch dalam operasi batch. Untuk informasi selengkapnya tentang batasan spesifik, lihat [BatchWriteItem](#). Jika API model dokumen mendeteksi bahwa permintaan tulis batch melebihi jumlah permintaan tulis yang diizinkan, atau ukuran payload HTTP batch melebihi batas yang diizinkan oleh `BatchWriteItem`, batch tersebut akan dipecah menjadi beberapa batch yang lebih kecil. Selain itu, jika respons ke penulisan batch mengembalikan item yang belum diproses, API model dokumen otomatis mengirimkan permintaan batch lain dengan item yang belum diproses.

Contoh kode #C berikut menunjukkan langkah-langkah sebelumnya. Contoh tersebut menggunakan operasi penulisan batch untuk melakukan dua penulisan; mengunggah item buku dan menghapus item buku lainnya.

```
Table productCatalog = Table.LoadTable(client, "ProductCatalog");
var batchWrite = productCatalog.CreateBatchWrite();

var book1 = new Document();
book1["Id"] = 902;
book1["Title"] = "My book1 in batch write using .NET document model";
book1["Price"] = 10;
book1["Authors"] = new List<string> { "Author 1", "Author 2", "Author 3" };
book1["InStock"] = new DynamoDBBool(true);
book1["QuantityOnHand"] = 5;

batchWrite.AddDocumentToPut(book1);
// specify delete item using overload that takes PK.
batchWrite.AddKeyToDelete(12345);

batchWrite.Execute();
```

Untuk contoh pekerjaan, lihat [Contoh: Operasi batch menggunakan API model dokumen AWS SDK for .NET](#).

Anda dapat menggunakan operasi `batchWrite` untuk melakukan operasi penempatan dan penghapusan pada beberapa tabel. Berikut langkah-langkah untuk menempatkan atau menghapus beberapa item dari beberapa tabel menggunakan model dokumen AWS SDK for .NET.

1. Anda membuat instans `DocumentBatchWrite` untuk setiap tabel tempat Anda ingin menempatkan atau menghapus beberapa item, seperti yang dijelaskan dalam prosedur sebelumnya.
2. Buat instans `MultiTableDocumentBatchWrite` dan tambahkan objek `DocumentBatchWrite` individu ke instans tersebut.
3. Jalankan metode `MultiTableDocumentBatchWrite.Execute`.

Contoh kode #C berikut menunjukkan langkah-langkah sebelumnya. Contoh tersebut menggunakan operasi penulisan batch untuk melakukan operasi tulis berikut:

- Menempatkan item baru dalam item tabel Forum.

- Menempatkan item dalam tabel Thread dan menghapus item dari tabel yang sama.

```
// 1. Specify item to add in the Forum table.
Table forum = Table.LoadTable(client, "Forum");
var forumBatchWrite = forum.CreateBatchWrite();

var forum1 = new Document();
forum1["Name"] = "Test BatchWrite Forum";
forum1["Threads"] = 0;
forumBatchWrite.AddDocumentToPut(forum1);

// 2a. Specify item to add in the Thread table.
Table thread = Table.LoadTable(client, "Thread");
var threadBatchWrite = thread.CreateBatchWrite();

var thread1 = new Document();
thread1["ForumName"] = "Amazon S3 forum";
thread1["Subject"] = "My sample question";
thread1["Message"] = "Message text";
thread1["KeywordTags"] = new List<string>{ "Amazon S3", "Bucket" };
threadBatchWrite.AddDocumentToPut(thread1);

// 2b. Specify item to delete from the Thread table.
threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

// 3. Create multi-table batch.
var superBatch = new MultiTableDocumentBatchWrite();
superBatch.AddBatch(forumBatchWrite);
superBatch.AddBatch(threadBatchWrite);

superBatch.Execute();
```

Contoh: Operasi CRUD menggunakan model dokumen AWS SDK for .NET

Contoh kode C# berikut melakukan tindakan berikut:

- Membuat item buku di tabel ProductCatalog.
- Mengambil item buku.

- Memperbarui item buku. Contoh kode menunjukkan pembaruan normal yang menambahkan atribut baru dan memperbarui atribut yang ada. Contoh ini juga menunjukkan pembaruan bersyarat yang memperbarui harga buku hanya jika nilai harga yang ada sama seperti yang ditentukan dalam kode.
- Menghapus item buku.

Untuk step-by-step instruksi untuk menguji contoh berikut, lihat [Contoh kode .NET](#).

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class MidlevelItemCRUD
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        private static string tableName = "ProductCatalog";
        // The sample uses the following id PK value to add book item.
        private static int sampleBookId = 555;

        static void Main(string[] args)
        {
            try
            {
                Table productCatalog = Table.LoadTable(client, tableName);
                CreateBookItem(productCatalog);
                RetrieveBook(productCatalog);
                // Couple of sample updates.
                UpdateMultipleAttributes(productCatalog);
                UpdateBookPriceConditionally(productCatalog);

                // Delete.
                DeleteBook(productCatalog);
                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); }
    }

    // Creates a sample book item.
    private static void CreateBookItem(Table productCatalog)
    {
        Console.WriteLine("\n*** Executing CreateBookItem() ***");
        var book = new Document();
        book["Id"] = sampleBookId;
        book["Title"] = "Book " + sampleBookId;
        book["Price"] = 19.99;
        book["ISBN"] = "111-1111111111";
        book["Authors"] = new List<string> { "Author 1", "Author 2", "Author 3" };
        book["PageCount"] = 500;
        book["Dimensions"] = "8.5x11x.5";
        book["InPublication"] = new DynamoDBBool(true);
        book["InStock"] = new DynamoDBBool(false);
        book["QuantityOnHand"] = 0;

        productCatalog.PutItem(book);
    }

    private static void RetrieveBook(Table productCatalog)
    {
        Console.WriteLine("\n*** Executing RetrieveBook() ***");
        // Optional configuration.
        GetItemOperationConfig config = new GetItemOperationConfig
        {
            AttributesToGet = new List<string> { "Id", "ISBN", "Title", "Authors",
"Price" },
            ConsistentRead = true
        };
        Document document = productCatalog.GetItem(sampleBookId, config);
        Console.WriteLine("RetrieveBook: Printing book retrieved...");
        PrintDocument(document);
    }

    private static void UpdateMultipleAttributes(Table productCatalog)
    {
        Console.WriteLine("\n*** Executing UpdateMultipleAttributes() ***");
        Console.WriteLine("\nUpdating multiple attributes....");
        int partitionKey = sampleBookId;
```

```
var book = new Document();
book["Id"] = partitionKey;
// List of attribute updates.
// The following replaces the existing authors list.
book["Authors"] = new List<string> { "Author x", "Author y" };
book["newAttribute"] = "New Value";
book["ISBN"] = null; // Remove it.

// Optional parameters.
UpdateItemOperationConfig config = new UpdateItemOperationConfig
{
    // Get updated item in response.
    ReturnValues = ReturnValues.AllNewAttributes
};
Document updatedBook = productCatalog.UpdateItem(book, config);
Console.WriteLine("UpdateMultipleAttributes: Printing item after
updates ...");
PrintDocument(updatedBook);
}

private static void UpdateBookPriceConditionally(Table productCatalog)
{
    Console.WriteLine("\n*** Executing UpdateBookPriceConditionally() ***");

    int partitionKey = sampleBookId;

    var book = new Document();
    book["Id"] = partitionKey;
    book["Price"] = 29.99;

    // For conditional price update, creating a condition expression.
    Expression expr = new Expression();
    expr.ExpressionStatement = "Price = :val";
    expr.ExpressionAttributeValueValues[":val"] = 19.00;

    // Optional parameters.
    UpdateItemOperationConfig config = new UpdateItemOperationConfig
    {
        ConditionalExpression = expr,
        ReturnValues = ReturnValues.AllNewAttributes
    };
    Document updatedBook = productCatalog.UpdateItem(book, config);
```

```
        Console.WriteLine("UpdateBookPriceConditionally: Printing item whose price
was conditionally updated");
        PrintDocument(updatedBook);
    }

private static void DeleteBook(Table productCatalog)
{
    Console.WriteLine("\n*** Executing DeleteBook() ***");
    // Optional configuration.
    DeleteItemOperationConfig config = new DeleteItemOperationConfig
    {
        // Return the deleted item.
        ReturnValues = ReturnValues.AllOldAttributes
    };
    Document document = productCatalog.DeleteItem(sampleBookId, config);
    Console.WriteLine("DeleteBook: Printing deleted just deleted...");
    PrintDocument(document);
}

private static void PrintDocument(Document updatedDocument)
{
    foreach (var attribute in updatedDocument.GetAttributeNames())
    {
        string stringValue = null;
        var value = updatedDocument[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("{0} - {1}", attribute, stringValue);
    }
}
}
```

Contoh: Operasi batch menggunakan API model dokumen AWS SDK for .NET

Topik

- [Contoh: Penulisan batch menggunakan model dokumen AWS SDK for .NET](#)

Contoh: Penulisan batch menggunakan model dokumen AWS SDK for .NET

Contoh kode C# berikut menggambarkan operasi penulisan batch tabel tunggal dan multi-tabel. Contoh tersebut melakukan tugas sebagai berikut:

- Menggambarkan penulisan batch tabel tunggal. Dua item akan ditambahkan ke tabel ProductCatalog.
- Menggambarkan penulisan batch multi-tabel. Item akan ditambahkan ke tabel Forum dan Thread. Selain itu, item dari tabel Thread akan dihapus.

Jika mengikuti langkah-langkah dalam [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#), Anda telah membuat tabel ProductCatalog, Forum, dan Thread. Anda juga dapat membuat tabel sampel tersebut secara terprogram. Untuk informasi selengkapnya, lihat [Membuat tabel contoh dan mengunggah data menggunakan AWS SDK for .NET](#). Untuk step-by-step instruksi untuk menguji contoh berikut, lihat [Contoh kode .NET](#).

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class MidLevelBatchWriteItem
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        static void Main(string[] args)
        {
            try
            {
                SingleTableBatchWrite();
                MultiTableBatchWrite();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            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 SingleTableBatchWrite()
    {
        Table productCatalog = Table.LoadTable(client, "ProductCatalog");
        var batchWrite = productCatalog.CreateBatchWrite();

        var book1 = new Document();
        book1["Id"] = 902;
        book1["Title"] = "My book1 in batch write using .NET helper classes";
        book1["ISBN"] = "902-11-11-1111";
        book1["Price"] = 10;
        book1["ProductCategory"] = "Book";
        book1["Authors"] = new List<string> { "Author 1", "Author 2", "Author 3" };
        book1["Dimensions"] = "8.5x11x.5";
        book1["InStock"] = new DynamoDBBool(true);
        book1["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()");
        batchWrite.Execute();
    }

    private static void MultiTableBatchWrite()
    {
        // 1. Specify item to add in the Forum table.
        Table forum = Table.LoadTable(client, "Forum");
        var forumBatchWrite = forum.CreateBatchWrite();

        var forum1 = new Document();
        forum1["Name"] = "Test BatchWrite Forum";
        forum1["Threads"] = 0;
        forumBatchWrite.AddDocumentToPut(forum1);

        // 2a. Specify item to add in the Thread table.
        Table thread = Table.LoadTable(client, "Thread");
        var threadBatchWrite = thread.CreateBatchWrite();

        var thread1 = new Document();
```

```
thread1["ForumName"] = "S3 forum";
thread1["Subject"] = "My sample question";
thread1["Message"] = "Message text";
thread1["KeywordTags"] = new List<string> { "S3", "Bucket" };
threadBatchWrite.AddDocumentToPut(thread1);

// 2b. Specify item to delete from the Thread table.
threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

// 3. Create multi-table batch.
var superBatch = new MultiTableDocumentBatchWrite();
superBatch.AddBatch(forumBatchWrite);
superBatch.AddBatch(threadBatchWrite);
Console.WriteLine("Performing batch write in MultiTableBatchWrite()");
superBatch.Execute();
    }
}
}
```

Menggunakan tabel di DynamoDB dengan model dokumen AWS SDK for .NET

Topik

- [Metode Table.Query di AWS SDK for .NET](#)
- [Metode Table.Scan dalam AWS SDK for .NET](#)

Metode Table.Query di AWS SDK for .NET

Metode Query memungkinkan Anda mengkueri tabel. Anda hanya dapat mengkueri tabel yang memiliki kunci primer komposit (kunci partisi dan kunci urutan). Jika kunci primer tabel Anda hanya terbuat dari kunci partisi, operasi Query tidak didukung. Secara default, Query melakukan kueri secara internal yang akhirnya konsisten. Untuk mempelajari model konsistensi, lihat [Konsistensi baca](#).

Metode Query menyediakan dua beban berlebih. Parameter minimum yang diperlukan untuk metode Query adalah nilai kunci partisi dan filter kunci urutan. Anda dapat menggunakan beban berlebih berikut untuk memberikan parameter minimum yang diperlukan.

Example

```
Query(Primitive partitionKey, RangeFilter Filter);
```

Misalnya, kueri kode C# berikut untuk semua balasan dalam forum yang diposting dalam 15 hari terakhir.

Example

```
string tableName = "Reply";
Table table = Table.LoadTable(client, tableName);

DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
RangeFilter filter = new RangeFilter(QueryOperator.GreaterThan, twoWeeksAgoDate);
Search search = table.Query("DynamoDB Thread 2", filter);
```

Kode ini akan membuat objek Search. Sekarang, Anda dapat memanggil metode Search.GetNextSet secara iteratif untuk mengambil satu halaman hasil pada satu waktu, seperti yang ditunjukkan dalam contoh kode C# berikut. Kode akan mencetak nilai atribut untuk setiap item yang ditampilkan kueri.

Example

```
List<Document> documentSet = new List<Document>();
do
{
    documentSet = search.GetNextSet();
    foreach (var document in documentSet)
        PrintDocument(document);
} while (!search.IsDone);

private 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;
        else if (value is PrimitiveList)
            stringValue = string.Join(",", (from primitive
                                           in value.AsPrimitiveList().Entries
                                           select primitive.Value).ToArray());
        Console.WriteLine("{0} - {1}", attribute, stringValue);
    }
}
```

```
}
```

Menentukan parameter opsional

Anda juga dapat menentukan parameter opsional untuk Query, seperti menentukan daftar atribut yang akan diambil, bacaan sangat konsisten, ukuran halaman, dan jumlah item yang ditampilkan per halaman. Untuk daftar lengkap parameter, lihat [Kueri](#). Untuk menentukan parameter opsional, Anda harus menggunakan beban berlebih berikut tempat Anda memberikan objek QueryOperationConfig.

Example

```
Query(QueryOperationConfig config);
```

Misalnya, Anda ingin menjalankan kueri dalam contoh sebelumnya (mengambil balasan forum yang diposting dalam 15 hari terakhir). Namun, anggaplah Anda ingin menyediakan parameter kueri opsional hanya untuk mengambil atribut tertentu serta meminta bacaan sangat konsisten. Contoh kode C# berikut menampilkan permintaan menggunakan objek QueryOperationConfig.

Example

```
Table table = Table.LoadTable(client, "Reply");
DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
QueryOperationConfig config = new QueryOperationConfig()
{
    HashKey = "DynamoDB Thread 2", //Partition key
    AttributesToGet = new List<string>
    { "Subject", "ReplyDateTime", "PostedBy" },
    ConsistentRead = true,
    Filter = new RangeFilter(QueryOperator.GreaterThan, twoWeeksAgoDate)
};

Search search = table.Query(config);
```

Contoh: Kueri menggunakan metode Table.Query

Contoh kode C# berikut menggunakan metode Table.Query untuk menjalankan kueri sampel berikut.

- Kueri berikut dijalankan terhadap tabel Reply.
 - Temukan balasan utas forum yang diposting dalam 15 hari terakhir.

Kueri ini dijalankan dua kali. Dalam panggilan `Table.Query` pertama, contoh berikut hanya menyediakan parameter kueri yang diperlukan. Dalam panggilan `Table.Query` kedua, Anda menyediakan parameter kueri opsional untuk meminta bacaan sangat konsisten dan daftar atribut yang akan diambil.

- Temukan balasan utas forum yang diposting selama periode waktu tertentu.

Kueri ini menggunakan operator kueri `Between` untuk menemukan balasan yang diposting di antara dua tanggal.

- Dapatkan produk dari tabel `ProductCatalog`.

Karena tabel `ProductCatalog` memiliki kunci primer yang hanya berupa kunci partisi, Anda hanya bisa mendapatkan item, tetapi tidak dapat mengkueri tabel. Contoh akan mengambil item produk tertentu menggunakan item `Id`.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class MidLevelQueryAndScan
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                // Query examples.
                Table replyTable = Table.LoadTable(client, "Reply");
                string forumName = "Amazon DynamoDB";
                string threadSubject = "DynamoDB Thread 2";
                FindRepliesInLast15Days(replyTable, forumName, threadSubject);
            }
        }
    }
}
```

```
        FindRepliesInLast15DaysWithConfig(replyTable, forumName,
threadSubject);
        FindRepliesPostedWithinTimePeriod(replyTable, forumName,
threadSubject);

        // Get Example.
        Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
        int productId = 101;
        GetProduct(productCatalogTable, productId);

        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 GetProduct(Table tableName, int productId)
{
    Console.WriteLine("*** Executing GetProduct() ***");
    Document productDocument = tableName.GetItem(productId);
    if (productDocument != null)
    {
        PrintDocument(productDocument);
    }
    else
    {
        Console.WriteLine("Error: product " + productId + " does not exist");
    }
}

private static void FindRepliesInLast15Days(Table table, string forumName,
string threadSubject)
{
    string Attribute = forumName + "#" + threadSubject;

    DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
    QueryFilter filter = new QueryFilter("Id", QueryOperator.Equal,
partitionKey);
    filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

    // Use Query overloads that takes the minimum required query parameters.
```

```
Search search = table.Query(filter);

List<Document> documentSet = new List<Document>();
do
{
    documentSet = search.GetNextSet();
    Console.WriteLine("\nFindRepliesInLast15Days: printing .....");
    foreach (var document in documentSet)
        PrintDocument(document);
} while (!search.IsDone);
}

private static void 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));

    QueryFilter filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"# " + threadSubject);
    filter.AddCondition("ReplyDateTime", QueryOperator.Between, startDate,
endDate);

    QueryOperationConfig 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);

List<Document> documentList = new List<Document>();

do
{
    documentList = search.GetNextSet();
    Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing
replies posted within dates: {0} and {1} .....", startDate, endDate);
    foreach (var document in documentList)
```

```
        {
            PrintDocument(document);
        }
    } while (!search.IsDone);
}

private static void FindRepliesInLast15DaysWithConfig(Table table, string
forumName, string threadName)
{
    DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
    QueryFilter filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"# " + threadName);
    filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);
    // You are specifying optional parameters so use QueryOperationConfig.
    QueryOperationConfig config = new QueryOperationConfig()
    {
        Filter = filter,
        // Optional parameters.
        Select = SelectValues.SpecificAttributes,
        AttributesToGet = new List<string> { "Message", "ReplyDateTime",
"PostedBy" },
        ConsistentRead = true
    };

    Search search = table.Query(config);

    List<Document> documentSet = new List<Document>();
    do
    {
        documentSet = search.GetNextSet();
        Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
printing .....");
        foreach (var document in documentSet)
            PrintDocument(document);
    } while (!search.IsDone);
}

private static void PrintDocument(Document document)
{
    // count++;
    Console.WriteLine();
    foreach (var attribute in document.GetAttributeNames())
    {
```

```
        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("{0} - {1}", attribute, stringValue);
    }
}
}
```

Metode Table.Scan dalam AWS SDK for .NET

Metode Scan melakukan pemindaian tabel penuh. Metode ini menyediakan dua beban berlebih. Satu-satunya parameter yang diperlukan oleh metode Scan adalah filter pemindaian, yang dapat Anda berikan menggunakan beban berlebih berikut.

Example

```
Scan(ScanFilter filter);
```

Misalnya, Anda mengelola tabel utas forum yang melacak informasi seperti subjek utas (primer), pesan terkait, Id forum tempat utas berada, Tags, dan informasi lainnya. Anggaplah subjek adalah kunci primer.

Example

```
Thread(Subject, Message, ForumId, Tags, LastPostedDateTime, .... )
```

Subjek ini adalah versi sederhana dari forum dan utas yang Anda lihat di forum AWS (lihat [Forum diskusi](#)). Contoh kode C # berikut menanyakan semua utas di forum tertentu (ForumId = 101) yang diberi tag "sortkey". Karena ForumId bukan kunci primer, contoh akan memindai tabel. ScanFilter mencakup dua syarat. Kueri akan menampilkan semua utas yang memenuhi kedua syarat.

Example

```
string tableName = "Thread";
Table ThreadTable = Table.LoadTable(client, tableName);
```

```
ScanFilter scanFilter = new ScanFilter();
scanFilter.AddCondition("ForumId", ScanOperator.Equal, 101);
scanFilter.AddCondition("Tags", ScanOperator.Contains, "sortkey");

Search search = ThreadTable.Scan(scanFilter);
```

Menentukan parameter opsional

Anda juga dapat menentukan parameter opsional untuk Scan, seperti daftar atribut tertentu yang akan diambil atau apakah akan melakukan bacaan sangat konsisten. Untuk menentukan parameter opsional, Anda harus membuat objek `ScanOperationConfig` berisi parameter yang diperlukan dan opsional, serta menggunakan beban berlebih berikut.

Example

```
Scan(ScanOperationConfig config);
```

Kode contoh C# berikut menjalankan kueri sebelumnya yang sama (temukan utas forum dengan `ForumId` adalah 101 dan atribut `Tag` berisi kata kunci "sortkey"). Anggaplah Anda ingin menambahkan parameter opsional hanya untuk mengambil daftar atribut tertentu. Dalam hal ini, Anda harus membuat objek `ScanOperationConfig` dengan menyediakan semua parameter, yaitu parameter yang diperlukan dan parameter opsional, seperti yang ditunjukkan dalam contoh kode berikut.

Example

```
string tableName = "Thread";
Table ThreadTable = Table.LoadTable(client, tableName);

ScanFilter scanFilter = new ScanFilter();
scanFilter.AddCondition("ForumId", ScanOperator.Equal, forumId);
scanFilter.AddCondition("Tags", ScanOperator.Contains, "sortkey");

ScanOperationConfig config = new ScanOperationConfig()
{
    AttributesToGet = new List<string> { "Subject", "Message" } ,
    Filter = scanFilter
};

Search search = ThreadTable.Scan(config);
```

Contoh: Pindai menggunakan metode Table.Scan

Operasi Scan melakukan pemindaian tabel penuh sehingga operasi berpotensi menimbulkan biaya yang mahal. Sebagai gantinya, Anda harus menggunakan kueri. Namun, terkadang Anda mungkin perlu menjalankan pemindaian pada tabel. Misalnya, Anda mungkin mengalami kesalahan entri data dalam harga produk dan harus memindai tabel seperti yang ditunjukkan dalam contoh kode C# berikut. Contoh memindai tabel ProductCatalog untuk menemukan produk dengan nilai harga kurang dari 0. Contoh menggambarkan penggunaan dua beban berlebih Table.Scan.

- Table.Scan yang mengambil objek ScanFilter sebagai parameter.

Anda dapat meneruskan parameter ScanFilter saat meneruskan parameter yang diperlukan saja.

- Table.Scan yang mengambil objek ScanOperationConfig sebagai parameter.

Anda harus menggunakan parameter ScanOperationConfig jika ingin meneruskan setiap parameter opsional untuk metode Scan.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;

namespace com.amazonaws.codesamples
{
    class MidLevelScanOnly
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
            // Scan example.
            FindProductsWithNegativePrice(productCatalogTable);
            FindProductsWithNegativePriceWithConfig(productCatalogTable);

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }
    }
}
```

```
    }

    private static void FindProductsWithNegativePrice(Table productCatalogTable)
    {
        // Assume there is a price error. So we scan to find items priced < 0.
        ScanFilter scanFilter = new ScanFilter();
        scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

        Search search = productCatalogTable.Scan(scanFilter);

        List<Document> documentList = new List<Document>();
        do
        {
            documentList = search.GetNextSet();
            Console.WriteLine("\nFindProductsWithNegativePrice:
printing .....");
            foreach (var document in documentList)
                PrintDocument(document);
        } while (!search.IsDone);
    }

    private static void FindProductsWithNegativePriceWithConfig(Table
productCatalogTable)
    {
        // Assume there is a price error. So we scan to find items priced < 0.
        ScanFilter scanFilter = new ScanFilter();
        scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

        ScanOperationConfig config = new ScanOperationConfig()
        {
            Filter = scanFilter,
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string> { "Title", "Id" }
        };

        Search search = productCatalogTable.Scan(config);

        List<Document> documentList = new List<Document>();
        do
        {
            documentList = search.GetNextSet();
            Console.WriteLine("\nFindProductsWithNegativePriceWithConfig:
printing .....");
            foreach (var document in documentList)
```



```
        PrintDocument(document);
    } while (!search.IsDone);
}

private static void PrintDocument(Document document)
{
    // count++;
    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("{0} - {1}", attribute, stringValue);
    }
}
}
```

.NET: Model persistensi objek

Topik

- [Atribut DynamoDB](#)
- [Kelas DynamoDBContext](#)
- [Jenis data yang didukung](#)
- [Penguncian optimis menggunakan nomor versi dengan DynamoDB menggunakan model persistensi objek AWS SDK for .NET](#)
- [Memetakan data arbitrer dengan DynamoDB Menggunakan model persistensi objek AWS SDK for .NET](#)
- [Operasi batch menggunakan model persistensi objek AWS SDK for .NET](#)
- [Contoh: Operasi CRUD menggunakan model persistensi objek AWS SDK for .NET](#)
- [Contoh: Operasi tulis batch menggunakan model persistensi objek AWS SDK for .NET](#)
- [Contoh: Kueri dan pemindaian di DynamoDB menggunakan model persistensi objek AWS SDK for .NET](#)

AWS SDK for .NET menyediakan model persistensi objek yang memungkinkan Anda memetakan kelas sisi klien ke tabel Amazon DynamoDB. Kemudian, setiap instans objek dipetakan ke item dalam tabel yang sesuai. Untuk menyimpan objek sisi klien ke tabel, model persistensi objek menyediakan kelas `DynamoDBContext`, yaitu titik masuk ke DynamoDB. Kelas ini menyediakan koneksi ke DynamoDB dan memungkinkan Anda mengakses tabel, melakukan berbagai operasi CRUD, dan menjalankan kueri.

Model persistensi objek menyediakan sekumpulan atribut untuk memetakan kelas sisi klien ke tabel, dan properti/bidang ke atribut tabel.

Note

Model persistensi objek tidak menyediakan API untuk membuat, memperbarui, atau menghapus tabel. Model ini hanya menyediakan operasi data. Anda hanya dapat menggunakan API tingkat rendah AWS SDK for .NET untuk membuat, memperbarui, dan menghapus tabel. Untuk informasi selengkapnya, lihat [Bekerja dengan tabel DynamoDB di .NET](#).

Contoh berikut menunjukkan cara kerja model persistensi objek. Contoh ini dimulai dengan tabel `ProductCatalog`. Contoh ini memiliki `Id` sebagai kunci primernya.

```
ProductCatalog(Id, ...)
```

Katakanlah Anda memiliki kelas `Book` yang berisi properti `Title`, `ISBN`, dan `Authors`. Anda dapat memetakan kelas `Book` ke tabel `ProductCatalog` dengan menambahkan atribut yang didefinisikan oleh model persistensi objek, seperti yang ditunjukkan dalam kode contoh C# berikut.

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; }
}
```

Dalam contoh sebelumnya, atribut `DynamoDBTable` memetakan kelas `Book` ke tabel `ProductCatalog`.

Model persistensi objek mendukung pemetaan eksplisit dan default antara properti kelas dan atribut tabel.

- Pemetaan eksplisit—Untuk memetakan properti ke kunci primer, Anda harus menggunakan atribut model persistensi objek `DynamoDBHashKey` dan `DynamoDBRangeKey`. Selain itu, untuk atribut kunci nonprimer, jika nama properti di kelas Anda dan atribut tabel terkait yang ingin Anda petakan tidak sama, Anda harus mendefinisikan pemetaan dengan menambahkan atribut `DynamoDBProperty` secara eksplisit.

Dalam contoh sebelumnya, properti `Id` dipetakan ke kunci primer dengan nama yang sama, dan properti `BookAuthors` dipetakan ke atribut `Authors` dalam tabel `ProductCatalog`.

- Pemetaan default—Secara default, model persistensi objek memetakan properti kelas ke atribut dengan nama yang sama dalam tabel.

Dalam contoh sebelumnya, properti `Title` dan `ISBN` dipetakan ke atribut dengan nama yang sama dalam tabel `ProductCatalog`.

Anda tidak perlu memetakan setiap properti kelas tunggal. Anda mengidentifikasi properti ini dengan menambahkan atribut `DynamoDBIgnore`. Ketika Anda menyimpan instans `Book` ke tabel, `DynamoDBContext` tidak menyertakan properti `CoverPage`. Instans tersebut juga tidak mengembalikan properti ini saat Anda mengambil instans buku.

Anda dapat memetakan properti jenis `.NET` primitif seperti `int` dan `string`. Anda juga dapat memetakan setiap jenis data arbitrer selama Anda memberikan konverter yang sesuai untuk memetakan data arbitrer ke salah satu jenis `DynamoDB`. Untuk mempelajari cara memetakan jenis arbitrer, lihat [Memetakan data arbitrer dengan DynamoDB Menggunakan model persistensi objek AWS SDK for .NET](#).

Model persistensi objek mendukung penguncian optimis. Selama operasi pembaruan, model ini memastikan bahwa Anda memiliki salinan terbaru dari item yang akan Anda perbarui. Untuk

informasi selengkapnya, lihat [Penguncian optimis menggunakan nomor versi dengan DynamoDB menggunakan model persistensi objek AWS SDK for .NET](#).

Atribut DynamoDB

Bagian ini menjelaskan atribut yang ditawarkan model persistensi objek agar Anda dapat memetakan kelas dan properti ke tabel dan atribut DynamoDB.

Note

Dalam atribut berikut, hanya `DynamoDBTable` dan `DynamoDBHashKey` yang diperlukan.

DynamoDB GlobalSecondaryIndexHashKey

Memetakan properti kelas ke kunci partisi untuk indeks sekunder global. Gunakan atribut ini jika Anda perlu Query indeks sekunder global.

DynamoDB GlobalSecondaryIndexRangeKey

Memetakan properti kelas ke kunci urutan indeks sekunder global. Gunakan atribut ini jika Anda perlu Query indeks sekunder global dan ingin menyempurnakan hasil menggunakan kunci urutan indeks.

DynamoDB HashKey

Memetakan properti kelas ke kunci partisi untuk kunci primer tabel. Atribut kunci primer tidak boleh berupa jenis koleksi.

Contoh kode C# berikut memetakan kelas `Book` ke tabel `ProductCatalog`, dan properti `Id` ke kunci partisi kunci primer tabel.

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }

    // Additional properties go here.
}
```

DynamoDBIgnore

Menunjukkan bahwa properti terkait harus diabaikan. Jika tidak ingin menyimpan properti kelas, Anda dapat menambahkan atribut ini untuk meminta agar `DynamoDBContext` tidak menyertakan properti ini saat menyimpan objek ke tabel.

DynamoDB LocalSecondaryIndexRangeKey

Memetakan properti kelas ke kunci urutan indeks sekunder lokal. Gunakan atribut ini jika Anda perlu Query indeks sekunder lokal dan ingin menyempurnakan hasil menggunakan kunci urutan indeks.

DynamoDBProperty

Memetakan properti kelas ke atribut tabel. Jika properti kelas dipetakan ke atribut tabel dengan nama yang sama, Anda tidak perlu menentukan atribut ini. Namun, jika nama tidak sama, Anda dapat menggunakan tanda ini untuk memberikan pemetaan. Dalam pernyataan C# berikut, `DynamoDBProperty` memetakan properti `BookAuthors` ke atribut `Authors` dalam tabel.

```
[DynamoDBProperty("Authors")]  
public List<string> BookAuthors { get; set; }
```

`DynamoDBContext` menggunakan informasi pemetaan ini untuk membuat atribut `Authors` saat menyimpan data objek ke tabel yang sesuai.

DynamoDBRenamable

Menentukan nama alternatif untuk properti kelas. Langkah ini berguna jika Anda menulis konverter khusus untuk memetakan data arbitrer ke tabel DynamoDB yang nama properti kelasnya berbeda dengan atribut tabel.

DynamoDB RangeKey

Memetakan properti kelas ke kunci urutan untuk kunci primer tabel. Jika tabel memiliki kunci primer komposit (kunci partisi dan kunci urutan), Anda harus menentukan atribut `DynamoDBHashKey` dan `DynamoDBRangeKey` dalam pemetaan kelas Anda.

Misalnya, tabel sampel `Reply` memiliki kunci primer yang dibuat dari kunci partisi `Id` dan kunci urutan `ReplyEnishment`. Contoh kode C# berikut memetakan kelas `Reply` ke tabel `Reply`. Definisi kelas juga menunjukkan bahwa dua propertinya dipetakan ke kunci primer.

Untuk informasi selengkapnya tentang tabel sampel, lihat [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

```
[DynamoDBTable("Reply")]
public class Reply
{
    [DynamoDBHashKey]
    public int ThreadId { get; set; }
    [DynamoDBRangeKey]
    public string Replenishment { get; set; }

    // Additional properties go here.
}
```

DynamoDBTable

Mengidentifikasi tabel target di DynamoDB tempat kelas dipetakan. Misalnya, contoh kode C# berikut memetakan kelas Developer ke tabel People di DynamoDB.

```
[DynamoDBTable("People")]
public class Developer { ...}
```

Atribut ini dapat diwariskan atau diganti.

- Atribut `DynamoDBTable` dapat diwariskan. Dalam contoh sebelumnya, jika Anda menambahkan kelas baru, yaitu `Lead`, yang diwariskan dari kelas `Developer`, kelas tersebut juga dipetakan ke tabel `People`. Objek `Developer` dan `Lead` disimpan dalam tabel `People`.
- Atribut `DynamoDBTable` juga dapat diganti. Dalam contoh kode C# berikut, kelas `Manager` diwariskan dari kelas `Developer`. Namun, penambahan eksplisit atribut `DynamoDBTable` memetakan kelas ke tabel lainnya (`Managers`).

```
[DynamoDBTable("Managers")]
public class Manager : Developer { ...}
```

Anda dapat menambahkan parameter opsional, yaitu `LowerCamelCaseProperties`, untuk meminta DynamoDB menulis huruf pertama dari nama properti dalam huruf kecil ketika menyimpan objek ke tabel, seperti yang ditunjukkan dalam contoh C# berikut.

```
[DynamoDBTable("People", LowerCamelCaseProperties=true)]
public class Developer
{
    string DeveloperName;
```

```
    ...  
}
```

Ketika menyimpan instans kelas `Developer`, `DynamoDBContext` menyimpan properti `DeveloperName` sebagai `developerName`.

DynamoDBVersion

Mengidentifikasi properti kelas untuk menyimpan nomor versi item. Untuk informasi selengkapnya tentang versioning, lihat [Penguncian optimis menggunakan nomor versi dengan DynamoDB menggunakan model persistensi objek AWS SDK for .NET](#).

Kelas DynamoDBContext

Kelas `DynamoDBContext` adalah titik masuk ke basis data Amazon DynamoDB. Kelas ini menyediakan koneksi ke DynamoDB dan memungkinkan Anda mengakses data dalam berbagai tabel, melakukan berbagai operasi CRUD, dan menjalankan kueri. Kelas `DynamoDBContext` menyediakan metode berikut.

Buat MultiTable BatchGet

Membuat objek `MultiTableBatchGet`, yang terdiri dari beberapa objek `BatchGet` individual. Setiap objek `BatchGet` dapat digunakan untuk mengambil item dari tabel DynamoDB tunggal.

Untuk mengambil item dari tabel, gunakan metode `ExecuteBatchGet`, dengan meneruskan objek `MultiTableBatchGet` sebagai parameter.

Buat MultiTable BatchWrite

Membuat objek `MultiTableBatchWrite`, yang terdiri dari beberapa objek `BatchWrite` individual. Setiap objek `BatchWrite` dapat digunakan untuk menulis atau menghapus item dalam tabel DynamoDB tunggal.

Untuk menulis ke tabel, gunakan metode `ExecuteBatchWrite`, dengan meneruskan objek `MultiTableBatchWrite` sebagai parameter.

CreateBatchGet

Membuat objek `BatchGet` yang dapat Anda gunakan untuk mengambil beberapa item dari tabel. Untuk informasi selengkapnya, lihat [Batch get: Mendapatkan beberapa item](#).

createBatchWrite

Membuat objek `BatchWrite` yang dapat Anda gunakan untuk menempatkan beberapa item ke dalam tabel, atau untuk menghapus beberapa item dari tabel. Untuk informasi selengkapnya, lihat [Penulisan batch: Menempatkan dan menghapus beberapa item](#).

Delete

Menghapus item dari tabel. Metode ini memerlukan kunci primer dari item yang ingin Anda hapus. Anda dapat memberikan nilai kunci primer maupun objek sisi klien yang berisi nilai kunci primer sebagai parameter untuk metode ini.

- Jika Anda menentukan objek sisi klien sebagai parameter dan Anda telah mengaktifkan penguncian optimis, penghapusan hanya akan berhasil jika versi sisi klien dan sisi server objek cocok.
- Jika Anda hanya menetapkan nilai kunci primer sebagai parameter, penghapusan akan berhasil terlepas dari apakah Anda telah mengaktifkan penguncian optimis atau tidak.

Note

Untuk melakukan operasi ini di latar belakang, gunakan metode `DeleteAsync`.

Dispose

Membuang semua sumber daya yang dikelola dan tidak dikelola.

Executebatchget

Membaca data dari satu atau beberapa tabel, yang memproses semua objek `BatchGet` dalam `MultiTableBatchGet`.

Note

Untuk melakukan operasi ini di latar belakang, gunakan metode `ExecuteBatchGetAsync`.

ExecuteBatchWrite

Menulis atau menghapus data dalam satu atau beberapa tabel, yang memproses semua objek `BatchWrite` dalam `MultiTableBatchWrite`.

Note

Untuk melakukan operasi ini di latar belakang, gunakan metode `ExecuteBatchWriteAsync`.

FromDocument

Mengingat instans `Document`, metode `FromDocument` akan mengembalikan instans dari kelas sisi klien.

Langkah ini membantu jika Anda ingin menggunakan kelas model dokumen bersama dengan model persistensi objek untuk melakukan operasi data. Untuk informasi selengkapnya tentang kelas model dokumen yang disediakan oleh AWS SDK for .NET, lihat [.NET: Model dokumen](#).

Misalkan Anda memiliki objek `Document` bernama `doc`, yang berisi representasi item `Forum`. (Untuk mengetahui cara membuat objek ini, lihat deskripsi untuk metode `ToDocument` nanti dalam topik ini.) Anda dapat menggunakan `FromDocument` untuk mengambil item `Forum` dari `Document`, seperti yang ditunjukkan dalam contoh kode C# berikut.

Example

```
forum101 = context.FromDocument<Forum>(101);
```

Note

Jika objek `Document` mengimplementasikan antarmuka `IEnumerable`, Anda dapat menggunakan metode `FromDocuments` sebagai gantinya. Metode ini memungkinkan Anda melakukan iterasi pada semua instans kelas dalam `Document`.

FromQuery

Menjalankan operasi `Query`, dengan parameter kueri yang didefinisikan dalam objek `QueryOperationConfig`.

Note

Untuk melakukan operasi ini di latar belakang, gunakan metode `FromQueryAsync`.

FromScan

Menjalankan operasi `Scan`, dengan parameter pemindaian yang didefinisikan dalam objek `ScanOperationConfig`.

Note

Untuk melakukan operasi ini di latar belakang, gunakan metode `FromScanAsync`.

Gettable

Mengambil tabel target untuk jenis yang ditentukan. Langkah ini berguna jika Anda menulis konverter khusus untuk memetakan data arbitrer ke tabel DynamoDB, dan Anda perlu menentukan tabel yang terkait dengan jenis data khusus.

Load

Mengambil item dari tabel. Metode ini hanya memerlukan kunci primer dari item yang ingin Anda ambil.

Secara default, DynamoDB akan mengembalikan item dengan nilai yang akhirnya konsisten. Untuk informasi tentang model konsistensi akhir, lihat [Konsistensi baca](#).

Note

Untuk melakukan operasi ini di latar belakang, gunakan metode `LoadAsync`.

Query

Mengkueri tabel berdasarkan parameter kueri yang Anda berikan.

Anda dapat mengkueri tabel hanya jika memiliki kunci primer komposit (kunci partisi dan kunci urutan). Saat mengkueri, Anda harus menentukan kunci partisi dan syarat yang berlaku untuk kunci urutan.

Katakanlah Anda memiliki kelas Reply sisi klien yang dipetakan ke tabel Reply di DynamoDB. Contoh kode C# berikut mengkueri tabel Reply untuk menemukan balasan utas dalam forum yang diposting dalam 15 hari terakhir. Tabel Reply memiliki kunci primer yang berisi kunci partisi Id dan kunci urutan ReplyDateTime. Untuk informasi selengkapnya tentang tabel Reply, lihat [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

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);
```

Tabel ini mengembalikan koleksi objek Reply.

Metode Query mengembalikan koleksi IEnumerable "lazy-loaded". Metode ini awalnya hanya mengembalikan satu halaman hasil, lalu membuat panggilan layanan untuk halaman berikutnya jika diperlukan. Untuk mendapatkan semua item yang cocok, Anda perlu melakukan iterasi hanya pada IEnumerable.

Jika tabel Anda memiliki kunci primer sederhana (kunci partisi), Anda tidak dapat menggunakan metode Query. Sebaliknya, Anda dapat menggunakan metode Load dan menyediakan kunci partisi untuk mengambil item.

Note

Untuk melakukan operasi ini di latar belakang, gunakan metode QueryAsync.

Save

Menyimpan objek tertentu ke tabel. Jika kunci primer yang ditentukan dalam objek input tidak ada dalam tabel, metode akan menambahkan item baru ke tabel. Jika kunci primer ada, metode akan memperbarui item yang ada.

Jika Anda mengonfigurasi penguncian optimis, pembaruan hanya akan berhasil jika versi klien dan sisi server item cocok. Untuk informasi selengkapnya, lihat [Penguncian optimis menggunakan nomor versi dengan DynamoDB menggunakan model persistensi objek AWS SDK for .NET](#).

Note

Untuk melakukan operasi ini di latar belakang, gunakan metode `SaveAsync`.

Pindai

Melakukan pemindaian pada seluruh tabel.

Anda dapat memfilter hasil pemindaian dengan menentukan syarat pemindaian. Syarat ini dapat dievaluasi pada setiap atribut dalam tabel. Katakanlah Anda memiliki kelas sisi klien `Book` yang dipetakan ke tabel `ProductCatalog` di DynamoDB. Contoh C# berikut memindai tabel dan hanya mengembalikan item buku dengan harga kurang dari 0.

Example

```
IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
    new ScanCondition("Price", ScanOperator.LessThan, price),
    new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
);
```

Metode `Scan` mengembalikan koleksi `IEnumerable` “lazy-loaded”. Metode ini awalnya hanya mengembalikan satu halaman hasil, lalu membuat panggilan layanan untuk halaman berikutnya jika diperlukan. Untuk mendapatkan semua item yang cocok, Anda hanya perlu melakukan iterasi pada `IEnumerable`.

Untuk alasan performa, Anda harus mencari tabel dan menghindari pemindaian tabel.

Note

Untuk melakukan operasi ini di latar belakang, gunakan metode `ScanAsync`.

ToDocument

Mengembalikan instans untuk kelas model dokumen `Document` dari instans kelas Anda.

Langkah ini membantu jika Anda ingin menggunakan kelas model dokumen bersama dengan model persistensi objek untuk melakukan operasi data. Untuk informasi selengkapnya tentang kelas model dokumen yang disediakan oleh AWS SDK for .NET, lihat [.NET: Model dokumen](#).

Misalkan Anda memiliki kelas sisi klien yang dipetakan ke tabel Forum sampel. Anda kemudian dapat menggunakan `DynamoDBContext` untuk mendapatkan item sebagai objek `Document` dari tabel Forum, seperti yang ditunjukkan dalam contoh kode C# berikut.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

Forum forum101 = context.Load<Forum>(101); // Retrieve a forum by primary key.
Document doc = context.ToDocument<Forum>(forum101);
```

Menentukan parameter opsional untuk DynamoDBContext

Saat menggunakan model persistensi objek, Anda dapat menentukan parameter opsional berikut untuk `DynamoDBContext`.

- **ConsistentRead**—Saat mengambil data menggunakan operasi `Load`, `Query`, atau `Scan`, Anda dapat menambahkan parameter opsional ini guna meminta nilai data terbaru.
- **IgnoreNullValues**—Parameter ini memberi tahu `DynamoDBContext` untuk mengabaikan nilai nol pada atribut selama operasi `Save`. Jika parameter ini adalah `false` (atau jika tidak diatur), nilai nol ditafsirkan sebagai direktif untuk menghapus atribut tertentu.
- **SkipVersionCheck**— Parameter ini memberi tahu `DynamoDBContext` untuk tidak membandingkan versi saat menyimpan atau menghapus item. Untuk informasi selengkapnya tentang versioning, lihat [Penguncian optimis menggunakan nomor versi dengan DynamoDB menggunakan model persistensi objek AWS SDK for .NET](#).
- **TableNamePrefix**— Memberikan prefiks pada semua nama tabel dengan string tertentu. Jika parameter ini adalah nol (atau jika tidak diatur), tidak ada prefiks yang digunakan.

Contoh C# berikut akan membuat `DynamoDBContext` dengan menentukan dua parameter opsional sebelumnya.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context =
    new DynamoDBContext(client, new DynamoDBContextConfig { ConsistentRead = true,
        SkipVersionCheck = true});
```

DynamoDBContext menyertakan parameter opsional ini dengan setiap permintaan yang Anda kirimkan menggunakan konteks ini.

Alih-alih mengatur parameter ini di tingkat DynamoDBContext, Anda dapat menentukannya untuk setiap operasi yang Anda jalankan menggunakan DynamoDBContext, seperti yang ditunjukkan dalam contoh kode C# berikut. Contoh ini memuat item buku tertentu. Metode Load untuk DynamoDBContext menentukan parameter opsional sebelumnya.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();  
...  
DynamoDBContext context = new DynamoDBContext(client);  
Book bookItem = context.Load<Book>(productId, new DynamoDBContextConfig{ ConsistentRead  
    = true, SkipVersionCheck = true });
```

Dalam kasus ini, DynamoDBContext menyertakan parameter tersebut hanya ketika mengirimkan permintaan Get.

Jenis data yang didukung

Model persistensi objek mendukung sekumpulan jenis data .NET primitif, koleksi, dan jenis data arbitrer. Model ini mendukung jenis data primitif berikut.

- bool
- byte
- char
- DateTime
- decimal
- double
- float
- Int16
- Int32
- Int64
- SByte
- string

- UInt16
- UInt32
- UInt64

Model persistensi objek juga mendukung jenis koleksi .NET. `DynamoDBContext` dapat mengubah jenis koleksi konkret dan Plain Old CLR Object (PoCo) sederhana.

Tabel berikut merangkum pemetaan jenis .NET sebelumnya untuk jenis DynamoDB.

Jenis .NET primitif	Jenis DynamoDB
Semua jenis angka	N (jenis angka)
Semua jenis string	S (jenis string)
<code>MemoryStream</code> , <code>byte []</code>	B (jenis biner)
<code>bool</code>	N (jenis angka). 0 mewakili false dan 1 mewakili true.
Jenis koleksi	Jenis BS (kumpulan biner), jenis SS (kumpulan string), dan jenis NS (kumpulan angka)
<code>DateTime</code>	S (jenis string). Nilai <code>DateTime</code> disimpan sebagai string berformat ISO-8601.

Model persistensi objek juga mendukung jenis data arbitrer. Namun, Anda harus menyediakan kode konverter untuk memetakan jenis kompleks ke jenis DynamoDB.

Note

- Nilai biner kosong didukung.
- Pembacaan nilai string kosong didukung. Nilai atribut string kosong didukung dalam nilai atribut string dari jenis Set saat menulis ke DynamoDB. Nilai atribut string kosong dari jenis string dan nilai string kosong yang berada dalam jenis Daftar atau Peta dihilangkan dari permintaan tulis

Penguncian optimis menggunakan nomor versi dengan DynamoDB menggunakan model persistensi objek AWS SDK for .NET

Dukungan penguncian optimis dalam model persistensi objek memastikan bahwa versi item untuk aplikasi Anda sama dengan versi item di sisi server sebelum memperbarui atau menghapus item. Misalkan Anda mengambil item untuk pembaruan. Namun, sebelum Anda mengirimkan pembaruan Anda kembali, beberapa aplikasi lain memperbarui item yang sama. Sekarang aplikasi Anda memiliki salinan item yang sudah usang. Tanpa penguncian optimis, setiap pembaruan yang Anda lakukan akan menimpa pembaruan yang dibuat oleh aplikasi lain.

Fitur penguncian optimis model persistensi objek menyediakan tanda `DynamoDBVersion` yang dapat Anda gunakan untuk mengaktifkan penguncian optimis. Untuk menggunakan fitur ini, Anda menambahkan properti ke kelas Anda untuk menyimpan nomor versi. Anda menambahkan atribut `DynamoDBVersion` untuk properti. Ketika Anda pertama kali menyimpan objek, `DynamoDBContext` menetapkan nomor versi dan menambah nilai ini setiap kali Anda memperbarui item.

Permintaan perbarui atau hapus Anda akan berhasil hanya jika versi objek sisi kliennya cocok dengan nomor versi item yang sesuai di sisi server. Jika aplikasi Anda memiliki salinan yang sudah usang, aplikasi tersebut harus mendapatkan versi terbaru dari server sebelum dapat memperbarui atau menghapus item.

Contoh kode C# berikut mendefinisikan kelas `Book` dengan atribut persistensi objek yang memetakannya ke tabel `ProductCatalog`. Properti `VersionNumber` di kelas yang dilengkapi dengan atribut `DynamoDBVersion` menyimpan nilai nomor versi.

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

Anda dapat menerapkan atribut `DynamoDBVersion` hanya untuk jenis primitif numerik yang dapat dinihilkan (seperti `int?`).

Penguncian optimis memiliki dampak berikut terhadap operasi `DynamoDBContext`:

- Untuk item baru, `DynamoDBContext` menetapkan nomor versi awal 0. Jika Anda mengambil item yang ada, memperbarui satu atau beberapa properti, dan mencoba untuk menyimpan perubahan, operasi simpan hanya akan berhasil jika nomor versi di sisi klien dan sisi server cocok. `DynamoDBContext` menambahkan nomor versi. Anda tidak perlu mengatur nomor versi.
- Metode `Delete` menyediakan beban berlebih yang dapat mengambil nilai kunci primer atau objek sebagai parameter, seperti yang ditunjukkan dalam contoh kode C# berikut.

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);
```

Jika Anda menyediakan objek sebagai parameter, penghapusan berhasil hanya jika versi objek cocok dengan versi item sisi server yang sesuai. Namun, jika Anda memberikan nilai kunci primer sebagai parameter, `DynamoDBContext` tidak mengenali nomor versi apa pun, dan menghapus item tanpa melakukan pemeriksaan versi.

Perhatikan bahwa implementasi internal penguncian optimis dalam kode model persistensi objek menggunakan tindakan API pembaruan bersyarat dan penghapusan bersyarat di DynamoDB.

Menonaktifkan penguncian optimis

Untuk menonaktifkan penguncian optimis, gunakan properti konfigurasi `SkipVersionCheck`. Anda dapat mengatur properti ini saat membuat `DynamoDBContext`. Dalam kasus ini, penguncian optimis dinonaktifkan untuk permintaan apa pun yang Anda buat menggunakan konteks. Untuk informasi selengkapnya, lihat [Menentukan parameter opsional untuk `DynamoDBContext`](#).

Alih-alih menetapkan properti pada tingkat konteks, Anda dapat menonaktifkan penguncian optimis untuk operasi tertentu, seperti yang ditunjukkan dalam contoh kode C# berikut. Contoh ini menggunakan konteks untuk menghapus item buku. Metode `Delete` menetapkan properti `SkipVersionCheck` opsional ke `true`, menonaktifkan pemeriksaan versioning.

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 });
```

Memetakan data arbitrer dengan DynamoDB Menggunakan model persistensi objek AWS SDK for .NET

Selain jenis .NET yang didukung (lihat [Jenis data yang didukung](#)), Anda dapat menggunakan jenis di aplikasi Anda yang tidak memiliki pemetaan langsung ke jenis Amazon DynamoDB. Model persistensi objek mendukung jenis penyimpanan data arbitrer selama Anda memberikan konverter untuk mengonversi data dari jenis arbitrer untuk jenis DynamoDB dan sebaliknya. Kode konverter mengubah data saat menyimpan dan memuat objek.

Anda dapat membuat jenis apa pun di sisi klien. Namun, data yang disimpan di tabel adalah salah satu jenis DynamoDB, dan selama kueri dan pemindaian, perbandingan data apa pun yang dibuat didasarkan pada data yang disimpan di DynamoDB.

Contoh kode C# berikut mendefinisikan kelas `Book` dengan properti `Id`, `Title`, `ISBN`, dan `Dimension`. Properti `Dimension` adalah bagian dari `DimensionType` yang mendeskripsikan properti `Height`, `Width`, dan `Thickness`. Contoh kode ini menyediakan metode konverter `ToEntry` dan `FromEntry` untuk mengonversi data antara jenis string `DimensionType` dan DynamoDB. Misalnya, saat menyimpan instans `Book`, konverter membuat sebuah string `Dimension` buku seperti

"8.5x11x.05". Saat Anda mengambil buku, konverter akan mengonversi string tersebut menjadi instans `DimensionType`.

Contoh ini memetakan jenis `Book` ke tabel `ProductCatalog`. Ini menyimpan instans `Book` sampel, mengambilnya, memperbarui dimensinya, dan menyimpan `Book` yang terbaru lagi.

Untuk step-by-step instruksi untuk menguji contoh berikut, lihat [Contoh kode .NET](#).

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;
}
}
```

Operasi batch menggunakan model persistensi objek AWS SDK for .NET

Penulisan batch: Menempatkan dan menghapus beberapa item

Untuk memasukkan atau menghapus beberapa objek dari tabel dalam satu permintaan, lakukan langkah berikut:

- Jalankan metode `createBatchWrite` dari `DynamoDBContext`, dan buat instans kelas `BatchWrite`.
- Tentukan item yang ingin Anda masukkan atau hapus.
 - Untuk memasukkan satu atau beberapa item, gunakan metode `AddPutItem` atau `AddPutItems`.
 - Untuk menghapus satu atau beberapa item, Anda dapat menentukan kunci primer item tersebut atau objek sisi klien yang dipetakan ke item yang ingin Anda hapus. Gunakan metode

AddDeleteItem, AddDeleteItems, dan AddDeleteKey untuk menentukan daftar item yang akan menghapus.

- Panggil metode `BatchWrite.Execute` untuk memasukkan dan menghapus semua item tertentu dari tabel.

Note

Jika menggunakan model persistensi objek, Anda dapat menentukan berapa pun jumlah operasi dalam batch. Namun, perhatikan bahwa Amazon DynamoDB membatasi jumlah operasi dalam batch dan ukuran total batch dalam operasi batch. Untuk informasi selengkapnya tentang batasan spesifik, lihat [BatchWriteItem](#). Jika API mendeteksi bahwa permintaan tulis batch Anda melebihi jumlah yang diizinkan permintaan tulis atau melebihi ukuran muatan HTTP maksimum yang diizinkan, batch akan dipecah menjadi beberapa batch yang lebih kecil. Selain itu, jika respons ke penulisan batch mengembalikan item yang belum diproses, API otomatis mengirimkan permintaan batch lain dengan item yang belum diproses.

Katakanlah Anda mendefinisikan kelas C# dan kelas `Book` yang dipetakan ke tabel `ProductCatalog` di DynamoDB. Contoh kode C# berikut menggunakan objek `BatchWrite` untuk mengunggah dua item dan menghapus satu item dari tabel `ProductCatalog`.

Example

```
DynamoDBContext context = new DynamoDBContext(client);
var bookBatch = context.CreateBatchWrite<Book>();

// 1. Specify two books to add.
Book book1 = new Book
{
    Id = 902,
    ISBN = "902-11-11-1111",
    ProductCategory = "Book",
    Title = "My book3 in batch write"
};
Book book2 = new Book
{
    Id = 903,
    ISBN = "903-11-11-1111",
    ProductCategory = "Book",
```

```
Title = "My book4 in batch write"
};

bookBatch.AddPutItems(new List<Book> { book1, book2 });

// 2. Specify one book to delete.
bookBatch.AddDeleteKey(111);

bookBatch.Execute();
```

Untuk memasukkan atau menghapus objek dari beberapa tabel, lakukan langkah berikut:

- Buat satu instans kelas `BatchWrite` untuk setiap jenis dan tentukan item yang ingin Anda masukkan atau hapus seperti yang dijelaskan di bagian sebelumnya.
- Buat instans `MultiTableBatchWrite` menggunakan salah satu metode berikut:
 - Jalankan metode `Combine` pada salah satu objek `BatchWrite` yang Anda buat pada langkah sebelumnya.
 - Buat instans jenis `MultiTableBatchWrite` dengan menyediakan daftar objek `BatchWrite`.
 - Jalankan metode `CreateMultiTableBatchWrite` `DynamoDBContext` dan teruskan dalam daftar objek `BatchWrite`.
- Panggil metode `Execute` `MultiTableBatchWrite`, yang melakukan operasi masukkan dan hapus yang ditentukan pada berbagai tabel.

Misalkan Anda mendefinisikan kelas C# `Forum` dan `Thread` yang dipetakan ke tabel `Forum` dan `Thread` di `DynamoDB`. Selain itu, anggaplah kelas `Thread` sudah mengaktifkan `versioning`. Karena `versioning` tidak didukung saat menggunakan operasi batch, Anda harus secara eksplisit menonaktifkan `versioning` seperti yang ditunjukkan dalam contoh kode C# berikut. Contoh penggunaan objek `MultiTableBatchWrite` untuk melakukan pembaruan multi-tabel.

Example

```
DynamoDBContext context = new DynamoDBContext(client);
// Create BatchWrite objects for each of the Forum and Thread classes.
var forumBatch = context.CreateBatchWrite<Forum>();

DynamoDBOperationConfig config = new DynamoDBOperationConfig();
config.SkipVersionCheck = true;
var threadBatch = context.CreateBatchWrite<Thread>(config);
```



```
// 1. New Forum item.
Forum newForum = new Forum
{
    Name = "Test BatchWrite Forum",
    Threads = 0
};
forumBatch.AddPutItem(newForum);

// 2. Specify a forum to delete by specifying its primary key.
forumBatch.AddDeleteKey("Some forum");

// 3. New Thread item.
Thread newThread = new Thread
{
    ForumName = "Amazon S3 forum",
    Subject = "My sample question",
    KeywordTags = new List<string> { "Amazon S3", "Bucket" },
    Message = "Message text"
};

threadBatch.AddPutItem(newThread);

// Now run multi-table batch write.
var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);
superBatch.Execute();
```

Untuk contoh pekerjaan, lihat [Contoh: Operasi tulis batch menggunakan model persistensi objek AWS SDK for .NET](#).

Note

API batch DynamoDB membatasi jumlah tulis dalam batch dan juga membatasi ukuran batch. Untuk informasi lebih lanjut, lihat [BatchWriteItem](#). Jika menggunakan API model persistensi objek .NET, Anda dapat menentukan berapa pun jumlah operasi. Namun, jika jumlah operasi dalam batch atau ukurannya melebihi batas, API .NET memecah permintaan penulisan batch menjadi batch yang lebih kecil dan mengirimkan beberapa permintaan penulisan batch ke DynamoDB.

Batch get: Mendapatkan beberapa item

Untuk mengambil beberapa item dari tabel dalam satu permintaan, lakukan langkah berikut:

- Buat instans kelas `CreateBatchGet`.
- Tentukan daftar kunci primer yang akan diambil.
- Panggil metode `Execute`. Respons ini mengembalikan item dalam properti `Results`.

Contoh kode C# berikut mengambil tiga item dari tabel `ProductCatalog`. Item dalam hasil tidak selalu dalam urutan yang sama dengan kunci utama yang Anda tentukan.

Example

```
DynamoDBContext context = new DynamoDBContext(client);
var bookBatch = context.CreateBatchGet<ProductCatalog>();
bookBatch.AddKey(101);
bookBatch.AddKey(102);
bookBatch.AddKey(103);
bookBatch.Execute();
// Process result.
Console.WriteLine(bookBatch.Results.Count);
Book book1 = bookBatch.Results[0];
Book book2 = bookBatch.Results[1];
Book book3 = bookBatch.Results[2];
```

Untuk mengambil objek dari beberapa tabel, lakukan langkah berikut:

- Untuk setiap jenis, buat instans jenis `CreateBatchGet` dan berikan nilai kunci primer yang ingin Anda ambil dari setiap tabel.
- Buat instans kelas `MultiTableBatchGet` menggunakan salah satu metode berikut:
 - Jalankan metode `Combine` pada salah satu objek `BatchGet` yang Anda buat pada langkah sebelumnya.
 - Buat instans jenis `MultiBatchGet` dengan menyediakan daftar objek `BatchGet`.
 - Jalankan metode `CreateMultiTableBatchGet` `DynamoDBContext` dan teruskan dalam daftar objek `BatchGet`.
- Panggil metode `Execute` `MultiTableBatchGet`, yang mengembalikan hasil yang memiliki jenis dalam objek `BatchGet` individual.

Contoh kode C# berikut mengambil beberapa item dari tabel `Order` dan `OrderDetail` menggunakan metode `CreateBatchGet`.

Example

```
var orderBatch = context.CreateBatchGet<Order>();
orderBatch.AddKey(101);
orderBatch.AddKey(102);

var orderDetailBatch = context.CreateBatchGet<OrderDetail>();
orderDetailBatch.AddKey(101, "P1");
orderDetailBatch.AddKey(101, "P2");
orderDetailBatch.AddKey(102, "P3");
orderDetailBatch.AddKey(102, "P1");

var orderAndDetailSuperBatch = orderBatch.Combine(orderDetailBatch);
orderAndDetailSuperBatch.Execute();

Console.WriteLine(orderBatch.Results.Count);
Console.WriteLine(orderDetailBatch.Results.Count);

Order order1 = orderBatch.Results[0];
Order order2 = orderBatch.Results[1];
OrderDetail orderDetail1 = orderDetailBatch.Results[0];
```

Contoh: Operasi CRUD menggunakan model persistensi objek AWS SDK for .NET

Contoh kode C# berikut menyatakan kelas `Book` dengan properti `Id`, `Title`, `ISBN`, dan `Authors`. Contoh ini menggunakan atribut persistensi objek untuk memetakan properti ini ke tabel `ProductCatalog` di Amazon DynamoDB. Contoh ini kemudian menggunakan `DynamoDBContext` untuk mengilustrasikan operasi buat, baca, perbarui, dan hapus (CRUD) yang umum. Contoh ini membuat instans `Book` sampel dan menyimpannya ke tabel `ProductCatalog`. Kemudian, contoh ini akan mengambil item buku dan memperbarui properti `ISBN` dan `Authors`-nya. Perhatikan bahwa pembaruan menggantikan daftar penulis yang ada. Terakhir, contoh menghapus item buku.

Untuk informasi selengkapnya tentang tabel `ProductCatalog` yang digunakan dalam contoh ini, lihat [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#). Untuk step-by-step instruksi untuk menguji contoh berikut, lihat [Contoh kode .NET](#).

Note

Contoh berikut tidak berfungsi dengan inti .NET karena tidak mendukung metode sinkron. Untuk informasi selengkapnya, lihat [API Asinkron AWS untuk .NET](#).

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class HighLevelItemCRUD
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                DynamoDBContext context = new DynamoDBContext(client);
                TestCRUDOperations(context);
                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 TestCRUDOperations(DynamoDBContext 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.
context.Save(myBook);
// Retrieve the book.
Book bookRetrieved = context.Load<Book>(bookID);

// Update few properties.
bookRetrieved.ISBN = "222-2222221001";
bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
x" }; // Replace existing authors list with this.
context.Save(bookRetrieved);

// Retrieve the updated book. This time add the optional ConsistentRead
parameter using DynamoDBContextConfig object.
Book updatedBook = context.Load<Book>(bookID, new DynamoDBContextConfig
{
    ConsistentRead = true
});

// Delete the book.
context.Delete<Book>(bookID);
// Try to retrieve deleted book. It should return null.
Book deletedBook = context.Load<Book>(bookID, new DynamoDBContextConfig
{
    ConsistentRead = true
});
if (deletedBook == null)
    Console.WriteLine("Book is deleted");
}
}

[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")] //String Set datatype
    public List<string> BookAuthors
    {
        get; set;
    }
}
}
```

Contoh: Operasi tulis batch menggunakan model persistensi objek AWS SDK for .NET

Contoh kode C# berikut menyatakan kelas Book, Forum, Thread, dan Reply serta memetakannya ke tabel Amazon DynamoDB menggunakan atribut model persistensi objek.

Contoh kemudian menggunakan DynamoDBContext untuk menggambarkan operasi tulis batch berikut:

- Objek BatchWrite untuk memasukkan dan menghapus item buku dari tabel ProductCatalog.
- Objek MultiTableBatchWrite untuk memasukkan dan menghapus item dari tabel Forum dan Thread.

Untuk informasi selengkapnya tentang tabel yang digunakan dalam contoh ini, lihat [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#). Untuk step-by-step instruksi untuk menguji contoh berikut, lihat [Contoh kode .NET](#).

Note

Contoh berikut tidak berfungsi dengan inti .NET karena tidak mendukung metode sinkron. Untuk informasi selengkapnya, lihat [API Asinkron AWS untuk .NET](#).

Example

```
using System;
using System.Collections.Generic;
```

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class HighLevelBatchWriteItem
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                DynamoDBContext context = new DynamoDBContext(client);
                SingleTableBatchWrite(context);
                MultiTableBatchWrite(context);
            }
            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 SingleTableBatchWrite(DynamoDBContext 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("Performing batch write in SingleTableBatchWrite().");
    bookBatch.Execute();
}

private static void MultiTableBatchWrite(DynamoDBContext context)
{
    // 1. New Forum item.
    Forum newForum = new Forum
    {
        Name = "Test BatchWrite Forum",
        Threads = 0
    };
    var forumBatch = context.CreateBatchWrite<Forum>();
    forumBatch.AddPutItem(newForum);

    // 2. 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().");
    superBatch.Execute();
}
}
```



```
[DynamoDBTable("Reply")]
public class Reply
{
    [DynamoDBHashKey] //Partition key
    public string Id
    {
        get; set;
    }

    [DynamoDBRangeKey] //Sort key
    public DateTime ReplyDateTime
    {
        get; set;
    }

    // Properties included implicitly.
    public string Message
    {
        get; set;
    }
    // Explicit property mapping with object persistence model attributes.
    [DynamoDBProperty("LastPostedBy")]
    public string PostedBy
    {
        get; set;
    }
    // Property to store version number for optimistic locking.
    [DynamoDBVersion]
    public int? Version
    {
        get; set;
    }
}

[DynamoDBTable("Thread")]
public class Thread
{
    // PK mapping.
    [DynamoDBHashKey] //Partition key
    public string ForumName
    {
        get; set;
    }
}
```

```
[DynamoDBRangeKey]    //Sort key
public String Subject
{
    get; set;
}
// Implicit mapping.
public string Message
{
    get; set;
}
public string LastPostedBy
{
    get; set;
}
public int Views
{
    get; set;
}
public int Replies
{
    get; set;
}
public bool Answered
{
    get; set;
}
public DateTime LastPostedDateTime
{
    get; set;
}
// Explicit mapping (property and table attribute names are different.
[DynamoDBProperty("Tags")]
public List<string> KeywordTags
{
    get; set;
}
// Property to store version number for optimistic locking.
[DynamoDBVersion]
public int? Version
{
    get; set;
}
}
```

```
[DynamoDBTable("Forum")]
public class Forum
{
    [DynamoDBHashKey] //Partition key
    public string Name
    {
        get; set;
    }
    // All the following properties are explicitly mapped,
    // only to show how to provide mapping.
    [DynamoDBProperty]
    public int Threads
    {
        get; set;
    }
    [DynamoDBProperty]
    public int Views
    {
        get; set;
    }
    [DynamoDBProperty]
    public string LastPostBy
    {
        get; set;
    }
    [DynamoDBProperty]
    public DateTime LastPostDateTime
    {
        get; set;
    }
    [DynamoDBProperty]
    public int Messages
    {
        get; set;
    }
}

[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey] //Partition key
    public int Id
    {
        get; set;
    }
}
```

```
    }
    public string Title
    {
        get; set;
    }
    public string ISBN
    {
        get; set;
    }
    public int Price
    {
        get; set;
    }
    public string PageCount
    {
        get; set;
    }
    public string ProductCategory
    {
        get; set;
    }
    public bool InPublication
    {
        get; set;
    }
}
}
```

Contoh: Kueri dan pemindaian di DynamoDB menggunakan model persistensi objek AWS SDK for .NET

Contoh C# di bagian ini menetapkan kelas dan peta berikut ke tabel di DynamoDB. Untuk informasi selengkapnya tentang membuat tabel yang digunakan dalam contoh ini, lihat [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

- Kelas Book dipetakan ke tabel ProductCatalog.
- Kelas Forum, Thread, dan Reply dipetakan ke tabel dengan nama yang sama.

Contoh ini kemudian menjalankan operasi kueri dan pemindaian berikut menggunakan DynamoDBContext.

- Dapatkan buku dengan Id.

Tabel `ProductCatalog` memiliki Id sebagai kunci primernya. Tabel tersebut tidak memiliki kunci urutan sebagai bagian dari kunci primernya. Oleh karena itu, Anda tidak dapat mengkueri tabel. Anda bisa mendapatkan item menggunakan nilai Id-nya.

- Jalankan kueri berikut terhadap tabel `Reply`. (Kunci primer tabel `Reply` terdiri dari atribut `Id` dan `ReplyDateTime`. `ReplyDateTime` adalah kunci urutan. Oleh karena itu, Anda dapat mengkueri tabel ini.)
 - Temukan balasan untuk utas forum yang diposting dalam 15 hari terakhir.
 - Temukan balasan untuk utas forum yang diposting dalam rentang tanggal tertentu.
- Pindai tabel `ProductCatalog` untuk menemukan buku yang harganya kurang dari nol.

Untuk alasan performa, Anda harus menggunakan operasi kueri, bukan operasi pindai. Namun, Anda mungkin perlu memindai tabel. Misalkan ada kesalahan entri data dan salah satu harga buku diatur ke kurang dari 0. Contoh ini memindai tabel `ProductCategory` untuk menemukan item buku (`ProductCategory` adalah buku) dengan harga kurang dari 0.

Untuk petunjuk tentang cara membuat sampel yang berfungsi, lihat [Contoh kode .NET](#).

Note

Contoh berikut tidak berfungsi dengan inti .NET karena tidak mendukung metode sinkron. Untuk informasi selengkapnya, lihat [API asinkron AWS untuk .NET](#).

Example

```
using System;
using System.Collections.Generic;
using System.Configuration;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class HighLevelQueryAndScan
    {
```

```
private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

static void Main(string[] args)
{
    try
    {
        DynamoDBContext context = new DynamoDBContext(client);
        // Get an item.
        GetBook(context, 101);

        // Sample forum and thread to test queries.
        string forumName = "Amazon DynamoDB";
        string threadSubject = "DynamoDB Thread 1";
        // Sample queries.
        FindRepliesInLast15Days(context, forumName, threadSubject);
        FindRepliesPostedWithinTimePeriod(context, forumName, threadSubject);

        // Scan table.
        FindProductsPricedLessThanZero(context);
        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 GetBook(DynamoDBContext context, int productId)
{
    Book bookItem = context.Load<Book>(productId);

    Console.WriteLine("\nGetBook: Printing result.....");
    Console.WriteLine("Title: {0} \n No.Of threads:{1} \n No. of messages:
{2}",
        bookItem.Title, bookItem.ISBN, bookItem.PageCount);
}

private static void FindRepliesInLast15Days(DynamoDBContext context,
        string forumName,
        string threadSubject)
{
    string replyId = forumName + "#" + threadSubject;
    DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
    IEnumerable<Reply> latestReplies =
```

```
        context.Query<Reply>(replyId, QueryOperator.GreaterThan,
twoWeeksAgoDate);
        Console.WriteLine("\nFindRepliesInLast15Days: Printing result.....");
        foreach (Reply r in latestReplies)
            Console.WriteLine("{0}\t{1}\t{2}\t{3}", r.Id, r.PostedBy, r.Message,
r.ReplyDateTime);
    }

    private static void FindRepliesPostedWithinTimePeriod(DynamoDBContext context,
        string forumName,
        string threadSubject)
    {
        string forumId = forumName + "#" + threadSubject;
        Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: Printing
result.....");

        DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
        DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);

        IEnumerable<Reply> repliesInAPeriod = context.Query<Reply>(forumId,
            QueryOperator.Between, startDate, endDate);
        foreach (Reply r in repliesInAPeriod)
            Console.WriteLine("{0}\t{1}\t{2}\t{3}", r.Id, r.PostedBy, r.Message,
r.ReplyDateTime);
    }

    private static void FindProductsPricedLessThanZero(DynamoDBContext context)
    {
        int price = 0;
        IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
            new ScanCondition("Price", ScanOperator.LessThan, price),
            new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
        );
        Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
result.....");
        foreach (Book r in itemsWithWrongPrice)
            Console.WriteLine("{0}\t{1}\t{2}\t{3}", r.Id, r.Title, r.Price,
r.ISBN);
    }
}

[DynamoDBTable("Reply")]
public class Reply
{
```

```
[DynamoDBHashKey] //Partition key
public string Id
{
    get; set;
}

[DynamoDBRangeKey] //Sort key
public DateTime ReplyDateTime
{
    get; set;
}

// Properties included implicitly.
public string Message
{
    get; set;
}
// Explicit property mapping with object persistence model attributes.
[DynamoDBProperty("LastPostedBy")]
public string PostedBy
{
    get; set;
}
// Property to store version number for optimistic locking.
[DynamoDBVersion]
public int? Version
{
    get; set;
}
}

[DynamoDBTable("Thread")]
public class Thread
{
    // Partition key mapping.
    [DynamoDBHashKey] //Partition key
    public string ForumName
    {
        get; set;
    }
    [DynamoDBRangeKey] //Sort key
    public DateTime Subject
    {
        get; set;
    }
}
```



```
    }
    // Implicit mapping.
    public string Message
    {
        get; set;
    }
    public string LastPostedBy
    {
        get; set;
    }
    public int Views
    {
        get; set;
    }
    public int Replies
    {
        get; set;
    }
    public bool Answered
    {
        get; set;
    }
    public DateTime LastPostedDateTime
    {
        get; set;
    }
    // Explicit mapping (property and table attribute names are different).
    [DynamoDBProperty("Tags")]
    public List<string> KeywordTags
    {
        get; set;
    }
    // Property to store version number for optimistic locking.
    [DynamoDBVersion]
    public int? Version
    {
        get; set;
    }
}

[DynamoDBTable("Forum")]
public class Forum
{
    [DynamoDBHashKey]
```

```
    public string Name
    {
        get; set;
    }
    // All the following properties are explicitly mapped
    // to show how to provide mapping.
    [DynamoDBProperty]
    public int Threads
    {
        get; set;
    }
    [DynamoDBProperty]
    public int Views
    {
        get; set;
    }
    [DynamoDBProperty]
    public string LastPostBy
    {
        get; set;
    }
    [DynamoDBProperty]
    public DateTime LastPostDateTime
    {
        get; set;
    }
    [DynamoDBProperty]
    public int Messages
    {
        get; set;
    }
}

[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey] //Partition key
    public int Id
    {
        get; set;
    }
    public string Title
    {
        get; set;
    }
}
```

```
    }  
    public string ISBN  
    {  
        get; set;  
    }  
    public int Price  
    {  
        get; set;  
    }  
    public string PageCount  
    {  
        get; set;  
    }  
    public string ProductCategory  
    {  
        get; set;  
    }  
    public bool InPublication  
    {  
        get; set;  
    }  
}  
}
```

Menjalankan contoh kode dalam Panduan Pengembang ini

SDK AWS memberikan dukungan luas untuk Amazon DynamoDB dalam bahasa-bahasa berikut:

- [Jawa](#)
- [JavaScript di browser](#)
- [.NET](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Ruby](#)
- [C ++](#)
- [Pergi](#)
- [Android](#)

- [iOS](#)

Untuk segera memulai dengan bahasa ini, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Contoh kode dalam panduan developer ini menyediakan cakupan yang lebih mendalam tentang operasi DynamoDB, menggunakan bahasa pemrograman berikut:

- [Contoh kode Java](#)
- [Contoh kode .NET](#)

Sebelum dapat memulai dengan latihan ini, Anda perlu membuat akun AWS, dapatkan access key dan kunci rahasia, dan atur AWS Command Line Interface (AWS CLI) di komputer Anda. Untuk informasi selengkapnya, lihat [Menyiapkan DynamoDB \(layanan web\)](#).

Note

Jika menggunakan versi DynamoDB yang dapat diunduh, Anda harus menggunakan AWS CLI untuk membuat tabel dan data sampel. Anda juga perlu menentukan parameter `--endpoint-url` dengan masing-masing perintah AWS CLI. Untuk informasi selengkapnya, lihat [Mengatur Titik Akhir Lokal](#).

Membuat tabel dan memuat data untuk contoh kode di DynamoDB

Lihat di bawah untuk dasar-dasar pembuatan tabel di DynamoDB, memuat dalam kumpulan data sampel, mengkueri data, dan memperbarui data.

- [Langkah 2: Buat tabel](#)
- [Langkah 2: Tulis data ke tabel menggunakan konsol atau AWS CLI](#)
- [Langkah 3: Baca data dari Tabel](#)
- [Langkah 4: Perbarui data di Tabel](#)

Contoh kode Java

Topik

- [Java: Mengatur AWS Kredensi Anda](#)

- [Java: Mengatur AWS Wilayah dan Titik Akhir](#)

Panduan Developer ini berisi potongan kode Java dan ready-to-run program Anda dapat menemukan contoh kode ini di bagian berikut:

- [Bekerja dengan item dan atribut](#)
- [Bekerja dengan tabel dan data di DynamoDB](#)
- [Operasi kueri di DynamoDB](#)
- [Bekerja dengan pemindaian di DynamoDB](#)
- [Meningkatkan akses data dengan indeks sekunder](#)
- [Java 1.x: DynamoDBMapper](#)
- [Ubah tangkapan data DynamoDB Streams](#)

Anda dapat memulai dengan cepat menggunakan Eclipse dengan [AWS Toolkit for Eclipse](#). Selain IDE berfitur lengkap, Anda juga mendapatkan AWS SDK for Java dengan pembaruan otomatis, dan template yang telah dikonfigurasi untuk membuat aplikasi AWS.

Untuk menjalankan contoh kode Java (menggunakan Eclipse)

1. Unduh dan instal [Eclipse](#) IDE.
2. Unduh dan instal [AWS Toolkit for Eclipse](#).
3. Mulailah Eclipse, dan pada menu Eclipse, pilih File, Baru, lalu Lainnya.
4. Dalam Pilih wizard, pilih AWS, pilih Proyek Java AWS, lalu pilih Selanjutnya.
5. Dalam Buat Java AWS, lakukan hal berikut:
 - a. Dalam Nama proyek, masukkan nama untuk proyek Anda.
 - b. Dalam Pilih Akun, pilih profil kredensial Anda dari daftar.

Jika ini adalah pertama kalinya Anda menggunakan [AWS Toolkit for Eclipse](#), pilih Konfigurasi Akun AWS untuk menyiapkan kredensial AWS Anda.

6. Pilih Selesai untuk membuat proyek.
7. Dari menu Eclipse, pilih Berkas, Baru, kemudian Kelas.
8. Dalam Kelas Java, masukkan nama untuk kelas Anda di Nama (menggunakan nama yang sama sebagai contoh kode yang ingin Anda jalankan), dan kemudian pilih Selesai untuk membuat kelas.

9. Salin contoh kode dari halaman dokumentasi ke editor Eclipse.
10. Untuk menjalankan kode, pilih Jalankan pada menu Eclipse.

SDK for Java menyediakan klien thread-safe untuk bekerja dengan DynamoDB. Sebagai praktik terbaik, aplikasi Anda harus membuat satu klien dan menggunakan kembali klien antara thread.

Untuk informasi selengkapnya, lihat [AWS SDK for Java](#).

Note

Contoh kode dalam panduan ini dimaksudkan untuk digunakan dengan versi terbaru dari AWS SDK for Java.

Jika menggunakan AWS Toolkit for Eclipse, Anda dapat mengonfigurasi pembaruan otomatis untuk SDK for Java. Untuk melakukan hal ini di Eclipse, buka Preferensi dan pilih AWS Toolkit, AWS SDK for Java, Unduh SDK baru secara otomatis.

Java: Mengatur AWS Kredensi Anda

SDK for Java mengharuskan Anda memberikan kredensial AWS untuk aplikasi Anda pada saat runtime. Contoh kode dalam panduan ini menganggap bahwa Anda menggunakan file AWS kredensial, seperti yang dijelaskan dalam [Menyiapkan AWS Kredensial Anda](#) dalam Panduan AWS SDK for Java Developer.

Berikut ini adalah contoh file kredensial AWS bernama `~/.aws/credentials`, di mana karakter tilde (~) mewakili direktori beranda Anda.

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java: Mengatur AWS Wilayah dan Titik Akhir

Secara default, contoh kode mengakses DynamoDB di Wilayah US West (Oregon). Anda dapat mengubah Wilayah dengan memodifikasi properti `AmazonDynamoDB`.

Contoh kode berikut menunjukkan `AmazonDynamoDB` baru.

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.regions.Regions;
```

```
...
// This client will default to US West (Oregon)
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
.withRegion(Regions.US_WEST_2)
.build();
```

Anda dapat menggunakan metode `withRegion` untuk menjalankan kode Anda terhadap DynamoDB di Wilayah ketersediaan mana pun Untuk daftar lengkap, lihat [AWSwilayah dan titik akhir](#) di bagian Referensi Umum Amazon Web Services.

Jika Anda ingin menjalankan contoh kode menggunakan DynamoDB secara lokal pada komputer Anda, atur titik akhir sebagai berikut.

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();
```

Contoh kode .NET

Topik

- [.NET: MengaturAWS Kredensi Anda](#)
- [.NET: MengaturAWS Wilayah dan Titik Akhir](#)

Contoh .NET ready-to-run Anda dapat menemukan contoh kode ini di bagian berikut:

- [Bekerja dengan item dan atribut](#)

- [Bekerja dengan tabel dan data di DynamoDB](#)
- [Operasi kueri di DynamoDB](#)
- [Bekerja dengan pemindaian di DynamoDB](#)
- [Meningkatkan akses data dengan indeks sekunder](#)
- [.NET: Model dokumen](#)
- [.NET: Model persistensi objek](#)
- [Ubah tangkapan data DynamoDB Streams](#)

Anda dapat memulai dengan cepat menggunakan AWS SDK for .NET dengan Toolkit for Visual Studio.

Untuk menjalankan contoh kode .NET (menggunakan Visual Studio)

1. Unduh dan instal [Studio Microsoft Visual](#).
2. Unduh dan instal [Toolkit for Visual Studio](#).
3. Mulai Visual Studio. Pilih Berkas, Baru, Proyek.
4. Dalam Proyek Baru, pilih Proyek Kosong AWS, lalu pilih OK.
5. Dalam Kredensial Akses AWS, pilih Gunakan profil yang ada, pilih kredensial profil Anda dari daftar, lalu pilih OK.

Jika ini adalah pertama kalinya Anda menggunakan Toolkit for Visual Studio, pilih Menggunakan profil baru untuk menyiapkan kredensial AWS Anda.

6. Dalam proyek Visual Studio, pilih tab untuk kode sumber program Anda (`Program.cs`). Salin contoh kode dari halaman dokumentasi ke editor Visual Studio, menggantikan kode lain yang Anda lihat di editor.
7. Jika Anda melihat pesan kesalahan formulir Nama tipe atau namespace... tidak dapat ditemukan, Anda harus menginstal rakitan SDK AWS untuk DynamoDB sebagai berikut:
 - a. Dalam Solution Explorer, buka menu konteks (klik kanan) untuk proyek Anda, lalu pilih Mengelola NuGet Paket.
 - b. Dalam NuGet Package Manager, pilih Telusuri.
 - c. Di kotak pencarian, masukkan **AWSSDK.DynamoDBv2**, dan tunggu hingga pencarian selesai.
 - d. Pilih AWSSDK.DynamoDBv2, dan kemudian pilih Install.

- e. Saat penginstalan selesai, pilih opsi tab Program.cs untuk kembali ke program Anda.
8. Untuk menjalankan kode, pilih Mulai di toolbar Visual Studio.

AWS SDK for .NET menyediakan klien thread-safe untuk bekerja dengan DynamoDB. Sebagai praktik terbaik, aplikasi Anda harus membuat satu klien dan menggunakan kembali klien antara thread.

Untuk informasi selengkapnya, lihat [AWSSDK for .NET](#).

Note

Contoh kode dalam panduan ini dimaksudkan untuk digunakan dengan versi terbaru dari AWS SDK for .NET.

.NET: Mengatur AWS Kredensi Anda

Parameter AWS SDK for .NET mengharuskan Anda memberikan kredensial AWS untuk aplikasi Anda pada saat runtime. Contoh kode dalam panduan ini menganggap bahwa Anda menggunakan SDK Store untuk mengelola file AWS kredensial, seperti yang dijelaskan dalam [Menggunakan SDK store](#) dalam Panduan AWS SDK for .NET Developer.

Toolkit for Visual Studio mendukung beberapa set kredensial dari sejumlah akun. Setiap set disebut sebagai profil. Visual Studio menambahkan entri untuk file `App.config` proyek sehingga aplikasi Anda dapat menemukan kredensial AWS saat runtime.

Contoh berikut menunjukkan file `App.config` default yang dihasilkan ketika Anda membuat proyek baru menggunakan Toolkit for Visual Studio.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="AWSProfileName" value="default"/>
    <add key="AWSRegion" value="us-west-2" />
  </appSettings>
</configuration>
```

Pada saat runtime, program menggunakan set default kredensial AWS, sebagaimana ditentukan oleh entri `AWSProfileName`. Kredensial AWS sendiri disimpan di SDK Store dalam bentuk

terenkripsi. Toolkit for Visual Studio menyediakan antarmuka pengguna grafis untuk mengelola kredensial Anda, semua dari dalam Visual Studio. Untuk informasi selengkapnya, lihat [Menentukan kredensi](#) di PanduanAWS Toolkit for Visual Studio Pengguna.

Note

Secara default, contoh kode mengakses DynamoDB di Wilayah US West (Oregon). Anda dapat mengubah Wilayah dengan memodifikasi entri `AWSRegion` dalam berkas `App.config`. Anda dapat mengatur `AWSRegion` menjadi Wilayah mana pun di mana DynamoDB tersedia. Untuk daftar lengkap, lihat [AWSwilayah dan titik akhir](#) di bagian Referensi Umum Amazon Web Services.

.NET: MengaturAWS Wilayah dan Titik Akhir

Secara default, contoh kode mengakses DynamoDB di Wilayah US West (Oregon). Anda dapat mengubah Wilayah dengan memodifikasi entri `AWSRegion` dalam file `App.config`. Atau, Anda dapat mengubah Wilayah dengan memodifikasi properti `AmazonDynamoDBClient`.

Contoh kode berikut menunjukkan `AmazonDynamoDBClient` baru. Klien dimodifikasi sehingga kode berjalan terhadap DynamoDB di Wilayah yang berbeda.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// This client will access the US East 1 region.
clientConfig.RegionEndpoint = RegionEndpoint.USEast1;
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

Untuk daftar lengkap Wilayah, lihat [AWSwilayah dan titik akhir](#) di bagian Referensi Umum Amazon Web Services.

Jika Anda ingin menjalankan contoh kode menggunakan DynamoDB secara lokal pada komputer Anda, atur titik akhir sebagai berikut.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// Set the endpoint URL
clientConfig.ServiceURL = "http://localhost:8000";
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

Pemrograman Amazon DynamoDB dengan Python dan Boto3

Panduan ini memberikan orientasi kepada programmer yang ingin menggunakan Amazon DynamoDB dengan Python. Pelajari tentang berbagai lapisan abstraksi, manajemen konfigurasi, penanganan kesalahan, pengendalian kebijakan coba lagi, pengelolaan keep-alive, dan banyak lagi.

Topik

- [Tentang Boto](#)
- [Menggunakan dokumentasi Boto](#)
- [Memahami lapisan abstraksi klien dan sumber daya](#)
- [Menggunakan sumber daya tabel batch_writer](#)
- [Contoh kode tambahan yang mengeksplorasi lapisan klien dan sumber daya](#)
- [Memahami bagaimana objek Klien dan Sumber Daya berinteraksi dengan sesi dan utas](#)
- [Menyesuaikan objek Config](#)
- [Penanganan kesalahan](#)
- [Pencatatan log](#)
- [Kait acara](#)
- [Pagination dan Paginator](#)
- [Pelayan](#)

Tentang Boto

Anda dapat mengakses DynamoDB dari Python dengan menggunakan SDK AWS resmi untuk Python, yang biasa disebut sebagai Boto3. Nama Boto (diucapkan boh-toh) berasal dari lumba-lumba air tawar asli Sungai Amazon. Perpustakaan Boto3 adalah versi utama ketiga perpustakaan, pertama kali dirilis pada tahun 2015. Perpustakaan Boto3 cukup besar, karena mendukung semua AWS layanan, bukan hanya DynamoDB. Orientasi ini hanya menargetkan bagian Boto3 yang relevan dengan DynamoDB.

Boto dikelola dan diterbitkan oleh AWS sebagai proyek sumber terbuka yang dihosting di GitHub [Ini dibagi menjadi dua paket: Botocore dan Boto3.](#)

- Botocore menyediakan fungsionalitas tingkat rendah. Di Botocore Anda akan menemukan kelas klien, sesi, kredensyal, konfigurasi, dan pengecualian.

- Boto3 dibangun di atas Botocore. Ini menawarkan antarmuka Pythonic tingkat yang lebih tinggi dan lebih banyak. Secara khusus, ini memperlihatkan tabel DynamoDB sebagai Sumber Daya dan menawarkan antarmuka yang lebih sederhana dan lebih elegan dibandingkan dengan antarmuka klien berorientasi layanan tingkat rendah.

Karena proyek ini di-host GitHub, Anda dapat melihat kode sumber, melacak masalah terbuka, atau mengirimkan masalah Anda sendiri.

Menggunakan dokumentasi Boto

Mulailah dengan dokumentasi Boto dengan sumber daya berikut:

- Mulailah dengan [bagian Quickstart](#) yang menyediakan titik awal yang solid untuk instalasi paket. Pergi ke sana untuk petunjuk tentang menginstal Boto3 jika belum (Boto3 sering tersedia secara otomatis dalam AWS layanan seperti). AWS Lambda
- Setelah itu, fokuslah pada panduan [DynamoDB](#) dokumentasi. Ini menunjukkan cara melakukan aktivitas DynamoDB dasar: membuat dan menghapus tabel, memanipulasi item, menjalankan operasi batch, menjalankan kueri, dan melakukan pemindaian. Contohnya menggunakan antarmuka sumber daya. Ketika Anda melihat `boto3.resource('dynamodb')` itu menunjukkan Anda menggunakan antarmuka sumber daya tingkat yang lebih tinggi.
- Setelah panduan, Anda dapat meninjau referensi [DynamoDB](#). Halaman arahan ini menyediakan daftar lengkap kelas dan metode yang tersedia untuk Anda. Di bagian atas, Anda akan melihat `DynamoDB.Client` kelas. Ini memberikan akses tingkat rendah ke semua operasi bidang kontrol dan bidang data. Di bagian bawah, lihat `DynamoDB.ServiceResource` kelasnya. Ini adalah antarmuka Pythonic tingkat yang lebih tinggi. Dengan itu Anda dapat membuat tabel, melakukan operasi batch di seluruh tabel, atau mendapatkan `DynamoDB.ServiceResource.Table` instance untuk tindakan khusus tabel.

Memahami lapisan abstraksi klien dan sumber daya

Dua antarmuka yang akan Anda gunakan adalah antarmuka klien dan antarmuka sumber daya.

- Antarmuka klien tingkat rendah menyediakan pemetaan 1-ke-1 ke API layanan yang mendasarinya. Setiap API yang ditawarkan oleh DynamoDB tersedia melalui klien. Ini berarti antarmuka klien dapat menyediakan fungsionalitas yang lengkap, tetapi seringkali lebih bertele-tele dan kompleks untuk digunakan.

- Antarmuka sumber daya tingkat yang lebih tinggi tidak menyediakan pemetaan 1-ke-1 dari API layanan yang mendasarinya. Namun, ini menyediakan metode yang membuatnya lebih nyaman bagi Anda untuk mengakses layanan seperti `batch_writer`.

Berikut adalah contoh menyisipkan item menggunakan antarmuka klien. Perhatikan bagaimana semua nilai dilewatkan sebagai peta dengan kunci yang menunjukkan tipenya ('S' untuk string, 'N' untuk nomor) dan nilainya sebagai string. Ini dikenal sebagai format DynamoDB JSON.

```
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 ...
    }
)
```

Berikut adalah `PutItem` operasi yang sama menggunakan antarmuka sumber daya. Pengetikan data tersirat:

```
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 ...
    }
)
```

Jika diperlukan, Anda dapat mengonversi antara JSON biasa dan DynamoDB JSON menggunakan `TypeSerializer` kelas dan yang disediakan dengan `boto3`: `TypeDeserializer`

```
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()
    }
```

Berikut adalah cara melakukan query menggunakan antarmuka klien. Ini mengekspresikan query sebagai konstruksi JSON. Ini menggunakan `KeyConditionExpression` string yang membutuhkan substitusi variabel untuk menangani konflik kata kunci potensial:

```
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',
    }
)
```

Operasi kueri yang sama menggunakan antarmuka sumber daya dapat dipersingkat dan disederhanakan:

```
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')
)
```

Sebagai contoh terakhir, bayangkan Anda ingin mendapatkan perkiraan ukuran tabel (yang merupakan metadata yang disimpan di atas meja yang diperbarui setiap 6 jam). Dengan antarmuka klien, Anda melakukan `describe_table()` operasi dan menarik jawaban dari struktur JSON yang dikembalikan:

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.describe_table(TableName='YourTableName')
size = response['Table']['TableSizeBytes']
```

Dengan antarmuka sumber daya, tabel melakukan operasi deskripsi secara implisit dan menyajikan data secara langsung sebagai atribut:

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')
size = table.table_size_bytes
```

Note

Saat mempertimbangkan apakah akan mengembangkan menggunakan antarmuka klien atau sumber daya, ketahuilah bahwa fitur baru tidak akan ditambahkan ke antarmuka sumber

daya per [dokumentasi sumber daya](#): “Tim SDK AWS Python tidak bermaksud menambahkan fitur baru ke antarmuka sumber daya di boto3. Antarmuka yang ada akan terus beroperasi selama siklus hidup boto3. Pelanggan dapat menemukan akses ke fitur layanan yang lebih baru melalui antarmuka klien.

Menggunakan sumber daya tabel `batch_writer`

Satu kenyamanan yang hanya tersedia dengan sumber daya tabel tingkat yang lebih tinggi adalah `batch_writer` DynamoDB mendukung operasi batch write yang memungkinkan hingga 25 operasi put atau delete dalam satu permintaan jaringan. Batching seperti ini meningkatkan efisiensi dengan meminimalkan perjalanan pulang-pergi jaringan.

Dengan pustaka klien tingkat rendah, Anda menggunakan `client.batch_write_item()` operasi untuk menjalankan batch. Anda harus membagi pekerjaan Anda secara manual menjadi 25 batch. Setelah setiap operasi, Anda juga harus meminta untuk menerima daftar item yang belum diproses (beberapa operasi penulisan mungkin berhasil sementara yang lain bisa gagal). Anda kemudian harus meneruskan item yang belum diproses itu lagi ke `batch_write_item()` operasi selanjutnya. Ada sejumlah besar kode boilerplate.

Metode [Table.batch_writer](#) menciptakan manajer konteks untuk menulis objek dalam batch. Ini menyajikan antarmuka di mana tampaknya seolah-olah Anda sedang menulis item satu per satu, tetapi secara internal itu buffering dan mengirim item dalam batch. Ini juga menangani percobaan ulang item yang belum diproses secara implisit.

```
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)
```

Contoh kode tambahan yang mengeksplorasi lapisan klien dan sumber daya

Anda juga dapat merujuk ke repositori contoh kode berikut yang mengeksplorasi penggunaan berbagai fungsi, menggunakan klien dan sumber daya:

- [Contoh kode AWS aksi tunggal resmi.](#)
- [Contoh kode AWS berorientasi skenario resmi.](#)
- [Contoh kode aksi tunggal yang dikelola komunitas.](#)

Memahami bagaimana objek Klien dan Sumber Daya berinteraksi dengan sesi dan utas

Objek Resource tidak aman untuk utas dan tidak boleh dibagikan di seluruh utas atau proses. Lihat [panduan tentang Sumber Daya](#) untuk lebih jelasnya.

Objek Klien, sebaliknya, umumnya aman untuk utas, kecuali untuk fitur lanjutan tertentu. Lihat [panduan tentang Klien](#) untuk lebih jelasnya.

Objek Session tidak aman untuk utas. Jadi, setiap kali Anda membuat Klien atau Sumber Daya di lingkungan multi-utas, Anda harus membuat Sesi baru terlebih dahulu dan kemudian membuat Klien atau Sumber Daya dari Sesi. Lihat [panduan tentang Sesi](#) untuk lebih jelasnya.

Saat Anda memanggil `boto3.resource()`, Anda secara implisit menggunakan Sesi default. Ini nyaman untuk menulis kode single-threaded. Saat menulis kode multi-utas, Anda harus terlebih dahulu membuat Sesi baru untuk setiap utas dan kemudian mengambil sumber daya dari Sesi itu:

```
# Explicitly create a new Session for this thread
session = boto3.Session()
dynamodb = session.resource('dynamodb')
```

Menyesuaikan objek Config

Saat membuat objek Klien atau Sumber Daya, Anda dapat meneruskan parameter bernama opsional untuk menyesuaikan perilaku. Parameter bernama `config` membuka berbagai fungsi. Ini adalah instance dari `botocore.client.Config` dan [dokumentasi referensi untuk Config](#) menunjukkan semua yang diekspos untuk Anda kendalikan. [Panduan untuk Konfigurasi](#) memberikan gambaran yang baik.

Note

Anda dapat memodifikasi banyak pengaturan perilaku ini di tingkat Sesi, dalam file AWS konfigurasi, atau sebagai variabel lingkungan.

Config untuk batas waktu

Salah satu penggunaan konfigurasi khusus adalah untuk menyesuaikan perilaku jaringan:

- `connect_timeout` (float atau int) - Waktu dalam hitungan detik hingga pengecualian batas waktu dilemparkan ketika mencoba membuat koneksi. Bawaannya adalah 60 detik.
- `read_timeout` (float atau int) - Waktu dalam hitungan detik hingga pengecualian batas waktu dilemparkan ketika mencoba membaca dari koneksi. Bawaannya adalah 60 detik.

Batas waktu 60 detik berlebihan untuk DynamoDB. Ini berarti kesalahan jaringan sementara akan menyebabkan penundaan satu menit untuk klien sebelum dapat mencoba lagi. Kode berikut mempersingkat batas waktu menjadi satu detik:

```
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)
```

Untuk diskusi selengkapnya tentang batas waktu, lihat [Menyetel setelan permintaan HTTP SDK AWS Java untuk aplikasi DynamoDB yang sadar latensi](#). Perhatikan bahwa Java SDK memiliki konfigurasi batas waktu lebih banyak daripada Python.

Config untuk keep-alive

Jika Anda menggunakan botocore 1.27.84 atau yang lebih baru, Anda juga dapat mengontrol TCP Keep-Alive:

- `tcp_keepalive` (bool) - Mengaktifkan opsi soket TCP Keep-Alive yang digunakan saat membuat koneksi baru jika disetel ke (default ke). `True` `False` Ini hanya tersedia mulai dengan botocore 1.27.84.

Mengatur TCP Keep-Alive untuk `True` dapat mengurangi latensi rata-rata. Berikut contoh kode yang secara kondisional menetapkan TCP Keep-Alive ke `true` ketika Anda memiliki versi botocore yang tepat:

```
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 Keep-Alive berbeda dari HTTP Keep-Alive. Dengan TCP Keep-Alive, paket kecil dikirim oleh sistem operasi yang mendasarinya melalui koneksi soket untuk menjaga koneksi tetap hidup dan segera mendeteksi tetesan apa pun. Dengan HTTP Keep-Alive, koneksi web yang dibangun di atas soket yang mendasarinya akan digunakan kembali. HTTP Keep-Alive selalu diaktifkan dengan boto3.

Ada batasan berapa lama koneksi idle dapat tetap hidup. Pertimbangkan untuk mengirim permintaan berkala (katakanlah setiap menit) jika Anda memiliki koneksi idle tetapi ingin permintaan berikutnya menggunakan koneksi yang sudah dibuat.

Config untuk percobaan ulang

Konfigurasi juga menerima kamus yang disebut percobaan ulang di mana Anda dapat menentukan perilaku coba lagi yang Anda inginkan. Percobaan ulang terjadi dalam SDK ketika SDK menerima kesalahan dan kesalahan adalah tipe sementara. Jika kesalahan dicoba ulang secara internal (dan percobaan ulang akhirnya menghasilkan respons yang berhasil), tidak ada kesalahan yang terlihat dari perspektif kode panggilan, hanya latensi yang sedikit meningkat. Berikut adalah nilai yang dapat Anda tentukan:

- `max_attempts` — Bilangan bulat yang mewakili jumlah maksimum percobaan ulang yang akan dilakukan pada satu permintaan. Misalnya, menyetel nilai ini ke 2 akan mengakibatkan permintaan dicoba ulang paling banyak dua kali setelah permintaan awal. Menyetel nilai ini ke 0 akan menghasilkan tidak ada percobaan ulang yang pernah dicoba setelah permintaan awal.
- `total_max_attempts` — Bilangan bulat yang mewakili jumlah maksimum upaya total yang akan dilakukan pada satu permintaan. Ini termasuk permintaan awal, jadi nilai 1 menunjukkan bahwa tidak ada permintaan yang akan dicoba ulang. Jika `total_max_attempts` dan `max_attempts` keduanya disediakan, `total_max_attempts` diutamakan. `total_max_attempts` lebih disukai daripada `max_attempts` karena memetakan ke variabel `AWS_MAX_ATTEMPTS` lingkungan dan nilai file `max_attempts` konfigurasi.
- `mode` — String yang mewakili jenis mode coba lagi yang harus digunakan botocore. Nilai yang valid adalah:
 - `legacy` — Mode default. Menunggu 50 ms percobaan ulang pertama, kemudian menggunakan backoff eksponensial dengan faktor dasar 2. Untuk DynamoDB, ia melakukan hingga 10 upaya maksimal total (kecuali diganti dengan yang di atas).

 Note

Dengan backoff eksponensial, upaya terakhir akan menunggu hampir 13 detik.

- `standard` — Dinamakan standar karena lebih konsisten dengan AWS SDK lainnya. Menunggu waktu acak dari 0ms hingga 1.000 ms untuk percobaan ulang pertama. Jika percobaan lagi diperlukan, ia mengambil waktu acak lain dari 0ms menjadi 1.000 ms dan mengalikannya dengan 2. Jika percobaan ulang tambahan diperlukan, ia melakukan pengambilan acak yang sama dikalikan dengan 4, dan seterusnya. Setiap penantian dibatasi pada 20 detik. Mode ini akan melakukan percobaan ulang pada kondisi kegagalan yang lebih terdeteksi daripada Legacy mode. Untuk DynamoDB, ia melakukan hingga 3 upaya maksimal total (kecuali diganti dengan yang di atas).
- `adaptif` - Mode coba lagi eksperimental yang mencakup semua fungsionalitas mode standar tetapi menambahkan pelambatan sisi klien otomatis. Dengan pembatasan tingkat adaptif, SDK dapat memperlambat laju pengiriman permintaan untuk mengakomodasi kapasitas layanan dengan lebih baik. AWS Ini adalah mode sementara yang perilakunya mungkin berubah.

Definisi yang diperluas dari mode coba lagi ini dapat ditemukan di [panduan untuk mencoba ulang](#) serta dalam [topik perilaku Coba lagi dalam](#) referensi SDK.

Berikut adalah contoh yang secara eksplisit menggunakan kebijakan legacy coba lagi dengan maksimal 3 total permintaan (2 percobaan ulang):

```
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)
```

Karena DynamoDB adalah sistem latensi rendah yang sangat tersedia, Anda mungkin ingin lebih agresif dengan kecepatan percobaan ulang daripada yang diizinkan oleh kebijakan coba ulang bawaan. Anda dapat menerapkan kebijakan coba ulang Anda sendiri dengan menyetel upaya maksimal ke 0, menangkap pengecualian sendiri, dan mencoba lagi sebagaimana mestinya dari kode Anda sendiri alih-alih mengandalkan boto3 untuk melakukan percobaan ulang implisit.

Jika Anda mengelola kebijakan coba ulang Anda sendiri, Anda akan ingin membedakan antara throttle dan error:

- Throttle (ditunjukkan oleh `ProvisionedThroughputExceededException` atau `ThrottlingException`) menunjukkan layanan sehat yang memberi tahu Anda bahwa Anda telah melebihi kapasitas baca atau tulis pada tabel atau partisi DynamoDB. Setiap milidetik yang berlalu, sedikit lebih banyak kapasitas baca atau tulis tersedia, sehingga Anda dapat mencoba lagi dengan cepat (seperti setiap 50 ms) untuk mencoba mengakses kapasitas yang baru dirilis. Dengan throttle, Anda tidak memerlukan backoff eksponensial karena throttle ringan untuk DynamoDB untuk kembali dan tidak dikenakan biaya per permintaan kepada Anda. Backoff eksponensial memberikan penundaan yang lebih lama ke utas klien yang telah menunggu paling lama, yang secara statistik memperluas p50 dan p99 ke luar.
- Kesalahan (ditunjukkan oleh `InternalServerError` atau `ServiceUnavailable`, antara lain) menunjukkan masalah sementara dengan layanan. Ini bisa untuk seluruh tabel atau mungkin hanya partisi yang Anda baca atau tulis. Dengan kesalahan, Anda dapat berhenti lebih lama sebelum mencoba ulang (seperti 250ms atau 500ms) dan menggunakan jitter untuk membuat percobaan ulang terhuyung-huyung.

Config untuk koneksi kolam maksimal

Terakhir, konfigurasi memungkinkan Anda mengontrol ukuran kumpulan koneksi:

- `max_pool_connections` (int) - Jumlah maksimum koneksi untuk disimpan dalam kumpulan koneksi. Jika nilai ini tidak diatur, nilai default 10 digunakan.

Opsi ini mengontrol jumlah maksimum koneksi HTTP agar tetap dikumpulkan untuk digunakan kembali. Kolam yang berbeda disimpan per Sesi. Jika Anda mengantisipasi lebih dari 10 utas yang bertentangan dengan klien atau sumber daya yang dibangun dari Sesi yang sama, Anda harus mempertimbangkan untuk meningkatkan ini, jadi utas tidak harus menunggu utas lain menggunakan koneksi gabungan.

```
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
```

Penanganan kesalahan

AWS pengecualian layanan tidak semuanya didefinisikan secara statis di Boto3. Ini karena kesalahan dan pengecualian dari AWS layanan sangat bervariasi dan dapat berubah. Boto3 membungkus semua pengecualian layanan sebagai a `ClientError` dan mengekspos detail sebagai JSON terstruktur. Misalnya, respons kesalahan mungkin terstruktur seperti ini:

```
{
  '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
    }
}
```

Kode berikut menangkap `ClientError` pengecualian dan melihat nilai string di Code dalam Error untuk menentukan tindakan apa yang harus diambil:

```
import boto3
import boto3

dynamodb = boto3.client('dynamodb')

try:
    response = dynamodb.put_item(...)

except boto3.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
```

Beberapa (tetapi tidak semua) kode pengecualian telah diwujudkan sebagai kelas tingkat atas. Anda dapat memilih untuk menangani ini secara langsung. Saat menggunakan antarmuka Klien, pengecualian ini diisi secara dinamis di klien Anda dan Anda menangkap pengecualian ini menggunakan instance klien Anda, seperti ini:

```
except ddb_client.exceptions.ProvisionedThroughputExceededException:
```

Saat menggunakan antarmuka Resource, Anda harus menggunakan `.meta.client` untuk melintasi dari sumber daya ke Klien yang mendasarinya untuk mengakses pengecualian, seperti ini:

```
except ddb_resource.meta.client.exceptions.ProvisionedThroughputExceededException:
```

Untuk meninjau daftar jenis pengecualian terwujud, Anda dapat membuat daftar secara dinamis:

```
ddb = boto3.client("dynamodb")
print([e for e in dir(ddb.exceptions) if e.endswith('Exception') or
      e.endswith('Error')])
```

Saat melakukan operasi tulis dengan ekspresi kondisi, Anda dapat meminta bahwa jika ekspresi gagal, nilai item harus dikembalikan dalam respons kesalahan.

```
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'])
```

Untuk bacaan lebih lanjut tentang penanganan kesalahan dan pengecualian:

- [Panduan boto3 tentang penanganan kesalahan](#) memiliki informasi lebih lanjut tentang teknik penanganan kesalahan.
- [Bagian panduan pengembang DynamoDB tentang kesalahan pemrograman](#) mencantumkan kesalahan apa yang mungkin Anda temui.
- [Bagian Kesalahan Umum dalam referensi API](#).
- Dokumentasi pada setiap operasi API mencantumkan kesalahan apa yang mungkin dihasilkan oleh panggilan (misalnya [BatchWriteItem](#)).

Pencatatan log

Pustaka boto3 terintegrasi dengan modul logging bawaan Python untuk melacak apa yang terjadi selama sesi. Untuk mengontrol level logging, Anda dapat mengonfigurasi modul logging:


```
import logging

logging.basicConfig(level=logging.INFO)
```

Ini mengonfigurasi root logger untuk mencatat INFO dan pesan tingkat di atas. Pesan logging yang kurang parah dari level akan diabaikan. Tingkat logging termasuk DEBUG, INFO, WARNING, ERROR, dan CRITICAL. Nilai default-nya WARNING.

Logger di boto3 bersifat hierarkis. Pustaka menggunakan beberapa logger yang berbeda, masing-masing sesuai dengan bagian perpustakaan yang berbeda. Anda dapat mengontrol perilaku masing-masing secara terpisah:

- boto3: Logger utama untuk modul boto3.
- botocore: Logger utama untuk paket botocore.
- botocore.auth: Digunakan untuk mencatat pembuatan tanda tangan untuk permintaan. AWS
- botocore.credentials: Digunakan untuk mencatat proses pengambilan dan penyegaran kredensial.
- botocore.endpoint: Digunakan untuk pembuatan permintaan logging sebelum dikirim melalui jaringan.
- botocore.hooks: Digunakan untuk peristiwa logging yang dipicu di perpustakaan.
- botocore.loaders: Digunakan untuk logging ketika bagian-bagian dari model layanan dimuat. AWS
- botocore.parsers: Digunakan untuk mencatat respons AWS layanan sebelum diurai.
- botocore.retryhandler: Digunakan untuk mencatat pemrosesan percobaan ulang permintaan layanan (mode lama). AWS
- botocore.retries.standard: Digunakan untuk mencatat pemrosesan percobaan ulang permintaan AWS layanan (mode standar atau adaptif).
- botocore.utils: Digunakan untuk mencatat aktivitas lain-lain di perpustakaan.
- botocore.waiter: Digunakan untuk mencatat fungsionalitas pelayan, yang melakukan polling layanan sampai keadaan tertentu AWS tercapai.

Perpustakaan lain juga mencatat. Secara internal, boto3 menggunakan urllib3 pihak ketiga untuk penanganan koneksi HTTP. Ketika latensi penting, Anda dapat menonton lognya untuk memastikan kolam Anda dimanfaatkan dengan baik dengan melihat kapan urllib3 membuat koneksi baru atau menutup yang tidak aktif.

- urllib3.connectionpool: Digunakan untuk mencatat peristiwa penanganan kumpulan koneksi.

Cuplikan kode berikut menyetel sebagian besar logging INFO dengan DEBUG logging untuk aktivitas titik akhir dan kumpulan koneksi:

```
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)
```

Kait acara

Botocore memancarkan peristiwa selama berbagai bagian pelaksanaannya. Anda dapat mendaftarkan penanganan untuk acara ini sehingga setiap kali suatu peristiwa dipancarkan, handler Anda akan dipanggil. Ini memungkinkan Anda memperluas perilaku botocore tanpa harus memodifikasi internal.

Misalnya, katakanlah Anda ingin melacak setiap kali PutItem operasi dipanggil pada setiap tabel DynamoDB dalam aplikasi Anda. Anda dapat mendaftar pada 'provide-client-params.dynamodb.PutItem' acara untuk menangkap dan mencatat setiap kali PutItem operasi dipanggil pada Sesi terkait. Inilah contohnya:

```
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 ...  
    }  
)
```

Di dalam handler, Anda bahkan dapat memanipulasi params secara terprogram untuk mengubah perilaku:

```
params['TableName'] = "NewTableName"
```

Untuk informasi lebih lanjut tentang acara, lihat dokumentasi [botocore tentang acara dan dokumentasi boto3](#) tentang acara.

Pagination dan Paginator

Beberapa permintaan, seperti Query dan Scan, membatasi ukuran data yang dikembalikan pada satu permintaan dan mengharuskan Anda untuk membuat permintaan berulang untuk menarik halaman berikutnya.

Anda dapat mengontrol jumlah maksimum item yang akan dibaca untuk setiap halaman dengan `limit` parameter. Misalnya, jika Anda menginginkan 10 item terakhir, Anda dapat menggunakan `limit` untuk mengambil hanya 10 item terakhir. Perhatikan batasnya adalah berapa banyak yang harus dibaca dari tabel sebelum penyaringan apa pun diterapkan. Tidak ada cara untuk menentukan yang Anda inginkan tepat 10 setelah pemfilteran; Anda hanya dapat mengontrol jumlah yang telah difilter sebelumnya dan memeriksa sisi klien ketika Anda benar-benar mengambil 10. Terlepas dari batasnya, setiap respons selalu memiliki ukuran maksimum 1 MB.

Jika respons menyertakan `aLastEvaluatedKey`, ini menunjukkan respons berakhir karena mencapai batas hitungan atau ukuran. Kuncinya adalah kunci terakhir yang dievaluasi untuk respons. Anda dapat mengambil ini `LastEvaluatedKey` dan meneruskannya ke panggilan tindak lanjut `ExclusiveStartKey` untuk membaca potongan berikutnya dari titik awal itu. Ketika tidak ada yang `LastEvaluatedKey` dikembalikan, berarti tidak ada lagi item yang cocok dengan Kueri atau Pindai.

Berikut adalah contoh sederhana (menggunakan antarmuka Sumber Daya, tetapi antarmuka Klien memiliki pola yang sama) yang membaca paling banyak 100 item per halaman dan loop hingga semua item telah dibaca.

```
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']
```

Untuk kenyamanan, boto3 dapat melakukan ini untuk Anda dengan Paginator. Namun, ini hanya berfungsi dengan antarmuka Klien. Berikut kode yang ditulis ulang untuk menggunakan Paginator:

```
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)
```

Untuk informasi selengkapnya, lihat [Panduan tentang Paginator dan referensi API untuk DynamoDb.paginator.Query](#).

Note

Paginator juga memiliki pengaturan konfigurasi mereka sendiri bernama `MaxItems`, `StartingToken`, dan `PageSize` Untuk paginating dengan DynamoDB, Anda harus mengabaikan pengaturan ini.

Pelayan

Pelayan memberikan kemampuan untuk menunggu sesuatu selesai sebelum melanjutkan. Saat ini, mereka hanya mendukung menunggu tabel dibuat atau dihapus. Di latar belakang, operasi pelayan melakukan pemeriksaan untuk Anda setiap 20 detik hingga 25 kali. Anda bisa melakukannya sendiri, tetapi menggunakan pelayan itu elegan saat menulis otomatisasi.

Kode ini menunjukkan cara menunggu tabel tertentu dibuat:

```
# 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'])
```

Untuk informasi lebih lanjut, lihat [Panduan untuk Pelayan dan Referensi tentang Pelayan](#).

Pemrograman Amazon DynamoDB dengan JavaScript

Panduan ini memberikan orientasi kepada programmer yang ingin menggunakan Amazon DynamoDB dengan JavaScript. Pelajari tentang AWS SDK for JavaScript, lapisan abstraksi yang

tersedia, mengonfigurasi koneksi, menangani kesalahan, menentukan kebijakan coba lagi, mengelola keep-alive, dan banyak lagi.

Topik

- [Tentang AWS SDK for JavaScript](#)
- [Menggunakan AWS SDK for JavaScript V3](#)
- [Mengakses dokumentasi JavaScript](#)
- [Lapisan abstraksi](#)
- [Menggunakan fungsi utilitas marshall](#)
- [Membaca item](#)
- [Penulisan bersyarat](#)
- [Paginasi](#)
- [Menentukan konfigurasi](#)
- [Pelayan](#)
- [Penanganan kesalahan](#)
- [Pencatatan log](#)
- [Pertimbangan](#)

Tentang AWS SDK for JavaScript

AWS SDK for JavaScript ini menyediakan akses untuk Layanan AWS menggunakan skrip browser atau Node.js. Dokumentasi ini berfokus pada versi terbaru SDK (V3). AWS SDK for JavaScript V3 dikelola oleh AWS sebagai [proyek sumber terbuka yang dihosting di](#). GitHub Masalah dan permintaan fitur bersifat publik dan Anda dapat mengaksesnya di halaman masalah untuk GitHub repositori.

JavaScript V2 mirip dengan V3, tetapi berisi perbedaan sintaks. V3 lebih modular, membuatnya lebih mudah untuk mengirimkan dependensi yang lebih kecil, dan memiliki dukungan kelas satu. TypeScript Sebaiknya gunakan SDK versi terbaru.

Menggunakan AWS SDK for JavaScript V3

Anda dapat menambahkan SDK ke aplikasi Node.js Anda menggunakan Node Package Manager. Contoh di bawah ini menunjukkan cara menambahkan paket SDK paling umum untuk bekerja dengan DynamoDB.

- `npm install @aws-sdk/client-dynamodb`
- `npm install @aws-sdk/lib-dynamodb`
- `npm install @aws-sdk/util-dynamodb`

Menginstal paket menambahkan referensi ke bagian ketergantungan file proyek `package.json` Anda. Anda memiliki opsi untuk menggunakan sintaks modul ECMAScript yang lebih baru. Untuk detail lebih lanjut tentang dua pendekatan ini, lihat bagian [Pertimbangan](#).

Mengakses dokumentasi JavaScript

Mulailah dengan JavaScript dokumentasi dengan sumber daya berikut:

- Akses [panduan Pengembang](#) untuk JavaScript dokumentasi inti. Petunjuk instalasi terletak di bagian [Pengaturan](#).
- Akses dokumentasi [referensi API](#) untuk menjelajahi semua kelas dan metode yang tersedia.
- SDK untuk JavaScript mendukung banyak Layanan AWS selain DynamoDB. Gunakan prosedur berikut untuk menemukan cakupan API tertentu untuk DynamoDB:
 1. Dari [Services](#), pilih [DynamoDB](#) dan [Libraries](#). Ini mendokumentasikan klien tingkat rendah.
 2. Pilih `lib-dynamodb`. Ini mendokumentasikan klien tingkat tinggi. Kedua klien mewakili dua lapisan abstraksi berbeda yang Anda memiliki pilihan untuk digunakan. Lihat bagian di bawah ini untuk informasi lebih lanjut tentang lapisan abstraksi.

Lapisan abstraksi

SDK untuk JavaScript V3 memiliki klien tingkat rendah (`DynamoDBClient`) dan klien tingkat tinggi (`DynamoDBDocumentClient`).

Topik

- [Klien tingkat rendah \(\) `DynamoDBClient`](#)
- [Klien tingkat tinggi \(\) `DynamoDBDocumentClient`](#)

Klien tingkat rendah () **`DynamoDBClient`**

Klien tingkat rendah tidak memberikan abstraksi tambahan atas protokol kawat yang mendasarinya. Ini memberi Anda kontrol penuh atas semua aspek komunikasi, tetapi karena tidak ada abstraksi,

Anda harus melakukan hal-hal seperti memberikan definisi item menggunakan format DynamoDB JSON.

Seperti contoh di bawah ini menunjukkan, dengan format ini tipe data harus dinyatakan secara eksplisit. S menunjukkan nilai string dan N menunjukkan nilai angka. Angka pada kawat selalu dikirim sebagai string yang ditandai sebagai jenis angka untuk memastikan tidak ada kehilangan presisi. Panggilan API tingkat rendah memiliki pola penamaan seperti `PutItemCommand` dan `GetItemCommand`.

Contoh berikut adalah menggunakan klien tingkat rendah dengan Item didefinisikan menggunakan DynamoDB JSON:

```
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();
```

Klien tingkat tinggi () **DynamoDBDocumentClient**

Klien dokumen DynamoDB tingkat tinggi menawarkan fitur kenyamanan bawaan, seperti menghilangkan kebutuhan untuk mengumpulkan data secara manual dan memungkinkan untuk

membaca dan menulis langsung menggunakan objek standar. JavaScript [Dokumentasi untuk lib-dynamodb](#) menyediakan daftar keuntungan.

Untuk membuat instance `DynamoDBDocumentClient`, buat level rendah `DynamoDBClient` dan kemudian bungkus dengan `a`. `DynamoDBDocumentClient` Konvensi penamaan fungsi sedikit berbeda antara kedua paket. Misalnya, penggunaan tingkat rendah `PutItemCommand` sementara penggunaan tingkat tinggi. `PutCommand` Nama yang berbeda memungkinkan kedua set fungsi untuk hidup berdampingan dalam konteks yang sama, memungkinkan Anda untuk mencampur keduanya dalam skrip yang sama.

```
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();
```

Pola penggunaan konsisten saat Anda membaca item menggunakan operasi API seperti `GetItem`, `Query`, atau `Scan`.

Menggunakan fungsi utilitas marshall

Anda dapat menggunakan klien tingkat rendah dan marshall atau unmarshall tipe data Anda sendiri. Paket utilitas, [util-dynamodb](#), memiliki fungsi `marshall()` utilitas yang menerima JSON dan menghasilkan DynamoDB JSON, serta fungsi, yang melakukan sebaliknya. `unmarshall()` Contoh berikut menggunakan klien tingkat rendah dengan data marshalling ditangani oleh panggilan. `marshall()`

```
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();
```

Membaca item

Untuk membaca satu item dari DynamoDB, Anda menggunakan `GetItem` operasi API. Mirip dengan `PutItem` perintah, Anda memiliki pilihan untuk menggunakan klien tingkat rendah atau klien Dokumen tingkat tinggi. Contoh di bawah ini menunjukkan menggunakan klien Dokumen tingkat tinggi untuk mengambil item.

```
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();
```

Gunakan operasi Query API untuk membaca beberapa item. Anda dapat menggunakan klien tingkat rendah atau klien Dokumen. Contoh di bawah ini menggunakan klien Dokumen tingkat tinggi.

```
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();
```

Penulisan bersyarat

Operasi penulisan DynamoDB dapat menentukan ekspresi kondisi logis yang harus mengevaluasi ke true agar penulisan dapat dilanjutkan. Jika kondisi tidak mengevaluasi ke true, operasi tulis menghasilkan pengecualian. Ekspresi kondisi dapat memeriksa apakah item sudah ada atau apakah atributnya cocok dengan batasan tertentu.

```
ConditionExpression = "version = :ver AND size(VideoClip) < :maxsize"
```

Ketika ekspresi kondisional gagal, Anda dapat menggunakan `ReturnValuesOnConditionCheckFailure` untuk meminta agar respons kesalahan menyertakan item yang tidak memenuhi kondisi untuk menyimpulkan apa masalahnya. Untuk detail selengkapnya, lihat [Menangani kesalahan penulisan bersyarat dalam skenario konkurensi tinggi dengan Amazon DynamoDB](#).

```
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;
    }
}
```

[Contoh kode tambahan yang menunjukkan aspek lain dari penggunaan JavsScript SDK V3 tersedia di Dokumentasi SDK V3 dan di bawah repositori JavaScript DynamoDB-SDK-Examples. GitHub](#)

Paginasi

Topik

- [Menggunakan metode `paginateScan` kenyamanan](#)

Permintaan baca seperti `Scan` atau kemungkinan `Query` akan mengembalikan beberapa item dalam kumpulan data. Jika Anda melakukan `Scan` atau `Query` dengan `Limit` parameter, maka setelah sistem membaca banyak item, sebagian respons akan dikirim, dan Anda harus membuat paginasi untuk mengambil item tambahan.

Sistem hanya akan membaca maksimal 1 megabyte data per permintaan. Jika Anda menyertakan `Filter` ekspresi, sistem akan tetap membaca megabyte, maksimum, data dari disk, tetapi akan mengembalikan item megabyte yang cocok dengan filter. Operasi filter dapat mengembalikan 0 item untuk halaman, tetapi masih memerlukan pagination lebih lanjut sebelum pencarian habis.

Anda harus mencari `LastEvaluatedKey` dalam respons dan menggunakannya sebagai `ExclusiveStartKey` parameter dalam permintaan berikutnya untuk melanjutkan pengambilan data. Ini berfungsi sebagai bookmark seperti yang tercantum dalam contoh berikut.

Note

Sampel melewati nol `lastEvaluatedKey` sebagai `ExclusiveStartKey` pada iterasi pertama dan ini diperbolehkan.

Contoh menggunakan `LastEvaluatedKey`:

```
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);
});
```

Menggunakan metode **paginateScan** kenyamanan

SDK menyediakan metode kenyamanan yang dipanggil `paginateScan` dan `paginateQuery` yang melakukan ini bekerja untuk Anda dan membuat permintaan berulang di belakang layar. Tentukan jumlah maksimum item yang akan dibaca per permintaan menggunakan `Limit` parameter standar.

```
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

Melakukan pemindaian tabel penuh secara teratur bukanlah pola akses yang disarankan kecuali tabelnya kecil.

Menentukan konfigurasi

Topik

- [Config untuk batas waktu](#)
- [Config untuk keep-alive](#)
- [Config untuk percobaan ulang](#)

Saat mengatur `DynamoDBClient`, Anda dapat menentukan berbagai penggantian konfigurasi dengan meneruskan objek konfigurasi ke konstruktor. Misalnya, Anda dapat menentukan Wilayah yang akan disambungkan jika belum diketahui konteks panggilan atau URL titik akhir yang akan digunakan. Ini berguna jika Anda ingin menargetkan instance DynamoDB Local untuk tujuan pengembangan.

```
const client = new DynamoDBClient({
  region: "eu-west-1",
  endpoint: "http://localhost:8000",
});
```

Config untuk batas waktu

DynamoDB menggunakan HTTPS untuk komunikasi client-server. Anda dapat mengontrol beberapa aspek dari lapisan HTTP dengan menyediakan `NodeHttpHandler` objek. Misalnya, Anda dapat menyesuaikan nilai batas waktu kunci `connectionTimeout` dan `requestTimeout`. `connectionTimeout` ini adalah durasi maksimum, dalam milidetik, bahwa klien akan menunggu sambil mencoba membuat koneksi sebelum menyerah.

Ini `requestTimeout` menentukan berapa lama klien akan menunggu respons setelah permintaan dikirim, juga dalam milidetik. Default untuk keduanya adalah nol, artinya batas waktu dinonaktifkan dan tidak ada batasan berapa lama klien akan menunggu jika respons tidak tiba. Anda harus mengatur batas waktu ke sesuatu yang masuk akal sehingga jika terjadi masalah jaringan, permintaan akan error dan permintaan baru dapat dimulai. Sebagai contoh:

```
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

Contoh yang diberikan menggunakan impor [Smithy](#). Smithy adalah bahasa untuk mendefinisikan layanan dan SDK, sumber terbuka dan dikelola oleh AWS.

Selain mengonfigurasi nilai batas waktu, Anda dapat mengatur jumlah soket maksimum, yang memungkinkan peningkatan jumlah koneksi bersamaan per asal. Panduan pengembang mencakup [detail tentang mengonfigurasi `maxSockets` parameter](#).

Config untuk keep-alive

Saat menggunakan HTTPS, permintaan pertama selalu membutuhkan back-and-forth komunikasi untuk membuat koneksi yang aman. HTTP Keep-Alive memungkinkan permintaan berikutnya

untuk menggunakan kembali koneksi yang sudah dibuat, membuat permintaan lebih efisien dan menurunkan latensi. HTTP Keep-Alive diaktifkan secara default dengan V3. JavaScript

Ada batasan berapa lama koneksi idle dapat tetap hidup. Pertimbangkan untuk mengirim permintaan berkala, mungkin setiap menit, jika Anda memiliki koneksi idle tetapi ingin permintaan berikutnya menggunakan koneksi yang sudah dibuat.

Note

Perhatikan bahwa di V2 SDK yang lebih lama, keep-alive dimatikan secara default, yang berarti setiap koneksi akan segera ditutup setelah digunakan. Jika menggunakan V2, Anda dapat mengganti pengaturan ini.

Config untuk percobaan ulang

Ketika SDK menerima respons kesalahan dan kesalahan dapat dilanjutkan seperti yang ditentukan oleh SDK, seperti pengecualian pelambatan atau pengecualian layanan sementara, SDK akan mencoba lagi. Ini terjadi tanpa terlihat bagi Anda sebagai penelepon, kecuali bahwa Anda mungkin memperhatikan permintaan membutuhkan waktu lebih lama untuk berhasil.

SDK untuk JavaScript V3 akan membuat 3 permintaan total, secara default, sebelum menyerah dan meneruskan kesalahan ke dalam konteks panggilan. Anda dapat menyesuaikan jumlah dan frekuensi percobaan ulang ini.

`DynamoDBClient` Konstruktors menerima `maxAttempts` pengaturan yang membatasi berapa banyak upaya yang akan terjadi. Contoh di bawah ini menaikkan nilai dari default 3 menjadi total 5. Jika Anda menyetelnya ke 0 atau 1, itu menunjukkan Anda tidak ingin mencoba ulang otomatis dan ingin menangani kesalahan yang dapat dilanjutkan sendiri dalam blok catch Anda.

```
const client = new DynamoDBClient({
  maxAttempts: 5,
});
```

Anda juga dapat mengontrol waktu percobaan ulang dengan strategi coba lagi khusus. Untuk melakukan ini, impor paket `util-retry` utilitas dan buat fungsi backoff khusus yang menghitung waktu tunggu antara percobaan ulang berdasarkan jumlah coba lagi saat ini.

Contoh di bawah ini mengatakan untuk melakukan maksimal 5 upaya dengan penundaan 15, 30, 90, dan 360 milidetik jika upaya pertama gagal. Fungsi backoff kustom, `calculateRetryBackoff`,

menghitung penundaan dengan menerima nomor percobaan ulang (dimulai dengan 1 untuk percobaan ulang pertama) dan mengembalikan berapa milidetik untuk menunggu permintaan itu.

```
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.
  ),
});
```

Pelayan

Klien DynamoDB mencakup dua fungsi [pelayan berguna](#) yang dapat digunakan saat membuat, memodifikasi, atau menghapus tabel saat Anda ingin kode Anda menunggu untuk melanjutkan hingga modifikasi tabel selesai. Misalnya, Anda dapat menyebarkan tabel, memanggil `waitUntilTableExists` fungsi, dan kode akan memblokir sampai tabel telah dibuat AKTIF. Pelayan secara internal melakukan polling layanan DynamoDB dengan setiap 20 detik. `describe-table`

```
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` fitur mengembalikan kontrol hanya ketika dapat melakukan `describe-table` perintah yang menunjukkan status tabel ACTIVE. Ini memastikan bahwa Anda dapat menggunakan `waitUntilTableExists` untuk menunggu penyelesaian pembuatan, serta

modifikasi seperti menambahkan indeks GSI, yang mungkin memerlukan beberapa waktu untuk diterapkan sebelum tabel kembali ke status AKTIF.

Penanganan kesalahan

Dalam contoh awal di sini, kami telah menangkap semua kesalahan secara luas. Namun, dalam aplikasi praktis, penting untuk membedakan antara berbagai jenis kesalahan dan menerapkan penanganan kesalahan yang lebih tepat.

Respons kesalahan DynamoDB berisi metadata, termasuk nama kesalahan. Anda dapat menangkap kesalahan kemudian mencocokkan dengan kemungkinan nama string dari kondisi kesalahan untuk menentukan cara melanjutkan. Untuk kesalahan sisi server, Anda dapat memanfaatkan `instanceof` operator dengan jenis kesalahan yang diekspor oleh `@aws-sdk/client-dynamodb` paket untuk mengelola penanganan kesalahan secara efisien.

Penting untuk dicatat bahwa kesalahan ini hanya terwujud setelah semua percobaan ulang telah habis. Jika kesalahan dicoba lagi dan akhirnya diikuti oleh panggilan yang berhasil, dari perspektif kode, tidak ada kesalahan hanya latensi yang sedikit meningkat. Percobaan ulang akan muncul di CloudWatch bagan Amazon sebagai permintaan yang tidak berhasil, seperti permintaan throttle atau error. Jika klien mencapai jumlah percobaan ulang maksimum, itu akan menyerah dan menghasilkan pengecualian. Ini adalah cara klien mengatakan itu tidak akan mencoba lagi.

Di bawah ini adalah cuplikan untuk menangkap kesalahan dan mengambil tindakan berdasarkan jenis kesalahan yang dikembalikan.

```
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.
}
}
```

Lihat [the section called “Penanganan kesalahan”](#) string kesalahan umum di DynamoDB Developer Guide. Kesalahan persis yang mungkin terjadi dengan panggilan API tertentu dapat ditemukan dalam dokumentasi untuk panggilan API tersebut, seperti [dokumen Query API](#).

Metadata kesalahan mencakup properti tambahan, tergantung pada kesalahannya. Untuk a `TimeoutError`, metadata mencakup jumlah upaya yang dilakukan dan `totalRetryDelay`, seperti yang ditunjukkan di bawah ini.

```
{
  "name": "TimeoutError",
  "$metadata": {
    "attempts": 3,
    "totalRetryDelay": 199
  }
}
```

Jika Anda mengelola kebijakan coba ulang Anda sendiri, Anda akan ingin membedakan antara throttle dan error:

- Throttle (ditunjukkan oleh `ProvisionedThroughputExceededException` atau `ThrottlingException`) menunjukkan layanan sehat yang memberi tahu Anda bahwa Anda telah melebihi kapasitas baca atau tulis pada tabel atau partisi DynamoDB. Setiap milidetik yang berlalu, sedikit lebih banyak kapasitas baca atau tulis tersedia, sehingga Anda dapat mencoba lagi dengan cepat, seperti setiap 50 ms, untuk mencoba mengakses kapasitas yang baru dirilis.

Dengan throttle Anda tidak terlalu membutuhkan backoff eksponensial karena throttle ringan untuk DynamoDB untuk kembali dan tidak dikenakan biaya per permintaan kepada Anda. Backoff eksponensial memberikan penundaan yang lebih lama ke utas klien yang telah menunggu paling lama, yang secara statistik memperluas p50 dan p99 ke luar.

- Kesalahan (ditunjukkan oleh `InternalServerError` atau `ServiceUnavailable`, antara lain) menunjukkan masalah sementara dengan layanan, mungkin seluruh tabel atau hanya partisi

yang Anda baca atau tulis. Dengan kesalahan, Anda dapat berhenti lebih lama sebelum mencoba ulang, seperti 250ms atau 500ms, dan menggunakan jitter untuk membuat percobaan ulang terhuyung-huyung.

Pencatatan log

Aktifkan logging untuk mendapatkan detail lebih lanjut tentang apa yang dilakukan SDK. Anda dapat mengatur parameter pada `DynamoDBClient` seperti yang ditunjukkan pada contoh di bawah ini. Informasi log lainnya akan muncul di konsol dan menyertakan metadata seperti kode status dan kapasitas yang dikonsumsi. Jika Anda menjalankan kode secara lokal di jendela terminal, log muncul di sana. Jika Anda menjalankan kode di AWS Lambda, dan Anda memiliki CloudWatch log Amazon yang disiapkan, maka output konsol akan ditulis di sana.

```
const client = new DynamoDBClient({
  logger: console
});
```

Anda juga dapat menghubungkan ke aktivitas SDK internal dan melakukan pencatatan khusus saat peristiwa tertentu terjadi. Contoh di bawah ini menggunakan klien `middlewareStack` untuk mencegat setiap permintaan saat dikirim dari SDK dan mencatatnya saat terjadi.

```
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` ini menyediakan pengait yang kuat untuk mengamati dan mengendalikan perilaku SDK. Lihat blog [Memperkenalkan Middleware Stack in Modular AWS SDK for JavaScript](#), untuk informasi lebih lanjut.

Pertimbangan

Saat menerapkan AWS SDK for JavaScript dalam proyek Anda, berikut adalah beberapa faktor lebih lanjut yang perlu dipertimbangkan.

Sistem modul

SDK mendukung dua sistem modul, CommonJS dan ES (ECMAScript). CommonJS menggunakan `require` fungsi, sedangkan ES menggunakan `import` kata kunci.

1. JS umum - `const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");`
2. ES (ECMAScript) — `import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";`

Jenis proyek menentukan sistem modul yang akan digunakan dan ditentukan di bagian tipe file `package.json` Anda. Defaultnya adalah CommonJS. Gunakan `"type": "module"` untuk menunjukkan proyek ES. Jika Anda memiliki proyek Node.JS yang sudah ada yang menggunakan format paket CommonJS, Anda masih dapat menambahkan fungsi dengan sintaks Impor SDK V3 yang lebih modern dengan menamai file fungsi Anda dengan ekstensi `.mjs`. Ini akan memungkinkan file kode diperlakukan sebagai ES (ECMAScript).

Operasi asinkron

Anda akan melihat banyak contoh kode menggunakan callback dan janji untuk menangani hasil operasi DynamoDB. Dengan modern, kompleksitas JavaScript ini tidak lagi diperlukan dan pengembang dapat memanfaatkan sintaks `async/await` yang lebih ringkas dan mudah dibaca untuk operasi asinkron.

Waktu proses peramban web

Pengembang web dan seluler yang membangun dengan React atau React Native dapat menggunakan SDK untuk JavaScript proyek mereka. Dengan V2 SDK sebelumnya, pengembang web harus memuat SDK lengkap ke browser, merujuk gambar SDK yang dihosting di <https://sdk.amazonaws.com/js/>.

Dengan V3, dimungkinkan untuk menggabungkan hanya modul klien V3 yang diperlukan dan semua JavaScript fungsi yang diperlukan ke dalam satu JavaScript file menggunakan Webpack, dan menambahkannya dalam tag skrip di halaman HTML Anda, seperti yang dijelaskan di bagian [Memulai di skrip browser](#) dari dokumentasi SDK. `<head>`

Operasi pesawat data DAX

SDK untuk JavaScript V3 saat ini tidak memberikan dukungan untuk operasi pesawat data Amazon DynamoDB Streams Accelerator (DAX). Jika Anda meminta dukungan DAX, pertimbangkan untuk menggunakan SDK untuk JavaScript V2 yang mendukung operasi bidang data DAX.

Pemrograman Amazon DynamoDB dengan AWS SDK for Java 2.x

Panduan ini memberikan orientasi kepada programmer yang ingin menggunakan Amazon DynamoDB dengan Java. Panduan ini mencakup konsep yang berbeda, seperti lapisan abstraksi, manajemen konfigurasi, penanganan kesalahan, pengendalian kebijakan coba lagi, dan pengelolaan keep-alive.

Topik

- [Tentang AWS SDK for Java 2.x](#)
- [Memulai dengan AWS SDK for Java 2.x](#)
- [Menggunakan AWS SDK for Java 2.x dokumentasi](#)
- [Antarmuka yang didukung](#)
- [Contoh kode tambahan](#)
- [Pemrograman sinkron dan asinkron](#)
- [Klien HTTP](#)
- [Mengkonfigurasi klien HTTP](#)
- [Penanganan kesalahan](#)
- [AWS ID permintaan](#)
- [Pencatatan log](#)
- [Paginasi](#)
- [Anotasi kelas data](#)

Tentang AWS SDK for Java 2.x

Anda dapat mengakses DynamoDB dari Java menggunakan resmi. AWS SDK for Java SDK for Java memiliki dua versi: 1.x dan 2.x. Untuk 1.x end-of-support [diumumkan](#) pada 12 Januari 2024.

Ini memasuki mode pemeliharaan pada 31 Juli 2024 dan jatuh tempo pada 31 Desember 2025. end-of-support Untuk pengembangan baru, kami sangat menyarankan Anda menggunakan 2.x, yang pertama kali dirilis pada tahun 2018. Panduan ini secara eksklusif menargetkan 2.x dan hanya berfokus pada bagian SDK yang relevan dengan DynamoDB.

Anda dapat menemukan informasi selengkapnya tentang pemeliharaan dan dukungan untuk SDK dalam [kebijakan pemeliharaan AWS SDK dan Alat dan](#) topik [AWS matriks dukungan versi SDK dan Alat](#) dari AWS SDK dan Panduan Referensi Alat.

AWS SDK for Java 2.x Ini adalah penulisan ulang utama dari basis kode 1.x untuk mendukung fitur Java modern, seperti I/O non-blocking yang diperkenalkan di Java 8. SDK for Java 2.x juga menambahkan dukungan untuk implementasi klien HTTP pluggable untuk memberikan lebih banyak fleksibilitas dan konfigurasi untuk koneksi jaringan.

Perubahan nyata antara SDK for Java 1.x dan SDK for Java 2.x adalah penggunaan nama paket baru. Java 1.x SDK menggunakan nama `com.amazonaws` paket, sedangkan Java 2.x SDK menggunakan `software.amazon.awssdk` Demikian pula, artefak Maven untuk Java 1.x SDK menggunakan `com.amazonaws` GroupId, sedangkan artefak SDK Java 2.x menggunakan GroupId `software.amazon.awssdk`

Important

AWS SDK for Java 1.x memiliki paket DynamoDB bernama `com.amazonaws.dynamodbv2` V2 dalam nama paket tidak menunjukkan itu Java v2. v2 menunjukkan bahwa paket tersebut mendukung [versi kedua](#) API tingkat rendah DynamoDB alih-alih versi [asli](#) API tingkat rendah.

Support untuk versi Java

AWS SDK for Java 2.x Ini memberikan dukungan penuh untuk [rilis Java](#) dukungan jangka panjang (LTS).

Memulai dengan AWS SDK for Java 2.x

Tutorial berikut menunjukkan cara menggunakan [Apache Maven](#) untuk mendefinisikan dependensi untuk SDK for Java 2.x. Tutorial ini juga menunjukkan cara menulis kode yang menghubungkan ke DynamoDB untuk daftar tabel DynamoDB yang tersedia. Tutorial ini didasarkan pada [Memulai dengan AWS SDK for Java 2.x](#). Kami telah mengedit tutorial ini untuk melakukan panggilan ke DynamoDB alih-alih Amazon S3.

Lakukan langkah-langkah berikut untuk menyelesaikan tutorial ini:

- [Langkah 1: Siapkan tutorial ini](#)
- [Langkah 2: Buat proyek](#)
- [Langkah 3: Tulis kodenya](#)
- [Langkah 4: Bangun dan jalankan aplikasi](#)

Langkah 1: Siapkan tutorial ini

Sebelum Anda memulai tutorial ini, Anda memerlukan yang berikut:

- Izin untuk mengakses Amazon DynamoDB.
- Lingkungan pengembangan Java yang dikonfigurasi untuk mengakses Layanan AWS menggunakan sistem masuk tunggal ke AWS IAM Identity Center

Gunakan petunjuk dalam [ikhtisar Pengaturan](#) untuk menyiapkan tutorial ini. Setelah Anda [mengkonfigurasi lingkungan pengembangan Anda dengan akses masuk tunggal](#) untuk Java SDK dan Anda memiliki [sesi portal AWS akses aktif](#), lanjutkan dengan [Langkah 2](#) dari tutorial ini.

Langkah 2: Buat proyek

Untuk membuat proyek untuk tutorial ini, Anda menjalankan perintah Maven yang meminta Anda untuk masukan tentang cara mengkonfigurasi proyek. Setelah semua input dimasukkan dan dikonfirmasi, Maven selesai membangun proyek dengan membuat `pom.xml` file dan membuat file Java rintisan.

1. Buka terminal atau jendela prompt perintah dan arahkan ke direktori pilihan Anda, misalnya, Home folder Anda Desktop atau.
2. Masukkan perintah berikut di terminal dan tekan `Enter`.

```
mvn archetype:generate \  
  -DarchetypeGroupId=software.amazon.awssdk \  
  -DarchetypeArtifactId=archetype-app-quickstart \  
  -DarchetypeVersion=2.22.0
```

3. Masukkan nilai yang tercantum di kolom kedua untuk setiap prompt.

Prompt	Nilai untuk masuk
Define value for property 'service':	dynamodb
Define value for property 'httpClient' :	apache-client
Define value for property 'nativeImage' :	false
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. Setelah nilai terakhir dimasukkan, Maven mencantumkan pilihan yang Anda buat. Konfirmasikan dengan memasukkan Yatau memasukkan kembali nilai dengan memasukkan *N*.

Maven membuat folder proyek bernama `getstarted` berdasarkan `artifactId` nilai yang Anda masukkan. Di dalam `getstarted` folder, temukan `README.md` file yang dapat Anda tinjau, `pom.xml` file, dan `src` direktori.

Maven membangun pohon direktori berikut.

```
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

Berikut ini menunjukkan isi dari file `pom.xml` proyek.

pom.xml

dependencyManagementBagian ini berisi ketergantungan ke AWS SDK for Java 2.x dan dependencies bagian memiliki ketergantungan untuk Amazon DynamoDB. Menentukan dependensi ini memaksa Maven untuk menyertakan file jar yang relevan ke jalur kelas Java Anda. Secara default, AWS SDK tidak menyertakan semua kelas untuk semua Layanan AWS. Untuk DynamoDB, Anda harus memiliki ketergantungan dynamodb pada artefak jika Anda menggunakan antarmuka tingkat rendah dynamodb-enhanced atau jika Anda menggunakan antarmuka tingkat tinggi. Jika Anda tidak menyertakan dependensi yang relevan, kode Anda tidak akan dikompilasi. Proyek ini menggunakan Java 1.8 karena 1.8 nilai dalam `maven.compiler.source` dan `maven.compiler.target` properti.

```
<?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>ssoidc</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>
```

Langkah 3: Tulis kodenya

Kode berikut menunjukkan App kelas yang dibuat oleh Maven. `mainMetode` ini adalah titik masuk ke dalam aplikasi, yang menciptakan sebuah instance dari `Handler` kelas dan kemudian memanggil `sendRequest` metodenya.

App kelas

```
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");
    }
}
```

DependencyFactoryKelas yang dibuat oleh Maven berisi metode `dynamoDbClient` pabrik yang membangun dan mengembalikan sebuah instance. [DynamoDbClient](#) `DynamoDbClientInstance` menggunakan instance dari klien HTTP berbasis Apache. Ini karena Anda menentukan `apache-client` kapan Maven meminta Anda untuk klien HTTP mana yang akan digunakan.

DependencyFactoryHal ini ditunjukkan dalam kode berikut.

DependencyFactory kelas

```
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();
    }
}
```

HandlerKelas berisi logika utama program Anda. Ketika sebuah instance Handler dibuat di App kelas, DependencyFactory melengkapi klien DynamoDbClient layanan. Kode Anda menggunakan DynamoDbClient instance untuk memanggil layanan DynamoDB.

Maven menghasilkan Handler kelas berikut dengan komentar. *TODO* Langkah selanjutnya dalam tutorial menggantikan kode *TODO*dengan.

Handlerkelas, yang dihasilkan oleh Maven

```
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.
    }
}
```

Untuk mengisi logika, ganti seluruh isi Handler kelas dengan kode berikut. `sendRequest` Metode diisi dan impor yang diperlukan ditambahkan.

Kelas handler, diimplementasikan

Kode berikut menggunakan [DynamoDbClient](#) contoh untuk mengambil daftar tabel yang ada. Jika tabel ada untuk akun dan Wilayah tertentu, kode menggunakan `Logger` contoh untuk mencatat nama-nama tabel ini.

```
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));
}
}
```

Langkah 4: Bangun dan jalankan aplikasi

Setelah proyek dibuat dan berisi Handler kelas lengkap, bangun dan jalankan aplikasi.

1. Pastikan Anda memiliki sesi IAM Identity Center yang aktif. Untuk melakukannya, jalankan AWS Command Line Interface perintah `aws sts get-caller-identity` dan periksa responsnya. Jika Anda tidak memiliki sesi aktif, lihat [Masuk menggunakan AWS CLI](#) petunjuk.
2. Buka terminal atau jendela prompt perintah dan arahkan ke direktori proyek `Andagetstarted`.
3. Gunakan perintah berikut untuk membangun proyek Anda:

```
mvn clean package
```

4. Gunakan perintah berikut untuk menjalankan aplikasi.

```
mvn exec:java -Dexec.mainClass="org.example.App"
```

Setelah Anda melihat file, hapus objek, dan kemudian hapus ember.

Berhasil

Jika proyek Maven Anda dibangun dan berjalan tanpa kesalahan, maka selamat! Anda telah berhasil membangun aplikasi Java pertama Anda menggunakan SDK for Java 2.x.

Pembersihan

Untuk membersihkan sumber daya yang Anda buat selama tutorial ini, lakukan hal berikut:

- Hapus folder proyek `getstarted`.

Menggunakan AWS SDK for Java 2.x dokumentasi

[AWS SDK for Java 2.x Dokumentasi](#) mencakup semua aspek SDK di semua Layanan AWS.

Gunakan topik berikut sebagai titik awal Anda:

- [Migrasi dari versi 1.x ke 2.x](#) - Termasuk penjelasan rinci tentang perbedaan antara 1.x dan 2.x. Topik ini juga berisi petunjuk tentang cara menggunakan kedua versi utama side-by-side.
- [Panduan DynamoDB untuk Java 2.x SDK](#) - Menunjukkan cara melakukan operasi DynamoDB dasar: membuat tabel, memanipulasi item, dan mengambil item. Contoh-contoh ini menggunakan antarmuka tingkat rendah. Java memiliki beberapa antarmuka, seperti yang dijelaskan di bagian berikut: [Antarmuka yang didukung](#)

Tip

Kami menyarankan bahwa setelah melalui topik-topik ini dalam AWS SDK for Java 2.x dokumentasi, Anda menandai dokumentasi [Javadoc](#). Ini mencakup semua Layanan AWS dan akan bertindak sebagai referensi API utama Anda.

Antarmuka yang didukung

AWS SDK for Java 2.x Mendukung antarmuka berikut tergantung pada tingkat abstraksi yang Anda inginkan.

Topik di bagian ini

- [Antarmuka tingkat rendah](#)
- [Antarmuka tingkat tinggi](#)
- [Antarmuka Dokumen](#)
- [Membandingkan antarmuka dengan contoh Query](#)

Antarmuka tingkat rendah

Antarmuka tingkat rendah menyediakan one-to-one pemetaan ke API layanan yang mendasarinya. Setiap DynamoDB API tersedia melalui antarmuka ini. Ini berarti antarmuka tingkat rendah dapat menyediakan fungsionalitas yang lengkap, tetapi seringkali lebih bertele-tele dan kompleks untuk digunakan. Misalnya, Anda harus menggunakan `.s()` fungsi untuk menahan string dan.

`putItem()` berfungsi untuk menahan angka. Contoh berikut [PutItem](#) menyisipkan item menggunakan antarmuka tingkat rendah.

```
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)");
    }
}
```

Antarmuka tingkat tinggi

Antarmuka tingkat tinggi di AWS SDK for Java 2.x disebut DynamoDB Enhanced Client. Antarmuka ini memberikan pengalaman penulisan kode yang lebih idiomatis.

Klien yang disempurnakan menawarkan cara untuk memetakan antara kelas data sisi klien dan tabel DynamoDB yang dirancang untuk menyimpan data tersebut. Anda menentukan hubungan

antara tabel dan kelas model yang sesuai dalam kode Anda. Kemudian, Anda dapat mengandalkan SDK untuk mengelola manipulasi tipe data. Untuk informasi selengkapnya tentang klien yang disempurnakan, lihat [DynamoDB API klien yang disempurnakan](#) dalam dokumentasi AWS SDK for Java 2.x

Contoh berikut [PutItem](#) menggunakan antarmuka tingkat tinggi. Dalam contoh ini, `DynamoDbBean` nama `YourItem` menciptakan `TableSchema` memungkinkan penggunaan langsung sebagai input untuk `putItem()` panggilan.

```
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.x memiliki antarmuka tingkat tinggi sendiri, yang sering disebut oleh kelas utamanya. `DynamoDBMapper` AWS SDK for Java 2.x ini diterbitkan dalam paket terpisah (dan artefak Maven) bernama `software.amazon.awssdk.enhanced.dynamodb`. Java 2.x SDK sering disebut oleh kelas utamanya. `DynamoDbEnhancedClient`

Antarmuka tingkat tinggi menggunakan kelas data yang tidak dapat diubah

Fitur pemetaan API klien yang disempurnakan DynamoDB juga berfungsi dengan kelas data yang tidak dapat diubah. Kelas yang tidak dapat diubah hanya memiliki getter dan memerlukan kelas pembangun yang digunakan SDK untuk membuat instance kelas. Kekekalan di Java adalah gaya yang umum digunakan yang memungkinkan pengembang untuk membuat kelas yang bebas efek samping dan, oleh karena itu, lebih dapat diprediksi dalam perilaku mereka dalam aplikasi multi-threaded yang kompleks. Alih-alih menggunakan `@DynamoDbBean` anotasi seperti yang ditunjukkan dalam [High-level interface example](#), kelas yang tidak dapat diubah menggunakan `@DynamoDbImmutable` anotasi, yang mengambil kelas pembangun sebagai inputnya.

Contoh berikut mengambil kelas builder `DynamoDbEnhancedClientImmutablePutItem` sebagai masukan untuk membuat skema tabel. Contoh kemudian memberikan skema sebagai input untuk panggilan [PutItemAPI](#).

```
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)");
    }
}
```

Contoh berikut menunjukkan kelas data yang tidak dapat diubah.

```
@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); }
}
}
```

Antarmuka tingkat tinggi menggunakan kelas data yang tidak dapat diubah dan pustaka generasi boilerplate pihak ketiga

Contoh kelas data yang tidak dapat diubah yang disebutkan di bagian sebelumnya memerlukan beberapa kode boilerplate. Misalnya, logika pengambil dan penyetel pada kelas data selain kelas Builder. Pustaka pihak ketiga, seperti [Project Lombok](#), dapat membantu Anda menghasilkan jenis kode boilerplate tersebut.

[Perpustakaan pihak ketiga Use, seperti bagian Lombok](#) dalam AWS SDK for Java 2.x dokumentasi memperkenalkan konsep pemanfaatan perpustakaan Project Lombok. Mengurangi sebagian besar kode boilerplate membantu Anda membatasi jumlah kode yang diperlukan untuk bekerja dengan kelas data yang tidak dapat diubah dan SDK. AWS Ini selanjutnya menghasilkan peningkatan produktivitas dan keterbacaan kode Anda.

Contoh berikut menunjukkan bagaimana Project Lombok menyederhanakan kode yang diperlukan untuk menggunakan DynamoDB ditingkatkan client API.

```
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.class)
            .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)");
    }
}

```

Contoh berikut menunjukkan objek data yang tidak berubah dari kelas data yang tidak dapat diubah.

```

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;  
}
```

YourImmutableLombokItemKelas menggunakan anotasi berikut yang disediakan oleh Project Lombok dan SDK: AWS

- [@Builder](#) — Menghasilkan API pembangun kompleks untuk kelas data yang disediakan oleh Project Lombok.
- [@DynamoDbImmutable](#) — Mengidentifikasi DynamoDbImmutable kelas sebagai anotasi entitas DynamoDB yang dapat dipetakan yang disediakan oleh SDK. AWS
- [@Value](#) - Varian yang tidak dapat diubah dari `@Data`; semua bidang dibuat pribadi dan final secara default, dan penyeter tidak dibuat. Anotasi ini disediakan oleh Project Lombok.

Antarmuka Dokumen

Antarmuka Dokumen menghindari kebutuhan untuk menentukan deskriptor tipe data. Jenis data yang tersirat oleh semantik data itu sendiri. Jika Anda terbiasa dengan antarmuka Dokumen AWS SDK for Java 1.x, antarmuka Dokumen di AWS SDK for Java 2.x menawarkan antarmuka yang serupa tetapi didesain ulang.

Berikut ini [Document interface example](#) menunjukkan `PutItem` panggilan yang dinyatakan menggunakan antarmuka Dokumen. Contoh ini juga menggunakan `EnhancedDocument`. Untuk menjalankan perintah terhadap tabel DynamoDB menggunakan API dokumen yang disempurnakan, Anda harus terlebih dahulu mengaitkan tabel dengan skema tabel dokumen Anda untuk membuat objek sumber daya `DynamoDbTable`. Pembuat skema tabel Dokumen memerlukan kunci indeks utama dan penyedia konverter atribut.

Anda dapat menggunakan `AttributeConverterProvider.defaultProvider()` untuk mengonversi atribut dokumen dari tipe default. Anda dapat mengubah keseluruhan perilaku default dengan `AttributeConverterProvider` implementasi kustom. Anda juga dapat mengubah konverter untuk satu atribut. [Dokumentasi AWS SDK](#) memberikan detail dan contoh selengkapnya tentang cara menggunakan konverter kustom. Penggunaan utamanya adalah untuk atribut kelas domain Anda yang tidak memiliki konverter default yang tersedia. Menggunakan konverter khusus, Anda dapat memberikan SDK dengan informasi yang diperlukan untuk menulis atau membaca ke DynamoDB.

```
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)");
    }
}
```

Dengan menggunakan metode utilitas berikut, Anda dapat mengonversi dokumen JSON ke dan dari tipe data Amazon DynamoDB asli:

- [EnhancedDocument.fromJson\(String json\)](#)—Menciptakan EnhancedDocument contoh baru dari string JSON.
- [EnhancedDocument.toJson\(\)](#)—Membuat representasi string JSON dari dokumen yang memungkinkan Anda untuk menggunakannya dalam aplikasi Anda seperti objek JSON lainnya.

Membandingkan antarmuka dengan contoh Query

Bagian ini menunjukkan panggilan [Query](#) yang sama yang diekspresikan menggunakan berbagai antarmuka. Kueri ini menggunakan beberapa atribut untuk menyempurnakan hasil kueri:

- Anda harus menentukan kunci partisi sepenuhnya karena DynamoDB akan menargetkan ke satu nilai kunci partisi tertentu.
- Kunci sortir memiliki ekspresi kondisi kunci yang digunakan `begins_with` sehingga hanya item keranjang yang ditargetkan dengan kueri ini.
- Kami membatasi kueri untuk mengembalikan maksimum 100 item dengan menggunakan `limit()`.
- Kami mengatur `scanIndexForward` ke `false`. Hasilnya dikembalikan dalam urutan UTF-8 byte, yang biasanya berarti item keranjang dengan jumlah terendah dikembalikan terlebih dahulu. Dengan menyetel `scanIndexForward` ke `false`, kami membalikkan pesanan dan item keranjang dengan angka tertinggi dikembalikan terlebih dahulu.
- Kami menerapkan filter untuk menghapus hasil apa pun yang tidak sesuai dengan kriteria. Data yang difilter menghabiskan kapasitas baca apakah item tersebut cocok dengan filter atau tidak.

Example Kueri menggunakan antarmuka tingkat rendah

Contoh berikut query tabel bernama `YourTableName` menggunakan `keyConditionExpression`, yang membatasi query untuk nilai kunci partisi tertentu dan mengurutkan nilai kunci yang dimulai dengan nilai awalan tertentu. Kondisi utama ini membatasi jumlah data yang dibaca dari DynamoDB. Akhirnya, kueri menerapkan filter pada data yang diambil dari DynamoDB menggunakan `filterExpression`

```
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 Kueri menggunakan antarmuka Dokumen

Contoh berikut query tabel bernama `YourTableName` menggunakan antarmuka dokumen.

```
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 Kueri menggunakan antarmuka tingkat tinggi

Contoh berikut menanyakan tabel bernama `YourTableName` menggunakan DynamoDB API klien yang disempurnakan.

```
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;
    }
}
}
```


Antarmuka tingkat tinggi menggunakan kelas data yang tidak dapat diubah

Saat Anda melakukan Query dengan kelas data immutable tingkat tinggi, kodenya sama dengan contoh antarmuka tingkat tinggi kecuali untuk konstruksi kelas entitas atau `YourItem` `YourImmutableItem` Untuk informasi lebih lanjut, lihat [PutItem](#) contoh.

Antarmuka tingkat tinggi menggunakan kelas data yang tidak dapat diubah dan pustaka generasi boilerplate pihak ketiga

Saat Anda melakukan Query dengan kelas data immutable tingkat tinggi, kodenya sama dengan contoh antarmuka tingkat tinggi kecuali untuk konstruksi kelas entitas atau `YourItem` `YourImmutableLombokItem` Untuk informasi lebih lanjut, lihat [PutItem](#) contoh.

Contoh kode tambahan

Anda juga dapat merujuk ke repositori contoh kode berikut yang memberikan contoh tambahan untuk menggunakan DynamoDB dengan SDK for Java 2.x:

- [Contoh kode AWS aksi tunggal resmi](#)
- [Contoh kode aksi tunggal yang dikelola komunitas](#)
- [Contoh kode AWS berorientasi skenario resmi](#)

Pemrograman sinkron dan asinkron

AWS SDK for Java 2.x Ini menyediakan klien sinkron dan asinkron untuk, Layanan AWS seperti DynamoDB.

`DynamoDbEnhancedClient` Kelas `DynamoDbClient` dan menyediakan metode sinkron yang memblokir eksekusi thread Anda hingga klien menerima respons dari layanan. Klien ini adalah cara paling mudah untuk berinteraksi dengan DynamoDB jika Anda tidak memerlukan operasi asinkron.

`DynamoDbEnhancedAsyncClient` Kelas `DynamoDbAsyncClient` dan menyediakan metode asinkron yang segera kembali, dan memberikan kontrol kembali ke utas panggilan tanpa menunggu respons. Klien non-pemblokiran memiliki keuntungan yang memungkinkan konkurensi tinggi di beberapa utas, yang menyediakan penanganan permintaan I/O yang efisien dengan sumber daya komputasi minimal. Ini meningkatkan throughput dan daya tanggap.

AWS SDK for Java 2.x Menggunakan dukungan asli untuk non-blocking I/O. AWS SDK for Java 1.x harus mensimulasikan non-blocking I/O.

Metode sinkron kembali sebelum respons tersedia, jadi Anda memerlukan cara untuk mendapatkan respons saat sudah siap. Metode asinkron dalam AWS SDK for Java mengembalikan [CompletableFuture](#) objek yang berisi hasil operasi asinkron di masa depan. Ketika Anda memanggil `get()` atau `join()` pada `CompletableFuture` objek ini, kode Anda akan memblokir sampai hasilnya tersedia. Jika Anda melakukan panggilan ini bersamaan dengan membuat permintaan, perilakunya mirip dengan panggilan sinkron biasa.

Dokumentasi AWS SDK untuk [pemrograman asinkron](#) memberikan rincian lebih lanjut tentang cara memanfaatkan gaya asinkron ini.

Klien HTTP

Untuk mendukung setiap klien, ada klien HTTP yang menangani komunikasi dengan klien Layanan AWS. Anda dapat mencolokkan klien HTTP alternatif, memilih salah satu yang memiliki karakteristik yang paling sesuai dengan aplikasi Anda. Beberapa lebih ringan; beberapa memiliki lebih banyak opsi konfigurasi.

Beberapa klien HTTP hanya mendukung penggunaan sinkron, sementara yang lain hanya mendukung penggunaan asinkron. Dokumentasi AWS SDK menyediakan [diagram alur](#) yang membantu Anda memilih klien HTTP yang optimal untuk beban kerja Anda.

Daftar berikut menyajikan beberapa kemungkinan klien HTTP:

Topik

- [Klien HTTP berbasis Apache](#)
- [Klien HTTP berbasis URLConnection](#)
- [Klien HTTP berbasis Netty](#)
- [AWS Klien HTTP berbasis CRT](#)

Klien HTTP berbasis Apache

[ApacheHttpClient](#) mendukung klien layanan sinkron dan merupakan klien HTTP default untuk penggunaan sinkron. Untuk informasi tentang mengonfigurasi `ApacheHttpClient`, lihat [Mengkonfigurasi klien HTTP berbasis Apache](#) dalam dokumentasi. AWS SDK for Java 2.x

Klien HTTP berbasis URLConnection

[URLConnectionHttpClient](#) ini adalah opsi lain untuk klien sinkron. Ini memuat lebih cepat daripada klien HTTP berbasis Apache, tetapi memiliki lebih sedikit fitur. Untuk informasi tentang mengonfigurasi `URLConnectionHttpClient`, lihat [Mengkonfigurasi klien HTTP berbasis URLConnection](#).

Klien HTTP berbasis Netty

`NettyNioAsyncHttpClient` mendukung klien async dan merupakan pilihan default untuk penggunaan async. Untuk informasi tentang mengonfigurasi `NettyNioAsyncHttpClient`, lihat [Mengkonfigurasi klien HTTP berbasis Netty](#).

AWS Klien HTTP berbasis CRT

Yang lebih baru `AwsCrtHttpClient` dan `AwsCrtAsyncHttpClient` dari pustaka AWS Common Runtime (CRT) adalah opsi lain yang mendukung klien sinkron dan asinkron. Dibandingkan dengan klien HTTP lainnya, AWS CRT menawarkan:

- Waktu startup SDK lebih cepat
- Jejak memori yang lebih kecil
- Mengurangi waktu latensi
- Manajemen kesehatan koneksi
- Penyeimbangan beban DNS

Untuk informasi tentang mengkonfigurasi `AwsCrtHttpClient` dan `AwsCrtAsyncHttpClient`, lihat [Mengkonfigurasi klien HTTP AWS berbasis CRT](#).

AWS Klien HTTP berbasis CRT bukan default karena itu akan merusak kompatibilitas mundur untuk aplikasi yang ada. Namun, untuk DynamoDB kami menyarankan Anda menggunakan klien HTTP berbasis CRT untuk penggunaan AWS sinkronisasi dan asinkron.

Untuk pengenalan tentang klien HTTP AWS berbasis CRT, lihat [Mengumumkan ketersediaan AWS CRT HTTP Client di](#). AWS SDK for Java 2.x

Mengkonfigurasi klien HTTP

Saat mengonfigurasi klien, Anda dapat memberikan berbagai opsi konfigurasi, termasuk:

- Menyetel batas waktu untuk berbagai aspek panggilan API
- Mengontrol jika TCP Keep-Alive diaktifkan
- Mengontrol kebijakan coba lagi saat mengalami kesalahan
- Menentukan atribut [eksekusi yang dapat dimodifikasi oleh instance pencegat](#) Eksekusi. Pencegat eksekusi dapat menulis kode yang mencegat eksekusi permintaan dan tanggapan API Anda. Ini memungkinkan Anda untuk melakukan tugas, seperti mempublikasikan metrik dan memodifikasi permintaan dalam penerbangan.
- Menambahkan atau memanipulasi header HTTP
- Mengaktifkan pelacakan metrik [kinerja sisi klien](#). Menggunakan fitur ini membantu Anda mengumpulkan metrik tentang klien layanan di aplikasi Anda dan menganalisis output di Amazon CloudWatch.
- Menentukan layanan pelaksana alternatif yang akan digunakan untuk menjadwalkan tugas, seperti percobaan ulang async dan tugas batas waktu

Anda mengontrol konfigurasi dengan menyediakan [ClientOverrideConfiguration](#) objek ke `Builder` kelas klien layanan. Anda akan melihat ini di beberapa contoh kode di bagian berikut.

`ClientOverrideConfiguration` ini menyediakan pilihan konfigurasi standar. Klien HTTP pluggable yang berbeda memiliki kemungkinan konfigurasi khusus implementasi juga. Masing-masing klien ini juga memelihara dokumentasinya sendiri:

- [Klien HTTP berbasis Apache](#)
- [Klien HTTP berbasis URLConnection](#)
- [Klien HTTP berbasis Netty](#)
- [AWS Klien HTTP berbasis CRT](#)

Topik di bagian ini

- [Konfigurasi waktu habis](#)
- [RetryMode](#)
- [DefaultsMode](#)
- [Konfigurasi Keep-Alive](#)

Konfigurasi waktu habis

Anda dapat menyesuaikan konfigurasi klien untuk mengontrol berbagai batas waktu yang terkait dengan panggilan layanan. DynamoDB memberikan latensi yang lebih rendah dibandingkan dengan yang lain. Layanan AWS Oleh karena itu, Anda mungkin ingin menyesuaikan properti ini untuk menurunkan nilai batas waktu sehingga Anda dapat gagal dengan cepat jika ada masalah jaringan.

Anda dapat menyesuaikan perilaku terkait latensi menggunakan `ClientOverrideConfiguration` pada klien DynamoDB atau dengan mengubah opsi konfigurasi terperinci pada implementasi klien HTTP yang mendasarinya.

Anda dapat mengonfigurasi properti berdampak berikut menggunakan:

`ClientOverrideConfiguration`

- `apiCallAttemptTimeout`—Jumlah waktu untuk menunggu satu upaya untuk menyelesaikan permintaan HTTP sebelum menyerah dan waktu habis.
- `apiCallTimeout`—Jumlah waktu untuk memungkinkan klien menyelesaikan eksekusi panggilan API. Ini termasuk eksekusi penanganan permintaan yang terdiri dari semua permintaan HTTP termasuk percobaan ulang.

AWS SDK for Java 2.x Ini memberikan [nilai default](#) untuk beberapa opsi batas waktu, seperti batas waktu koneksi dan batas waktu soket. SDK tidak memberikan nilai default untuk batas waktu panggilan API atau batas waktu percobaan panggilan API individual. Jika batas waktu ini tidak disetel `ClientOverrideConfiguration`, SDK akan menggunakan nilai batas waktu soket sebagai gantinya untuk batas waktu panggilan API secara keseluruhan. Batas waktu soket memiliki nilai default 30 detik.

RetryMode

Konfigurasi lain yang terkait dengan konfigurasi batas waktu yang harus Anda pertimbangkan adalah objek `RetryMode` konfigurasi. Objek konfigurasi ini berisi kumpulan perilaku coba lagi.

SDK for Java 2.x mendukung mode coba lagi berikut:

- `Legacy`— Mode coba lagi default jika Anda tidak mengubahnya secara eksplisit. Mode coba lagi ini khusus untuk Java SDK dan ditandai dengan:
 - Hingga tiga percobaan ulang, atau lebih untuk layanan, seperti DynamoDB yang memiliki hingga 8 percobaan ulang.

- **standard**— Dinamakan standar karena lebih konsisten dengan AWS SDK lain. Mode ini menunggu jumlah waktu acak mulai dari 0ms hingga 1.000 ms untuk percobaan ulang pertama. Jika percobaan lagi diperlukan, ia mengambil waktu acak lain dari 0ms menjadi 1.000 ms dan mengalikannya dengan dua. Jika percobaan ulang tambahan diperlukan, ia melakukan pengambilan acak yang sama dikalikan dengan 4, dan seterusnya. Setiap penantian dibatasi pada 20 detik. Mode ini akan melakukan percobaan ulang pada kondisi kegagalan yang lebih terdeteksi daripada legacy mode. Untuk DynamoDB, ia melakukan hingga tiga upaya maksimal total kecuali Anda mengganti dengan [numRetries](#)
- **adaptive**— Dibangun pada standard mode dan secara dinamis membatasi tingkat AWS permintaan untuk memaksimalkan tingkat keberhasilan. Ini dapat dilakukan dengan mengorbankan latensi permintaan. Kami tidak merekomendasikan mode coba lagi adaptif ketika latensi yang dapat diprediksi penting.

Anda dapat menemukan definisi yang diperluas dari mode coba lagi ini dalam topik [perilaku Coba lagi](#) di Panduan Referensi AWS SDK dan Alat.

Coba lagi kebijakan

Semua RetryMode konfigurasi memiliki [RetryPolicy](#), yang dibangun berdasarkan satu atau lebih [RetryCondition](#) konfigurasi. [TokenBucketRetryCondition](#) Ini sangat penting untuk perilaku coba lagi implementasi klien DynamoDB SDK. Kondisi ini membatasi jumlah percobaan ulang yang dilakukan oleh SDK menggunakan algoritma token bucket. Bergantung pada mode coba lagi yang dipilih, pengecualian pembatasan mungkin atau mungkin tidak mengurangi token dari TokenBucket

Ketika klien menemukan kesalahan yang dapat dicoba ulang, seperti pengecualian pembatasan atau kesalahan server sementara, SDK akan secara otomatis mencoba ulang permintaan tersebut. Anda dapat mengontrol berapa kali dan seberapa cepat percobaan ulang ini terjadi.

Saat mengkonfigurasi klien, Anda dapat memberikan RetryPolicy yang mendukung parameter berikut:

- **numRetries** Jumlah maksimum percobaan ulang yang harus diterapkan sebelum permintaan dianggap gagal. Nilai default adalah 8 terlepas dari mode coba lagi yang Anda gunakan.

Warning

Pastikan Anda mengubah nilai default ini setelah mempertimbangkan.

- `backoffStrategy` [BackoffStrategy](#)—Untuk diterapkan pada percobaan ulang, dengan [FullJitterBackoffStrategy](#) menjadi strategi default. Strategi ini melakukan penundaan eksponensial antara percobaan ulang tambahan berdasarkan jumlah saat ini atau percobaan ulang, penundaan dasar, dan waktu backoff maksimum. Kemudian menambahkan jitter untuk memberikan sedikit keacakan. Penundaan dasar yang digunakan dalam penundaan eksponensial adalah 25 ms terlepas dari mode coba lagi.
- `retryCondition`—The [RetryCondition](#) menentukan apakah permintaan harus dicoba lagi sama sekali. Secara default ia akan mencoba lagi satu set kode status HTTP tertentu dan pengecualian yang diyakini dapat dicoba ulang. Untuk sebagian besar situasi, konfigurasi default harus cukup.

Kode berikut menyediakan kebijakan coba lagi alternatif. Ini menentukan untuk memungkinkan total lima percobaan ulang (enam permintaan total). Percobaan ulang pertama harus dilakukan setelah penundaan sekitar 100 ms, dengan setiap percobaan lagi tambahan menggandakan waktu itu secara eksponensial, hingga penundaan maksimum satu detik.

```
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();
```

DefaultsMode

Properti batas waktu tidak dikelola oleh `ClientOverrideConfiguration` dan biasanya tidak dikonfigurasi secara eksplisit. `RetryMode` Sebaliknya, konfigurasi mereka diatur secara implisit dengan menentukan a. `DefaultsMode`

Support for `DefaultsMode` diperkenalkan di AWS SDK for Java 2.x (versi 2.17.102 atau yang lebih baru) untuk menyediakan serangkaian nilai default untuk pengaturan umum yang dapat dikonfigurasi, seperti pengaturan komunikasi HTTP, perilaku coba lagi, pengaturan titik akhir Regional layanan, dan, berpotensi, konfigurasi terkait SDK apa pun. Saat Anda menggunakan fitur ini, Anda bisa mendapatkan default konfigurasi baru yang disesuaikan dengan skenario penggunaan umum.

Mode default distandarisasi di semua SDK. AWS SDK for Java 2.x mendukung mode default berikut:

- `legacy`— Menyediakan pengaturan default yang bervariasi menurut SDK dan ada sebelum `DefaultsMode` dibuat.
- `standard`— Menyediakan pengaturan default yang tidak dioptimalkan untuk sebagian besar skenario.
- `in-region`— Dibangun pada mode standar dan termasuk pengaturan yang disesuaikan untuk aplikasi yang memanggil Layanan AWS dari dalam yang sama. Wilayah AWS
- `cross-region`— Dibangun pada mode standar dan termasuk pengaturan dengan batas waktu tinggi untuk aplikasi yang memanggil Layanan AWS yang berbeda. Wilayah AWS
- `mobile`— Dibangun pada mode standar dan mencakup pengaturan dengan batas waktu tinggi yang disesuaikan untuk aplikasi seluler dengan latensi lebih tinggi.
- `auto`— Dibangun pada mode standar dan termasuk fitur eksperimental. SDK mencoba menemukan lingkungan runtime untuk menentukan pengaturan yang sesuai secara otomatis. Deteksi otomatis berbasis heuristik dan tidak memberikan akurasi 100%. Jika lingkungan runtime tidak dapat ditentukan, mode standar digunakan. Deteksi otomatis mungkin menanyakan [metadata Instance dan data pengguna](#), yang mungkin memperkenalkan latensi. Jika latensi startup sangat penting untuk aplikasi Anda, sebaiknya pilih yang eksplisit `DefaultsMode`.

Anda dapat mengonfigurasi mode default dengan cara berikut:

- Langsung pada klien melalui `AwsClientBuilder.Builder#defaultsMode(DefaultsMode)`.
- Pada profil konfigurasi melalui properti file `defaults_mode` profil.
- Secara global melalui properti `aws.defaultsMode` sistem.
- Secara global melalui variabel `AWS_DEFAULTS_MODE` lingkungan.

Note

Untuk mode apa pun selain `legacy`, nilai default yang dijual dapat berubah seiring berkembangnya praktik terbaik. Oleh karena itu, kami mendorong Anda untuk melakukan pengujian saat memutakhirkan SDK jika Anda menggunakan mode selain `legacy`.

[Konfigurasi Smart default](#) di AWS SDK dan Tools Reference Guide menyediakan daftar properti konfigurasi dan nilai defaultnya dalam mode default yang berbeda.

Anda memilih nilai mode default berdasarkan karakteristik aplikasi Anda dan aplikasi Layanan AWS Anda berinteraksi dengannya.

Nilai-nilai ini dikonfigurasi dengan berbagai pilihan Layanan AWS dalam pikiran. Untuk penerapan DynamoDB tipikal di mana tabel DynamoDB dan aplikasi Anda digunakan dalam satu Wilayah, mode default `in-region` paling relevan di antara mode default. `standard`

Example Konfigurasi klien DynamoDB SDK disetel untuk panggilan latensi rendah

Contoh berikut menyesuaikan batas waktu ke nilai yang lebih rendah untuk panggilan DynamoDB latensi rendah yang diharapkan.

```
DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.builder()
    .defaultsMode(DefaultsMode.IN_REGION)
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder())
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(3))
        .apiCallAttemptTimeout(Duration.ofMillis(500))
        .build())
    .build();
```

Implementasi klien HTTP individual dapat memberi Anda kontrol yang lebih terperinci atas batas waktu dan perilaku penggunaan koneksi. Misalnya, dalam kasus klien AWS berbasis CRT, Anda dapat mengaktifkan `ConnectionHealthConfiguration` yang memungkinkan klien AWS berbasis CRT untuk secara aktif memantau kesehatan koneksi yang digunakan. Untuk detail selengkapnya, lihat [dokumentasi](#) untuk AWS SDK for Java 2.x.

Konfigurasi Keep-Alive

Mengaktifkan keep-alive dapat mengurangi latensi dengan menggunakan kembali koneksi. Ada dua jenis keep-alive: HTTP Keep-Alive dan TCP Keep-Alive.

- HTTP Keep-Alive mencoba mempertahankan koneksi HTTPS antara klien dan server sehingga permintaan selanjutnya dapat menggunakan kembali koneksi tersebut. Ini melewati otentikasi HTTPS kelas berat pada permintaan selanjutnya. HTTP Keep-Alive diaktifkan secara default pada semua klien.
- TCP Keep-Alive meminta sistem operasi yang mendasarinya untuk mengirim paket kecil melalui koneksi soket untuk memberikan jaminan ekstra bahwa soket tetap hidup dan segera mendeteksi tetesan apa pun. Ini memastikan permintaan nanti tidak akan menghabiskan waktu mencoba menggunakan soket yang terjatuh. TCP Keep-Alive dinonaktifkan secara default pada semua klien.

Anda mungkin ingin mengaktifkannya. Contoh kode berikut menunjukkan bagaimana melakukan ini dengan setiap klien HTTP. Ketika diaktifkan untuk semua klien HTTP berbasis non-CRT, mekanisme Keep-Alive yang sebenarnya bergantung pada sistem operasi. Oleh karena itu, Anda harus mengonfigurasi nilai TCP Keep-Alive tambahan, seperti batas waktu dan jumlah paket, melalui sistem operasi. Anda dapat melakukan ini menggunakan `sysctl` pada mesin Linux atau Mac, atau nilai Registry pada mesin Windows.

Example untuk mengaktifkan TCP Keep-Alive pada klien HTTP berbasis Apache

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(ApacheHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Klien HTTP berbasis URLConnection

[Setiap klien sinkron yang menggunakan klien HTTP berbasis URLConnection](#)
[HttpURLConnection](#) tidak memiliki mekanisme untuk mengaktifkan keep-alive.

Example untuk mengaktifkan TCP Keep-Alive pada klien HTTP berbasis Netty

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Example untuk mengaktifkan TCP Keep-Alive pada klien HTTP berbasis CRT AWS

Dengan klien HTTP AWS berbasis CRT, Anda dapat mengaktifkan TCP keep-alive dan mengontrol durasinya.

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(AwsCrtHttpClient.builder()
        .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
            .keepAliveInterval(Duration.ofSeconds(50))
            .keepAliveTimeout(Duration.ofSeconds(5))
            .build()))
    .build();
```

Saat menggunakan klien DynamoDB asinkron, Anda dapat mengaktifkan TCP Keep-Alive seperti yang ditunjukkan pada kode berikut.

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()
        .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
            .keepAliveInterval(Duration.ofSeconds(50))
            .keepAliveTimeout(Duration.ofSeconds(5))
            .build()))
    .build();
```

Penanganan kesalahan

Ketika datang ke penanganan pengecualian, AWS SDK for Java 2.x menggunakan pengecualian runtime (tidak dicentang).

Pengecualian dasar, yang mencakup semua pengecualian SDK, adalah [SdkServiceException](#), yang meluas dari Java tidak dicentang. RuntimeException Jika Anda menangkap ini, Anda akan menangkap semua pengecualian yang dilemparkan oleh SDK.

[SdkServiceException](#) memiliki subclass yang disebut [AwsServiceException](#). Subkelas ini menunjukkan masalah apa pun dalam komunikasi dengan Layanan AWS. Ini memiliki subclass yang disebut [DynamoDbException](#), yang menunjukkan masalah dalam komunikasi dengan DynamoDB. Jika Anda menangkap ini, Anda akan menangkap semua pengecualian yang terkait dengan DynamoDB tetapi tidak ada pengecualian SDK lainnya.

Ada [jenis pengecualian](#) yang lebih spesifik di bawah [DynamoDbException](#). Beberapa jenis pengecualian ini berlaku untuk operasi bidang kontrol seperti [TableAlreadyExistsException](#). Lainnya berlaku untuk operasi data-plane. Berikut ini adalah contoh pengecualian bidang data umum:

- [ConditionalCheckFailedException](#)—Anda menentukan kondisi dalam permintaan yang dievaluasi menjadi false. Misalnya, Anda mungkin telah mencoba melakukan pembaruan bersyarat pada suatu item, tetapi nilai atribut sebenarnya tidak cocok dengan nilai yang diharapkan dalam kondisi yang dihadapi. Permintaan yang gagal dengan cara ini tidak akan dicoba lagi.

Situasi lain tidak memiliki pengecualian khusus yang ditentukan. Misalnya, ketika permintaan Anda dibatasi, spesifik [ProvisionedThroughputExceededException](#) mungkin dilemparkan, sementara dalam kasus lain permintaan yang lebih umum [DynamoDbException](#) dilemparkan. Dalam kedua kasus Anda dapat menentukan apakah pengecualian disebabkan oleh pelambatan dengan memeriksa apakah [isThrottlingException\(\)](#) pengembalian. true

Tergantung pada kebutuhan aplikasi Anda, Anda dapat menangkap semua `AwsServiceException` atau `DynamoDbException` instance. Tetapi seringkali Anda membutuhkan perilaku yang berbeda dalam situasi yang berbeda. Logika untuk menangani kegagalan pemeriksaan kondisi akan berbeda dibandingkan dengan berurusan dengan pelambatan. Tentukan jalur luar biasa mana yang ingin Anda tangani dan pastikan untuk menguji jalur alternatif. Ini membantu Anda memastikan Anda dapat menangani semua skenario yang relevan.

[Penanganan kesalahan dengan DynamoDB](#)Daftar beberapa kesalahan umum yang mungkin Anda temui. Anda juga dapat merujuk ke [Kesalahan Umum](#) untuk melihat daftar kesalahan umum. Untuk panggilan API tertentu, Anda juga dapat menemukan kesalahan persis yang mungkin terjadi bersama dengan dokumentasinya, seperti [Query](#) API. Untuk informasi tentang penanganan pengecualian, lihat [Penanganan pengecualian untuk AWS SDK for Java 2.x](#)

AWS ID permintaan

Setiap permintaan menyertakan ID Permintaan yang dapat berguna untuk ditarik jika Anda bekerja dengan AWS Support untuk mendiagnosis masalah. Setiap pengecualian yang berasal dari `SdkServiceException` memiliki metode [requestId\(\)](#) yang tersedia untuk mengambil ID permintaan.

Pencatatan log

Menggunakan logging yang disediakan oleh SDK dapat berguna baik untuk menangkap pesan penting dari pustaka klien dan untuk tujuan debugging yang lebih mendalam. Logger bersifat hierarkis dan SDK digunakan `software.amazon.awssdk` sebagai root logger-nya. Level dapat dikonfigurasi dengan salah satu `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `ALL`, atau `OFF`. Level yang dikonfigurasi akan berlaku untuk logger itu dan turun ke hierarki logger.

Ini AWS SDK for Java 2.x memanfaatkan Simple Logging Façade for Java (SLF4J) untuk logging nya, yang bertindak sebagai lapisan abstraksi di sekitar logger lainnya. Ini memungkinkan Anda untuk mencolokkan logger yang Anda inginkan. Untuk petunjuk tentang memasukkan logger, lihat panduan pengguna [SLF4J](#).

Setiap logger memiliki perilaku tertentu. Log4j 2.x logger secara default menciptakan sebuah `ConsoleAppender` yang menambahkan peristiwa log ke `System.out` dan default ke tingkat log. `ERROR`

`SimpleLogger` Logger yang disertakan dalam output SLF4J secara default ke `System.err` dan default ke tingkat log. `INFO`

Sebaiknya Anda mengatur level WARN untuk setiap penerapan produksi `software.amazon.awssdk` untuk menangkap pesan penting apa pun dari pustaka klien SDK sambil membatasi kuantitas keluaran.

[Jika SLF4J tidak dapat menemukan logger yang didukung di jalur kelas \(tidak ada pengikatan SLF4J\), SLF4J akan default ke implementasi tanpa operasi.](#) Implementasi ini menghasilkan pencatatan pesan untuk `System.err` menjelaskan bahwa SLF4J tidak dapat menemukan implementasi logger di classpath. Untuk mencegah situasi ini, Anda perlu menambahkan implementasi logger. Untuk melakukan ini, Anda dapat menambahkan ketergantungan di Apache Maven Anda `pom.xml` pada artefak, seperti `org.slf4j:slf4j-simple` atau `org.apache.logging.log4j:log4j-slf4j2-imp`

Dokumentasi untuk AWS SDK for Java 2.x menjelaskan cara mengonfigurasi logging di SDK termasuk menambahkan dependensi logging ke konfigurasi aplikasi Anda. Untuk selengkapnya, lihat [Logging dengan SDK for Java 2.x](#).

Konfigurasi berikut dalam `Log4j2.xml` file menunjukkan cara menyesuaikan perilaku logging jika Anda menggunakan logger Apache Log4j 2. Konfigurasi ini menetapkan level root logger ke WARN. Tingkat log ini akan diwarisi oleh semua logger dalam hierarki termasuk logger `software.amazon.awssdk`

Secara default, output akan masuk ke `System.out`. Dalam contoh berikut, kita masih mengganti appender Log4j keluaran default untuk menerapkan log4j yang disesuaikan. `PatternLayout`

Contoh file konfigurasi `Log4j2.xml`

Konfigurasi berikut akan mencatat pesan di ERROR dan WARN level ke konsol untuk semua hierarki logger.

```
<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>
```

```

</Loggers>
</Configuration>

```

AWS minta pencatatan ID

Anda dapat menemukan RequestIds dalam pengecualian ketika ada yang tidak beres. Namun, jika Anda menginginkan permintaan RequestIds untuk yang tidak menghasilkan pengecualian, Anda dapat menggunakan logging.

ID permintaan dikeluarkan pada DEBUG tingkat oleh logger. `software.amazon.awssdk.request` Contoh berikut memperluas sebelumnya [configuration example](#) untuk menjaga level root logger pada ERROR, level at WARN, dan level at. `software.amazon.awssdk` `software.amazon.awssdk.request` DEBUG Menyetel level ini membantu menangkap ID Permintaan dan detail terkait permintaan lainnya, seperti titik akhir dan kode status.

```

<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>

```

Berikut adalah contoh output log:

```

2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Sending Request:
DefaultSdkHttpRequestFullRequest(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:
QS9DUMME2NHEDH8TGT9N5V530JVV4KQNS05AEMVJF66Q9ASUAAJG, Extended Request ID: not
available

```

Paginasi

Beberapa permintaan, seperti [Query](#) dan [Scan](#), membatasi ukuran data yang dikembalikan pada satu permintaan dan mengharuskan Anda membuat permintaan berulang untuk menarik halaman berikutnya.

Anda dapat mengontrol jumlah maksimum item yang akan dibaca untuk setiap halaman dengan `Limit` parameter. Misalnya, Anda dapat menggunakan `Limit` parameter untuk mengambil hanya 10 item terakhir. Batas ini adalah berapa banyak item yang harus dibaca dari tabel sebelum penyaringan diterapkan. Tidak ada cara untuk menentukan yang Anda inginkan tepat 10 setelah pemfilteran. Anda hanya dapat mengontrol jumlah yang telah difilter sebelumnya dan memeriksa sisi klien ketika Anda benar-benar mengambil 10 item. Terlepas dari batasnya, setiap respons selalu memiliki ukuran maksimum 1 MB.

Dalam respons API, a `LastEvaluatedKey` mungkin disertakan, yang menunjukkan bahwa respons berakhir karena mencapai batas hitungan atau batas ukuran. Kunci ini adalah kunci terakhir yang dievaluasi untuk respons itu. Berinteraksi langsung dengan API Anda dapat mengambil ini `LastEvaluatedKey` dan meneruskannya ke panggilan tindak lanjut `ExclusiveStartKey` untuk membaca potongan berikutnya dari titik awal itu. Jika no `LastEvaluatedKey` dikembalikan, itu berarti tidak ada lagi item yang cocok dengan panggilan `Scan` API `Query` atau.

Contoh berikut menggunakan antarmuka tingkat rendah untuk membatasi item hingga 100 berdasarkan `keyConditionExpression` parameter.

```
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());
}
```

Ini AWS SDK for Java 2.x dapat menyederhanakan interaksi ini dengan DynamoDB dengan menyediakan metode pagination otomatis yang membuat beberapa panggilan layanan untuk mendapatkan halaman hasil berikutnya untuk Anda secara otomatis. Ini menyederhanakan kode Anda, tetapi menghilangkan beberapa kontrol pada penggunaan sumber daya yang akan Anda simpan dengan membaca halaman secara manual.

Dengan menggunakan `Iterable` metode yang tersedia di klien DynamoDB [QueryPaginator](#) seperti [ScanPaginator](#) dan, SDK menangani pagination. Jenis pengembalian metode ini adalah iterable khusus yang dapat Anda gunakan untuk mengulangi semua halaman. SDK akan menangani panggilan layanan secara internal untuk Anda. Menggunakan Java Stream API, Anda dapat menangani hasil `QueryPaginator` seperti yang ditunjukkan pada contoh berikut.

```
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();
```

Anotasi kelas data

Java SDK menyediakan beberapa anotasi yang dapat Anda masukkan pada atribut kelas data Anda yang akan memengaruhi cara SDK berinteraksi dengannya. Dengan menambahkan anotasi, Anda dapat memiliki atribut yang berperilaku sebagai penghitung atom implisit, mempertahankan nilai stempel waktu yang dibuat secara otomatis, atau melacak nomor versi item. Untuk informasi selengkapnya, lihat [Anotasi kelas data](#).

Bekerja dengan tabel, item, kueri, pindaian, dan indeks

Bagian ini menyediakan detail tentang bekerja dengan tabel, item, kueri, dan banyak lagi di Amazon DynamoDB.

Topik

- [Bekerja dengan tabel dan data di DynamoDB](#)
- [Tabel global - Replikasi multi-Wilayah untuk DynamoDB](#)
- [Bekerja dengan operasi baca dan tulis](#)
- [Meningkatkan akses data dengan indeks sekunder](#)
- [Mengelola alur kerja kompleks dengan DynamoDB Transactions](#)
- [Ubah tangkapan data dengan Amazon DynamoDB](#)
- [Menggunakan cadangan Sesuai Permintaan dan DynamoDB Permintaan](#)
- [oint-in-time Pemulihan P untuk DynamoDB](#)

Bekerja dengan tabel dan data di DynamoDB

Bagian ini menjelaskan cara menggunakan AWS Command Line Interface (AWS CLI) dan AWS SDK untuk membuat, memperbarui, dan menghapus tabel di Amazon DynamoDB.

Note

Anda juga dapat melakukan tugas-tugas yang sama ini menggunakan AWS Management Console. Untuk informasi selengkapnya, lihat [Menggunakan konsol](#).

Bagian ini juga menyediakan informasi selengkapnya tentang kapasitas throughput menggunakan penskalaan otomatis DynamoDB atau mengatur throughput yang disediakan secara manual.

Topik

- [Operasi dasar pada tabel DynamoDB](#)
- [Pertimbangan saat mengubah Mode Kapasitas baca/tulis](#)
- [Pertimbangan saat memilih kelas tabel](#)
- [Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB](#)

- [Ukuran dan format item DynamoDB](#)
- [Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB](#)
- [Menambahkan tag dan label ke sumber daya](#)
- [Bekerja dengan tabel DynamoDB di Java](#)
- [Bekerja dengan tabel DynamoDB di .NET](#)

Operasi dasar pada tabel DynamoDB

Mirip dengan sistem database lainnya, Amazon DynamoDB menyimpan data dalam tabel. Anda dapat mengelola tabel Anda menggunakan beberapa operasi dasar.

Topik

- [Membuat tabel](#)
- [Menjelaskan tabel](#)
- [Memperbarui tabel](#)
- [Menghapus tabel](#)
- [Mengggunakan perlindungan penghapusan](#)
- [Mencantumkan nama tabel](#)
- [Menggambarkan kuota throughput yang disediakan](#)

Membuat tabel

Gunakan operasi `CreateTable` untuk membuat tabel di Amazon DynamoDB. Untuk membuat tabel, Anda harus memberikan informasi berikut:

- Nama tabel. Nama harus sesuai dengan aturan penamaan DynamoDB, dan harus unik untuk akun saat ini dan Wilayah. AWS Misalnya, Anda dapat membuat tabel `People` di AS Timur (Virginia Utara) dan tabel `People` lainnya di Eropa (Irlandia). Namun, kedua tabel ini akan benar-benar berbeda satu sama lain. Untuk informasi selengkapnya, lihat [Jenis data dan aturan penamaan yang didukung di Amazon DynamoDB](#).
- Kunci primer. Kunci primer dapat terdiri dari satu atribut (kunci partisi) atau dua atribut (kunci partisi dan kunci urutan). Anda perlu memberikan nama atribut, tipe daya, dan peran dari masing-masing atribut: `HASH` (untuk kunci partisi) dan `RANGE` (untuk kunci urutan). Untuk informasi selengkapnya, lihat [Kunci primer](#).

- Pengaturan throughput (untuk tabel yang disediakan). Jika menggunakan mode yang disediakan, Anda harus menentukan pengaturan throughput tulis dan baca awal untuk tabel. Anda dapat mengubah pengaturan ini di waktu lain, atau mengaktifkan penskalaan otomatis DynamoDB untuk mengelola pengaturan untuk Anda. Untuk informasi selengkapnya, lihat [Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB](#) dan [Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB](#).

Contoh 1: Membuat tabel yang disediakan

AWS CLI Contoh berikut menunjukkan cara membuat tabel (Music). Kunci primer terdiri dari Artist (kunci partisi) dan SongTitle (kunci urutan), masing-masing memiliki tipe daya String. Throughput maksimum untuk tabel ini adalah 10 unit kapasitas baca dan 5 unit kapasitas tulis.

```
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
```

Operasi CreateTable mengembalikan metadata untuk tabel, seperti yang ditunjukkan berikut ini.

```
{  
  "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
}
}
```

Elemen `TableStatus` menunjukkan status tabel saat ini (`CREATING`). Mungkin perlu beberapa saat untuk membuat tabel, bergantung pada nilai yang Anda tentukan untuk `ReadCapacityUnits` dan `WriteCapacityUnits`. Nilai yang lebih besar untuk ini memerlukan DynamoDB mengalokasikan lebih banyak sumber daya untuk tabel.

Contoh 2: Membuat tabel sesuai permintaan

Untuk membuat tabel `Music` yang sama menggunakan mode sesuai permintaan.

```
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
```

Operasi `CreateTable` mengembalikan metadata untuk tabel, seperti yang ditunjukkan berikut ini.

```
{
  "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

Ketika memanggil `DescribeTable` pada tabel sesuai permintaan, unit kapasitas baca dan unit kapasitas tulis ditetapkan ke 0.

Contoh 3: Membuat tabel menggunakan kelas tabel akses standar-jarang DynamoDB

Untuk membuat tabel `Music` yang sama menggunakan kelas tabel akses standar-jarang DynamoDB

```
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
```

Operasi `CreateTable` mengembalikan metadata untuk tabel, seperti yang ditunjukkan berikut ini.

```
{  
  "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
  }
}
```

Menjelaskan tabel

Untuk melihat detail tentang tabel, gunakan operasi `DescribeTable`. Anda harus memberikan nama tabel. Output dari `DescribeTable` adalah dalam format yang sama seperti dari `CreateTable`. Ini mencakup stempel waktu pembuatan tabel, skema kunci, pengaturan throughput yang disediakan, perkiraan ukuran, dan indeks sekunder apa pun yang ada.

Important

Ketika menjalankan `DescribeTable` pada tabel sesuai permintaan, unit kapasitas baca dan unit kapasitas tulis ditetapkan ke 0.

Example

```
aws dynamodb describe-table --table-name Music
```

Tabel siap digunakan ketika `TableStatus` berubah dari `CREATING` menjadi `ACTIVE`.

Note

Jika Anda mengeluarkan permintaan `DescribeTable` segera setelah permintaan `CreateTable`, DynamoDB mungkin mengembalikan kesalahan (`ResourceNotFoundException`). Hal ini karena `DescribeTable` menggunakan kueri yang akhirnya konsisten, dan metadata untuk tabel Anda mungkin tidak tersedia pada saat itu. Tunggu selama beberapa detik, kemudian coba permintaan `DescribeTable` kembali. Untuk tujuan penagihan, biaya penyimpanan DynamoDB mencakup overhead per item sebesar 100 byte. (Untuk informasi selengkapnya, buka [Harga DynamoDB](#).) Tambahan 100 byte per item ini tidak digunakan dalam perhitungan unit kapasitas atau oleh operasi `DescribeTable`.

Memperbarui tabel

Operasi `UpdateTable` memungkinkan Anda melakukan salah satu hal berikut:

- Memodifikasi pengaturan throughput tabel yang disediakan (untuk tabel mode yang disediakan).
- Mengubah mode kapasitas baca/tulis tabel.
- Memanipulasi indeks sekunder global pada tabel (lihat [Menggunakan Indeks Sekunder Global di DynamoDB](#)).
- Mengaktifkan atau menonaktifkan DynamoDB Streams pada tabel (lihat [Ubah tangkapan data DynamoDB Streams](#)).

Example

AWS CLI Contoh berikut menunjukkan cara memodifikasi pengaturan throughput disediakan tabel.

```
aws dynamodb update-table --table-name Music \  
  --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10
```

Note

Ketika Anda mengeluarkan permintaan `UpdateTable`, status tabel berubah dari `AVAILABLE` menjadi `UPDATING`. Tabel tetap sepenuhnya tersedia untuk digunakan saat `UPDATING`. Setelah proses ini selesai, status tabel berubah dari `UPDATING` menjadi `AVAILABLE`.

Example

AWS CLI Contoh berikut menunjukkan cara memodifikasi mode kapasitas baca/tulis tabel ke mode on-demand.

```
aws dynamodb update-table --table-name Music \  
  --billing-mode PAY_PER_REQUEST
```

Menghapus tabel

Anda dapat menghapus tabel yang tidak digunakan dengan operasi `DeleteTable`. Menghapus tabel adalah operasi yang tidak dapat dipulihkan.

Example

AWS CLI Contoh berikut menunjukkan cara menghapus tabel.

```
aws dynamodb delete-table --table-name Music
```

Ketika Anda mengeluarkan permintaan `DeleteTable`, status tabel berubah dari `ACTIVE` menjadi `DELETING`. Mungkin perlu beberapa saat untuk menghapus tabel, bergantung pada sumber daya yang digunakan (seperti data yang disimpan dalam tabel, dan setiap aliran atau indeks pada tabel).

Di akhir operasi `DeleteTable`, tabel tidak akan ada lagi di DynamoDB.

Menggunakan perlindungan penghapusan

Anda dapat melindungi tabel dari penghapusan yang tidak disengaja dengan properti perlindungan penghapusan. Mengaktifkan properti ini untuk tabel membantu memastikan bahwa tabel tersebut tidak terhapus secara tidak sengaja selama operasi manajemen tabel normal administrator. Hal ini juga membantu mencegah gangguan terhadap operasi bisnis normal.

Pemilik tabel atau administrator yang berwenang mengontrol properti perlindungan penghapusan untuk setiap tabel. Properti perlindungan penghapusan untuk setiap tabel dinonaktifkan secara default. Ini termasuk replika global, dan tabel yang dipulihkan dari cadangan. Ketika perlindungan penghapusan dinonaktifkan untuk sebuah tabel, tabel tersebut dapat dihapus oleh pengguna mana pun yang diberi wewenang oleh kebijakan Manajemen Identitas dan Akses (IAM). Ketika perlindungan penghapusan diaktifkan untuk sebuah tabel, tabel tersebut tidak dapat dihapus oleh siapa pun.

Untuk mengubah pengaturan ini, buka Pengaturan tambahan tabel, arahkan ke panel Perlindungan Penghapusan dan pilih Aktifkan perlindungan penghapusan.

Properti proteksi penghapusan didukung oleh konsol DynamoDB, API, CLI/SDK dan. AWS CloudFormation `CreateTable` API mendukung properti proteksi penghapusan pada waktu pembuatan tabel, dan `UpdateTable` API mendukung perubahan properti proteksi penghapusan untuk tabel yang ada.

Note

- Jika AWS akun dihapus, semua data akun tersebut termasuk tabel masih dihapus dalam waktu 90 hari.
- Jika DynamoDB kehilangan akses ke kunci yang dikelola pelanggan yang digunakan untuk mengenkripsi tabel, DynamoDB akan tetap mengarsipkan tabel tersebut. Pengarsipan melibatkan membuat cadangan tabel dan menghapus aslinya.

Mencantumkan nama tabel

`ListTables` Operasi mengembalikan nama-nama tabel DynamoDB untuk akun AWS saat ini dan Wilayah.

Example

AWS CLI Contoh berikut menunjukkan bagaimana untuk daftar nama tabel DynamoDB.

```
aws dynamodb list-tables
```

Menggambarkan kuota throughput yang disediakan

`DescribeLimits` Operasi mengembalikan kuota kapasitas baca dan tulis saat ini untuk AWS akun saat ini dan Wilayah.

Example

AWS CLI Contoh berikut menunjukkan bagaimana mendeskripsikan kuota throughput yang disediakan saat ini.

```
aws dynamodb describe-limits
```

Output menunjukkan kuota atas unit kapasitas baca dan tulis untuk AWS akun saat ini dan Wilayah.

Untuk informasi selengkapnya tentang kuota ini, dan cara meminta peningkatan kuota, lihat [Kuota default throughput](#).

Pertimbangan saat mengubah Mode Kapasitas baca/tulis

Anda dapat beralih di antara mode kapasitas baca/tulis satu kali setiap 24 jam. Satu-satunya pengecualian untuk ini adalah jika Anda mengalihkan tabel mode yang disediakan ke mode sesuai permintaan: Anda dapat beralih kembali ke mode yang disediakan dalam periode 24 jam yang sama. Saat meningkatkan mode kapasitas baca/tulis di Amazon DynamoDB, pertimbangkan hal berikut.

Mengelola kapasitas

Saat Anda memperbarui tabel dari mode yang disediakan ke mode sesuai permintaan, Anda tidak perlu menentukan berapa banyak throughput baca dan tulis yang Anda harapkan untuk dijalankan aplikasi.

Saat memperbarui tabel dari mode sesuai permintaan ke mode yang ditetapkan, pertimbangkan hal berikut:

- Jika Anda menggunakan AWS CLI atau AWS SDK, pilih pengaturan kapasitas yang disediakan yang tepat untuk tabel dan indeks sekunder global dengan menggunakan Amazon CloudWatch untuk melihat konsumsi historis (`ConsumedWriteCapacityUnits` dan `ConsumedReadCapacityUnits` metrik) Anda guna menentukan setelan throughput baru.

Note

Jika Anda memindahkan tabel global ke mode yang ditetapkan, amati konsumsi maksimum di seluruh replika regional untuk tabel dasar global dan indeks sekunder saat membuat pengaturan throughput baru.

- Jika Anda beralih dari mode sesuai permintaan ke mode kapasitas yang dialokasikan, pastikan Anda menetapkan unit alokasi awal yang cukup untuk menangani kapasitas tabel atau indeks selama transisi.

Mengelola penskalaan otomatis

Saat Anda memperbarui tabel dari mode yang ditetapkan ke mode sesuai permintaan:

- Jika Anda menggunakan konsol, semua pengaturan penskalaan otomatis (jika ada) akan dihapus.
- Jika Anda menggunakan AWS CLI atau AWS SDK, semua pengaturan penskalaan otomatis Anda akan dipertahankan. Pengaturan ini dapat diterapkan ketika Anda memperbarui tabel lagi ke mode penagihan yang ditetapkan.
- Jika tabel adalah tabel global, pengaturan penskalaan otomatis tidak dipertahankan.

Saat Anda memperbarui tabel dari mode sesuai permintaan ke mode yang ditetapkan:

- Jika Anda menggunakan konsol tersebut, DynamoDB merekomendasikan untuk mengaktifkan penskalaan otomatis dengan pengaturan default berikut:
 - Target pemanfaatan: 70%
 - Kapasitas minimum yang disediakan: 5 unit
 - Kapasitas maksimum yang disediakan: Wilayah maksimum
- Jika Anda menggunakan AWS CLI atau SDK, pengaturan penskalaan otomatis Anda sebelumnya (jika ada) dipertahankan.
- Jika tabel adalah tabel global, pengaturan penskalaan otomatis sebelumnya akan ditimpa dengan default yang disarankan.

Pertimbangan saat memilih kelas tabel

DynamoDB menawarkan dua kelas tabel yang dirancang untuk membantu Anda mengoptimalkan biaya. Kelas tabel Standar DynamoDB adalah defaultnya, dan direkomendasikan untuk sebagian besar beban kerja. Kelas tabel Akses Jarang Standar DynamoDB (DynamoDB Standard-IA) dioptimalkan untuk tabel di mana penyimpanan merupakan biaya dominan. Misalnya, tabel yang menyimpan data yang jarang diakses, seperti log aplikasi, postingan media sosial lama, riwayat pesanan e-niaga, dan pencapaian game sebelumnya, merupakan kandidat yang baik untuk kelas tabel Standard-IA.

Setiap tabel DynamoDB dikaitkan dengan kelas tabel. Semua indeks sekunder yang terkait dengan tabel menggunakan kelas tabel yang sama. Anda dapat mengatur kelas tabel Anda saat membuat tabel Anda (DynamoDB Standard secara default) dan memperbarui kelas tabel tabel yang ada menggunakan AWS Management Console, AWS CLI, atau SDK. AWS DynamoDB juga mendukung pengelolaan kelas tabel Anda AWS CloudFormation menggunakan tabel satu wilayah (tabel yang bukan tabel global). Setiap kelas tabel menawarkan harga yang berbeda untuk penyimpanan data serta permintaan baca dan tulis. Saat memilih kelas tabel untuk tabel Anda, ingatlah hal berikut:

- Kelas tabel DynamoDB Standard menawarkan biaya throughput yang lebih rendah daripada DynamoDB Standard-IA dan merupakan opsi yang paling hemat biaya untuk tabel di mana throughput adalah biaya dominan.
- Kelas tabel DynamoDB Standard-IA menawarkan biaya penyimpanan yang lebih rendah dibandingkan DynamoDB Standard, dan merupakan pilihan paling hemat biaya untuk tabel di mana penyimpanan merupakan biaya dominan. Ketika penyimpanan melebihi 50% biaya throughput (baca dan tulis) tabel yang menggunakan kelas tabel Standar DynamoDB, kelas tabel IA-Standar DynamoDB dapat membantu Anda mengurangi total biaya tabel.
- Tabel DynamoDB Standard-IA menawarkan performa, ketahanan, dan ketersediaan yang sama dengan tabel DynamoDB Standard.
- Beralih antara kelas tabel DynamoDB Standard dan DynamoDB Standard-IA tidak memerlukan perubahan kode aplikasi Anda. Anda menggunakan DynamoDB API dan titik akhir layanan yang sama terlepas dari kelas tabel yang digunakan tabel Anda.
- Tabel DynamoDB Standard-IA kompatibel dengan semua fitur DynamoDB yang ada seperti auto scaling, on-demand mode, (TTL), on-demand backup, recovery (PITR) time-to-live , dan indeks sekunder global. point-in-time

Kelas tabel yang paling hemat biaya untuk tabel Anda bergantung pada penyimpanan yang diharapkan dan pola penggunaan throughput tabel Anda. Anda dapat melihat penyimpanan historis dan biaya throughput dan penggunaan tabel Anda dengan Laporan AWS Biaya dan Penggunaan dan AWS Cost Explorer. Gunakan data historis ini untuk menentukan kelas tabel yang paling hemat biaya untuk tabel Anda. Untuk mempelajari lebih lanjut tentang menggunakan Laporan AWS Biaya dan Penggunaan serta AWS Cost Explorer, lihat Dokumentasi [AWS Billing and Cost Management](#). Lihat Harga [Amazon DynamoDB](#) untuk detail tentang harga kelas tabel.

Note

Pembaruan kelas tabel adalah proses latar belakang. Anda masih dapat mengakses tabel Anda secara normal selama pembaruan kelas tabel. Waktu untuk memperbarui kelas tabel Anda bergantung pada lalu lintas tabel, ukuran penyimpanan, dan variabel terkait lainnya. Tidak lebih dari dua pembaruan kelas tabel di tabel Anda diperbolehkan dalam periode akhir 30 hari.

Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB

Saat Anda membuat tabel baru yang tersedia di Amazon DynamoDB, Anda harus menentukan kapasitas throughput yang tersedia. Ini adalah jumlah aktivitas membaca dan menulis yang dapat didukung oleh tabel. DynamoDB menggunakan informasi ini untuk mencadangkan sumber daya sistem yang memadai guna memenuhi persyaratan throughput Anda.

Note

Anda dapat membuat tabel mode sesuai permintaan sehingga Anda tidak perlu mengelola pengaturan kapasitas apa pun untuk server, penyimpanan, atau throughput. DynamoDB secara instan mengakomodasi beban kerja Anda saat beban kerja tersebut meningkat atau menurun ke tingkat lalu lintas yang telah dicapai sebelumnya. Jika tingkat lalu lintas beban kerja mencapai puncak baru, DynamoDB beradaptasi dengan cepat untuk mengakomodasi beban kerja. Untuk informasi selengkapnya, lihat [Mode sesuai permintaan](#).

Anda juga dapat mengizinkan penskalaan otomatis DynamoDB untuk mengelola kapasitas throughput tabel Anda. Namun, Anda tetap harus memberikan pengaturan awal kapasitas baca dan tulis saat membuat tabel. Penskalaan otomatis DynamoDB menggunakan pengaturan awal ini sebagai titik awal, dan kemudian menyesuaikannya secara dinamis sebagai respons terhadap kebutuhan aplikasi Anda. Untuk informasi selengkapnya, lihat [Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB](#).

Saat data aplikasi dan persyaratan akses Anda berubah, Anda mungkin perlu menyesuaikan pengaturan throughput tabel Anda. Jika Anda menggunakan penskalaan otomatis DynamoDB, pengaturan throughput secara otomatis disesuaikan sebagai respons terhadap beban kerja aktual. Anda juga dapat menggunakan operasi `UpdateTable` untuk menyesuaikan kapasitas throughput tabel Anda secara manual. Anda mungkin memutuskan untuk melakukan ini jika Anda perlu memuat data secara massal dari penyimpanan data yang ada ke tabel DynamoDB baru Anda. Anda dapat membuat tabel dengan pengaturan throughput tulis yang besar, lalu mengurangi pengaturan ini setelah pemuatan data massal selesai.

Anda menentukan persyaratan throughput dalam hal unit kapasitas—jumlah data yang perlu dibaca atau ditulis oleh aplikasi Anda per detik. Anda dapat mengubah pengaturan ini nanti, jika diperlukan, atau mengaktifkan penskalaan otomatis DynamoDB untuk mengubahnya secara otomatis.

Topik

- [Unit kapasitas baca](#)
- [Unit kapasitas tulis](#)
- [Meminta throttling dan kapasitas lonjakan](#)
- [Meminta throttling dan kapasitas adaptif](#)
- [Memilih pengaturan throughput awal](#)
- [Memodifikasi pengaturan throughput](#)
- [Mengatasi masalah throttling](#)
- [Menggunakan CloudWatch metrik untuk menyelidiki pelambatan](#)

Unit kapasitas baca

Unit kapasitas baca mewakili satu bacaan konsisten yang kuat per detik, atau dua bacaan akhir konsisten per detik, untuk sebuah item hingga ukuran 4 KB.

Note

Untuk mempelajari selengkapnya tentang model konsistensi baca DynamoDB, lihat [Konsistensi baca](#).

Misalnya, Anda membuat tabel dengan 10 unit kapasitas baca yang disediakan. Hal ini memungkinkan Anda melakukan 10 bacaan sangat konsisten per detik, atau 20 pembacaan yang pada akhirnya konsisten per detik, untuk item hingga 4 KB.

Membaca item yang lebih besar dari 4 KB menghabiskan lebih banyak unit kapasitas baca. Misalnya, pembacaan item yang sangat konsisten yaitu 8 KB ($4 \text{ KB} \times 2$) menggunakan 2 unit kapasitas baca. Pembacaan yang pada akhirnya konsisten pada item yang sama hanya menggunakan 1 unit kapasitas baca.

Ukuran item untuk pembacaan dibulatkan ke kelipatan 4 KB berikutnya. Misalnya, membaca item berukuran 3.500 byte menggunakan throughput yang sama dengan membaca item berukuran 4 KB.

Konsumsi unit kapasitas untuk bacaan

Berikut ini penjelasan bagaimana operasi baca DynamoDB menggunakan unit kapasitas baca:

- `GetItem`—Membaca satu item dari tabel. Untuk menentukan jumlah unit kapasitas yang akan dikonsumsi `GetItem`, ambil ukuran item dan bulatkan hingga ke batas 4 KB berikutnya. Jika Anda

menentukan bacaan sangat konsisten, ini adalah jumlah unit kapasitas yang diperlukan. Untuk pembacaan yang konsisten (default), bagi angka ini dengan dua.

Misalnya, jika Anda membaca item berukuran 3,5 KB, DynamoDB membulatkan ukuran item menjadi 4 KB. Jika Anda membaca item berukuran 10 KB, DynamoDB membulatkan ukuran item menjadi 12 KB.

- **BatchGetItem**—Membaca hingga 100 item, dari satu tabel atau lebih. DynamoDB memproses setiap item dalam batch sebagai permintaan `GetItem` individu, jadi DynamoDB terlebih dahulu mengumpulkan ukuran setiap item ke batas 4 KB berikutnya, lalu menghitung ukuran totalnya. Hasilnya belum tentu sama dengan total ukuran semua item. Misalnya, jika `BatchGetItem` membaca item sebesar 1,5 KB dan item 6,5 KB, DynamoDB menghitung ukuran sebagai 12 KB (4 KB + 8 KB), bukan 8 KB (1,5 KB + 6,5 KB).
- **Query**—Membaca beberapa item yang memiliki nilai kunci partisi yang sama. Semua item yang dikembalikan diperlakukan sebagai operasi baca tunggal, dimana DynamoDB menghitung ukuran total semua item dan kemudian membulatkannya ke batas 4 KB berikutnya. Misalnya, kueri Anda mengembalikan 10 item dengan ukuran gabungan 40,8 KB. DynamoDB membulatkan ukuran item untuk operasi menjadi 44 KB. Jika kueri mengembalikan 1500 item masing-masing 64 byte, ukuran kumulatifnya adalah 96 KB.
- **Scan**—Membaca semua item dalam tabel. DynamoDB mempertimbangkan ukuran data yang dievaluasi, bukan ukuran data yang dikembalikan oleh pemindaian.

Jika Anda melakukan operasi baca pada item yang tidak ada, DynamoDB masih menggunakan throughput baca yang disediakan: Permintaan baca yang sangat konsisten menggunakan satu unit kapasitas baca, sedangkan permintaan baca yang pada akhirnya konsisten menggunakan 0,5 unit kapasitas baca.

Untuk setiap operasi yang mengembalikan item, Anda dapat meminta subset atribut untuk diambil. Namun, hal ini tidak berdampak pada penghitungan ukuran item. Selain itu, `Query` dan `Scan` dapat mengembalikan jumlah item, bukan nilai atribut. Penghitungan item menggunakan jumlah unit kapasitas baca yang sama dan tunduk pada penghitungan ukuran item yang sama. Hal ini karena DynamoDB harus membaca setiap item untuk menambah jumlah.

Operasi baca dan konsistensi baca

Perhitungan sebelumnya mengasumsikan permintaan baca yang sangat konsisten. Untuk permintaan bacaan akhir konsisten, operasi tersebut hanya menggunakan setengah unit kapasitas.

Untuk bacaan akhir konsisten, jika ukuran item total adalah 80 KB, operasi hanya menggunakan 10 unit kapasitas.

Unit kapasitas tulis

Satu unit kapasitas tulis mewakili satu tulis per detik, untuk item hingga ukuran 1 KB.

Misalnya, Anda membuat tabel dengan 10 unit kapasitas tulis. Hal ini memungkinkan Anda melakukan 10 penulisan per detik, untuk item berukuran hingga 1 KB per detik.

Ukuran item untuk penulisan dibulatkan ke kelipatan 1 KB berikutnya. Misalnya, menulis item berukuran 500 byte memerlukan throughput yang sama dengan menulis item berukuran 1 KB.

Konsumsi unit kapasitas untuk penulisan

Berikut ini penjelasan bagaimana operasi tulis DynamoDB menggunakan unit kapasitas tulis:

- `PutItem`—Menulis satu item pada tabel. Jika item dengan kunci primer yang sama ada dalam tabel, operasi menggantikan item. Untuk menghitung konsumsi throughput yang disediakan, ukuran item yang penting lebih besar dari dua.
- `UpdateItem`—Memodifikasi satu item dalam tabel. DynamoDB menganggap ukuran item, seperti yang muncul sebelum dan setelah pembaruan. Throughput tersedia yang dikonsumsi mencerminkan ukuran item yang lebih besar. Meskipun Anda hanya memperbarui subset dari atribut item, `UpdateItem` masih akan mengonsumsi jumlah penuh throughput yang disediakan (lebih besar dari ukuran item "sebelum" dan "sesudah").
- `DeleteItem`—Menghapus satu item dari tabel. Konsumsi throughput yang disediakan didasarkan pada ukuran item yang dihapus.
- `BatchWriteItem`—Menulis hingga 25 item untuk satu atau beberapa tabel. DynamoDB memproses setiap item dalam batch sebagai permintaan `PutItem` atau `DeleteItem` individu (pembaruan tidak didukung). Jadi DynamoDB terlebih dahulu membulatkan ukuran setiap item ke batas 1 KB berikutnya, kemudian menghitung ukuran total. Hasilnya belum tentu sama dengan total ukuran semua item. Misalnya, jika `BatchWriteItem` menulis item sebesar 500 byte dan item 3,5 KB, DynamoDB menghitung ukurannya sebagai 5 KB (1 KB + 4 KB), bukan 4 KB (500 byte + 3,5 KB).

Untuk operasi `PutItem`, `UpdateItem`, dan `DeleteItem`, DynamoDB membulatkan ukuran item hingga 1 KB berikutnya. Misalnya, jika Anda menempatkan atau menghapus item berukuran 1,6 KB, DynamoDB membulatkan ukuran item menjadi 2 KB.

PutItem, UpdateItem, dan DeleteItem memungkinkan penulisan bersyarat, di mana Anda menentukan ekspresi yang harus bernilai true agar operasi berhasil. Jika ekspresi bernilai salah, DynamoDB masih menggunakan unit kapasitas tulis dari tabel. Jumlah yang dikonsumsi bergantung pada ukuran item (apakah item yang sudah ada di tabel atau item baru yang Anda coba buat atau diperbarui). Misalnya, jika item yang ada berukuran 300kb dan item baru yang Anda coba buat atau diperbarui berukuran 310kb, unit kapasitas tulis yang digunakan akan menjadi item 310kb.

Meminta throttling dan kapasitas lonjakan

Jika aplikasi Anda melakukan operasi baca atau tulis pada laju yang lebih tinggi dari yang bisa didukung tabel, DynamoDB mulai melakukan throttle pada permintaan tersebut. Ketika DynamoDB melakukan throttle operasi baca atau tulis, ia mengembalikan `ProvisionedThroughputExceededException` ke pemanggil. Aplikasi kemudian dapat mengambil tindakan yang sesuai, seperti menunggu beberapa saat sebelum mencoba kembali permintaan tersebut.

Note

Kami menyarankan Anda menggunakan AWS SDK untuk pengembangan perangkat lunak. AWS SDK menyediakan dukungan bawaan untuk mencoba kembali permintaan yang di-throttle; Anda tidak perlu menulis logika ini sendiri. Untuk informasi selengkapnya, lihat [Percobaan ulang kesalahan dan mundur eksponensial](#).

Konsol DynamoDB menampilkan metrik CloudWatch Amazon untuk tabel Anda, sehingga Anda dapat memantau permintaan baca dan permintaan tulis yang dibatasi. Jika Anda mengalami throttling berlebihan, Anda harus mempertimbangkan untuk meningkatkan pengaturan throughput yang disediakan pada tabel Anda.

Dalam beberapa kasus, DynamoDB menggunakan kapasitas lonjakan untuk mengakomodasi operasi baca atau tulis lebih dari pengaturan throughput tabel Anda. Dengan kapasitas lonjakan, permintaan baca atau tulis yang tidak terduga dapat berhasil ketika mereka semestinya di-throttle. Untuk informasi selengkapnya, lihat [Menggunakan kapasitas lonjakan secara efektif](#).

Meminta throttling dan kapasitas adaptif

DynamoDB secara otomatis mendistribusikan data Anda di seluruh partisi, yang disimpan di beberapa server di Cloud (Untuk informasi AWS selengkapnya, lihat). [Partisi dan distribusi data](#) Tidak selalu mungkin untuk mendistribusikan aktivitas baca dan tulis secara merata sepanjang

waktu. Ketika akses data tidak seimbang, partisi "panas" dapat menerima volume lalu lintas baca dan tulis yang lebih tinggi dibandingkan dengan partisi lainnya. Kapasitas adaptif bekerja dengan meningkatkan kapasitas throughput secara otomatis untuk partisi yang menerima lebih banyak lalu lintas. Untuk informasi selengkapnya, lihat [Memahami kapasitas adaptif DynamoDB](#).

Memilih pengaturan throughput awal

Setiap aplikasi memiliki persyaratan yang berbeda untuk membaca dan menulis dari database. Ketika Anda menentukan pengaturan throughput awal untuk tabel DynamoDB, pertimbangkan input berikut:

- Ukuran item. Beberapa item berukuran cukup kecil sehingga bisa dibaca atau ditulis menggunakan satu unit kapasitas. Item yang lebih besar memerlukan beberapa unit kapasitas. Dengan memperkirakan ukuran item yang akan ada di tabel, Anda dapat menentukan pengaturan yang akurat untuk throughput yang disediakan pada tabel Anda.
- Perkiraan laju permintaan baca dan tulis. Selain ukuran item, Anda harus memperkirakan jumlah aktivitas baca dan tulis yang perlu Anda lakukan per detik.
- Persyaratan konsistensi baca. Unit kapasitas baca didasarkan pada operasi baca yang sangat konsisten, yang menggunakan sumber daya database dua kali lebih banyak dibandingkan operasi baca yang konsisten. Anda harus menentukan apakah aplikasi Anda memerlukan bacaan akhir konsisten, atau apakah aplikasi dapat melonggarkan persyaratan ini dan pada akhirnya melakukan bacaan sangat konsisten. (Operasi baca di DynamoDB pada akhirnya konsisten, secara default. Anda dapat meminta bacaan sangat konsisten untuk operasi ini jika diperlukan.)

Misalnya, Anda ingin membaca 80 item per detik dari sebuah tabel. Item berukuran 3 KB, dan Anda ingin bacaan sangat konsisten. Untuk skenario ini, setiap pembacaan memerlukan satu unit kapasitas baca yang disediakan. Untuk menentukan angka ini, Anda membagi ukuran item operasi dengan 4 KB, lalu membulatkannya ke bilangan bulat terdekat, seperti dalam contoh ini:

- $3 \text{ KB} / 4 \text{ KB} = 0,75$, atau 1 unit kapasitas baca

Untuk skenario ini, Anda harus mengatur throughput baca yang disediakan tabel menjadi 80 unit kapasitas baca:

- 1 unit kapasitas baca setiap item \times 80 bacaan per detik = 80 unit kapasitas baca

Sekarang anggaplah Anda ingin menulis 100 item per detik ke tabel Anda, dan item tersebut berukuran 512 byte. Untuk skenario ini, setiap penulisan memerlukan satu unit kapasitas tulis yang

disediakan. Untuk menentukan angka ini, Anda membagi ukuran item operasi dengan 1 KB, lalu membulatkannya ke bilangan bulat terdekat:

- $512 \text{ byte} / 1 \text{ KB} = 0.5$, atau 1

Untuk skenario ini, Anda akan menetapkan throughput penulisan tabel yang disediakan menjadi 100 unit kapasitas tulis:

- $1 \text{ unit kapasitas tulis per item} \times 100 \text{ tulis per detik} = 100 \text{ unit kapasitas tulis}$

Note

Untuk rekomendasi tentang throughput yang disediakan dan topik terkait, lihat [Praktik Terbaik untuk merancang dan menggunakan kunci partisi secara efektif](#).

Memodifikasi pengaturan throughput

Jika Anda telah mengaktifkan penskalaan otomatis DynamoDB untuk sebuah tabel, maka kapasitas throughputnya disesuaikan secara dinamis sebagai respons terhadap penggunaan sebenarnya. Tidak diperlukan intervensi manual.

Anda dapat memodifikasi pengaturan throughput yang disediakan tabel menggunakan AWS Management Console atau operasi `UpdateTable`. Untuk informasi selengkapnya tentang peningkatan dan penurunan throughput per hari, lihat [Layanan, akun, dan tabel kuota di Amazon DynamoDB](#).

Mengatasi masalah throttling

Untuk pemecahan masalah yang tampaknya terkait dengan pembatasan, hal pertama yang penting adalah mengonfirmasi apakah pembatasan tersebut berasal dari DynamoDB atau dari aplikasi.

Berikut adalah beberapa skenario umum, dan langkah-langkah yang mungkin dilakukan untuk membantu mengatasinya.

Tabel DynamoDB tampaknya memiliki kapasitas yang disediakan cukup, tetapi permintaan dibatasi.

Hal ini dapat terjadi ketika throughput berada di bawah rata-rata per menit, namun throughput melebihi jumlah yang tersedia per detik. DynamoDB hanya melaporkan metrik level menit

CloudWatch ke, yang kemudian dihitung sebagai jumlah selama satu menit dan dirata-ratakan. Namun DynamoDB sendiri menerapkan batasan kecepatan per detik. Jadi, jika terlalu banyak throughput yang terjadi dalam sebagian kecil menit tersebut, misalnya beberapa detik atau kurang, maka permintaan untuk sisa menit tersebut dapat dibatasi. Misalnya, jika kita telah menyediakan 60 WCU pada sebuah tabel, tabel tersebut dapat melakukan 3600 penulisan dalam 1 menit. Namun jika seluruh 3600 permintaan WCU tercapai dalam detik yang sama, maka sisa menit tersebut akan di-throttle.

Salah satu cara untuk mengatasi skenario ini adalah dengan menambahkan beberapa kegugupan dan kemunduran eksponensial ke panggilan API. Untuk informasi lebih lanjut lihat posting ini tentang [backoff dan jitter](#).

Penskalaan otomatis diaktifkan, tetapi tabel masih dibatasi.

Hal ini dapat terjadi ketika terjadi lonjakan lalu lintas secara tiba-tiba. Penskalaan otomatis dapat dipicu ketika dua titik data melanggar nilai pemanfaatan target yang dikonfigurasi dalam rentang satu menit. Oleh karena itu, penskalaan otomatis dapat terjadi karena kapasitas yang dikonsumsi berada di atas target pemanfaatan selama dua menit yang konsisten. Tetapi jika lonjakan terpisah lebih dari satu menit, penskalaan otomatis mungkin tidak dipicu. Demikian pula, peristiwa penurunan skala dapat dipicu ketika 15 titik data berturut-turut lebih rendah dari pemanfaatan target. Dalam kedua kasus tersebut, setelah penskalaan otomatis dipicu, [UpdateTable](#) panggilan dipanggil. Kemudian dapat memakan waktu beberapa menit untuk memperbarui kapasitas yang disediakan untuk tabel atau indeks. Selama periode ini, setiap permintaan yang melebihi kapasitas tabel yang disediakan sebelumnya akan dibatasi.

Singkatnya, Penskalaan Otomatis memerlukan titik data berurutan di mana nilai pemanfaatan target dilanggar untuk meningkatkan skala tabel DynamoDB. Karena alasan inilah Penskalaan Otomatis tidak disarankan sebagai solusi untuk menangani beban kerja yang melonjak. Silakan merujuk ke [dokumentasi Penskalaan Otomatis](#) untuk informasi selengkapnya.

Tabel berada dalam mode kapasitas "Sesuai Permintaan" tetapi masih dibatasi.

Untuk tabel Sesuai Permintaan, DynamoDB secara otomatis mengalokasikan lebih banyak kapasitas seiring peningkatan volume lalu lintas Anda. Selama akses didistribusikan secara merata ke seluruh partisi, dan tabel tidak melebihi dua kali lipat lalu lintas puncak sebelumnya, keseluruhan tabel tidak akan dibatasi. Namun, pembatasan masih dapat terjadi jika throughput melebihi dua kali lipat puncak sebelumnya dalam waktu 30 menit yang sama. Silakan merujuk ke [OnDemand penskalaan](#) untuk informasi lebih lanjut.

Hot key mungkin menyebabkan masalah pelambatan.

Di DynamoDB, kunci partisi yang tidak memiliki kardinalitas tinggi dapat mengakibatkan banyak permintaan yang menargetkan beberapa partisi saja. Jika "partisi panas" yang dihasilkan melampaui batas per partisi sebesar 3000 RCU atau 1000 WCU per detik, hal ini dapat mengakibatkan throttling. Alat diagnostik CloudWatch Contributor Insights, dapat membantu men-debug ini, dengan menyediakan grafik CCI untuk setiap pola akses item tabel. Anda dapat terus memantau kunci tabel DynamoDB yang paling sering diakses dan tren lalu lintas lainnya. [Untuk mengetahui lebih lanjut tentang CloudWatch Contributor Insights, lihat CCI.](#) Untuk informasi spesifik tentang memilih hot key yang tepat, lihat [Memilih hot key yang tepat.](#)

Menggunakan CloudWatch metrik untuk menyelidiki pelambatan

Di bawah ini adalah beberapa metrik DynamoDB yang harus dipantau selama peristiwa throttling, untuk membantu menemukan operasi mana yang membuat permintaan yang di-throttle dan mengidentifikasi akar masalah.

1. ThrottledRequests

- Satu permintaan yang di-throttle dapat berisi beberapa peristiwa yang di-throttle, sehingga peristiwa dapat lebih relevan untuk diperiksa.

Misalnya, saat Anda memperbarui item dalam tabel dengan GSI, akan terjadi beberapa peristiwa - penulisan ke tabel, dan penulisan ke setiap indeks. Bahkan jika satu atau lebih dari peristiwa ini dibatasi, hanya akan ada satu. ThrottledRequest

2. ReadThrottleEvents

- Perhatikan permintaan yang melebihi RCU yang disediakan untuk tabel atau GSI.

3. WriteThrottleEvents

- Perhatikan permintaan yang melebihi WCU yang disediakan untuk sebuah tabel atau GSI.

4. OnlineIndexConsumedWriteCapacity

- Perhatikan jumlah WCU yang dikonsumsi saat menambahkan GSI baru ke tabel. Perhatikan bahwa ConsumedWriteCapacityUnits untuk GSI tidak menyertakan WCU yang digunakan selama pembuatan indeks.

- Jika Anda menyetel WCU untuk GSI terlalu rendah, maka aktivitas penulisan yang masuk selama fase pengisian ulang mungkin akan di-throttle.

5. OnlineIndexThrottleEvents

- Tinjau jumlah peristiwa pembatasan penulisan yang terjadi saat menambahkan GSI baru ke tabel.
- Jika ternyata WCU disetel terlalu rendah dan di-throttle, Anda dapat memperbarui nilai WCU untuk GSI bahkan selama pengisian ulang.

6. Provisioned Read/Write

- Lihat berapa banyak unit kapasitas baca atau tulis yang disediakan yang digunakan selama jangka waktu tertentu, untuk tabel atau indeks sekunder global tertentu.
- Perhatikan bahwa dimensi TableName mengembalikan ProvisionedReadCapacityUnits untuk tabel hanya secara default. Guna melihat jumlah unit kapasitas baca atau tulis yang disediakan untuk indeks sekunder global, Anda harus menentukan kedua TableName dan GlobalSecondaryIndexName.

7. Consumed Read/Write

- Lihat berapa banyak unit kapasitas baca atau tulis yang dikonsumsi selama jangka waktu tertentu.

[Untuk informasi selengkapnya tentang DynamoDB Metrics, CloudWatch lihat Metrik dan Dimensi.](#)

Ukuran dan format item DynamoDB

Tabel DynamoDB tidak memiliki skema, kecuali kunci primer, sehingga semua item dalam tabel dapat memiliki atribut, ukuran, dan tipe data yang berbeda.

Ukuran total suatu item adalah jumlah panjang nama dan nilai atributnya, ditambah overhead yang berlaku seperti dijelaskan di bawah. Anda dapat menggunakan panduan berikut untuk memperkirakan ukuran atribut:

- String adalah Unicode dengan pengodean biner UTF-8. Ukuran string adalah (jumlah byte UTF-8 yang dikodekan dari nama atribut) + (jumlah byte yang dikodekan UTF-8).

- Angka memiliki panjang yang bervariasi, hingga 38 angka penting. Angka nol di depan dan di belakang dipangkas. Ukuran angka kira-kira (jumlah byte yang dikodekan UTF-8 dari nama atribut) + (1 byte per dua digit signifikan) + (1 byte).
- Nilai biner harus dikodekan dalam format base64 sebelum dapat dikirim ke DynamoDB, namun panjang byte mentah nilai tersebut digunakan untuk menghitung ukuran. Ukuran atribut biner adalah (jumlah byte UTF-8 yang dikodekan dari nama atribut) + (jumlah byte mentah).
- Ukuran atribut null atau atribut Boolean adalah (jumlah byte UTF-8 yang dikodekan dari nama atribut) + (1 byte).
- Atribut jenis List atau Map memerlukan overhead 3 byte, apa pun kontennya. Ukuran a List or Map is (jumlah byte UTF-8 yang dikodekan dari nama atribut) + jumlah (ukuran elemen bersarang) + (3 byte). Ukuran kosong List atau Map is (jumlah byte UTF-8 yang dikodekan dari nama atribut) + (3 byte).
- Masing-masing elemen List atau Map juga membutuhkan 1 byte overhead.

Note

Sebaiknya Anda memilih nama atribut yang lebih pendek daripada nama lama. Hal ini membantu Anda mengurangi jumlah penyimpanan yang diperlukan, namun juga dapat menurunkan jumlah RCU/WCU yang Anda gunakan.

Untuk tujuan penagihan penyimpanan, setiap item mencakup overhead penyimpanan per item yang bergantung pada fitur yang telah Anda aktifkan.

- Semua item di DynamoDB memerlukan overhead penyimpanan sebesar 100 byte untuk pengindeksan.
- Beberapa fitur DynamoDB (tabel global, transaksi, perubahan pengambilan data untuk Kinesis Data Streams dengan DynamoDB) memerlukan overhead penyimpanan tambahan untuk memperhitungkan atribut yang dibuat sistem akibat pengaktifan fitur tersebut. Misalnya, tabel global memerlukan tambahan overhead penyimpanan sebesar 48 byte.

Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB

Banyak beban kerja basis data yang bersifat siklik, sementara yang lainnya sulit diprediksi sebelumnya. Sebagai salah satu contoh, perhatikan aplikasi jejaring sosial yang sebagian besar penggunanya aktif pada siang hari. Basis data harus mampu menangani aktivitas siang hari, tetapi tingkat throughput yang sama tidak diperlukan pada malam hari. Sebagai contoh lainnya, pertimbangkan aplikasi game seluler baru yang tidak disangka mengalami adopsi yang begitu cepat. Jika game menjadi terlalu populer, game tersebut bisa melampaui sumber daya basis data yang tersedia, sehingga performanya lambat dan pelanggan menjadi tidak puas. Beban kerja semacam ini kerap perlu dinaikkan atau dikurangi skala sumber daya basis datanya secara manual guna merespons berbagai tingkat penggunaan.

Penskalaan otomatis Amazon DynamoDB menggunakan layanan Application AWS Auto Scaling untuk secara dinamis menyesuaikan kapasitas throughput yang disediakan atas nama Anda, sebagai respons terhadap pola lalu lintas yang sebenarnya. Layanan ini memungkinkan tabel atau indeks sekunder global meningkatkan kapasitas baca dan tulis tersedia untuk menangani peningkatan lalu lintas yang mendadak, tanpa throttling. Ketika beban kerja berkurang, Application Auto Scaling dapat menurunkan throughput agar Anda tidak membayar kapasitas tersedia yang tidak terpakai.

Note

Jika Anda menggunakan AWS Management Console untuk membuat tabel atau indeks sekunder global, DynamoDB auto scaling diaktifkan secara default. Anda dapat memodifikasi pengaturan penskalaan otomatis kapan saja. Untuk informasi selengkapnya, lihat [Menggunakan AWS Management Console dengan penskalaan otomatis DynamoDB](#). Saat Anda menghapus tabel atau replika tabel global, target yang dapat diskalakan, kebijakan penskalaan, atau CloudWatch alarm yang terkait tidak akan dihapus secara otomatis dengannya.

Dengan Application Auto Scaling, Anda membuat kebijakan penskalaan untuk tabel atau indeks sekunder global. Kebijakan penskalaan menentukan apakah Anda ingin menskalakan kapasitas baca atau tulis (atau keduanya), serta pengaturan unit kapasitas minimum dan maksimum yang tersedia untuk tabel atau indeks.

Kebijakan penskalaan juga berisi pemanfaatan target—persentase throughput tersedia yang dikonsumsi pada suatu titik waktu. Application Auto Scaling menggunakan algoritme pelacakan target

untuk menyesuaikan throughput tersedia dari tabel (atau indeks) ke atas atau ke bawah sebagai respons atas beban kerja aktual, sehingga pemanfaatan kapasitas aktual tetap atau mendekati pemanfaatan target Anda.

Penskalaan otomatis dapat dipicu ketika dua titik data melanggar nilai pemanfaatan target yang dikonfigurasi dalam rentang satu menit. Oleh karena itu, penskalaan otomatis dapat terjadi karena kapasitas yang dikonsumsi berada di atas target pemanfaatan selama dua menit yang konsisten. Tetapi jika paku terpisah lebih dari satu menit, penskalaan otomatis mungkin tidak dipicu. Demikian pula, peristiwa penurunan skala dapat dipicu ketika 15 titik data berturut-turut lebih rendah dari pemanfaatan target. Dalam kedua kasus tersebut, setelah penskalaan otomatis dipicu, [UpdateTable](#) panggilan dipanggil. Kemudian dapat memakan waktu beberapa menit untuk memperbarui kapasitas yang disediakan untuk tabel atau indeks. Selama periode ini, setiap permintaan yang melebihi kapasitas tabel yang disediakan sebelumnya akan dibatasi.

Important

Anda tidak dapat menyesuaikan jumlah titik data yang akan dilanggar untuk memicu alarm yang mendasarinya (meskipun nomor saat ini dapat berubah di masa mendatang).

Anda dapat menetapkan nilai pemanfaatan target penskalaan otomatis antara 20 dan 90 persen untuk kapasitas baca dan tulis Anda.

Note

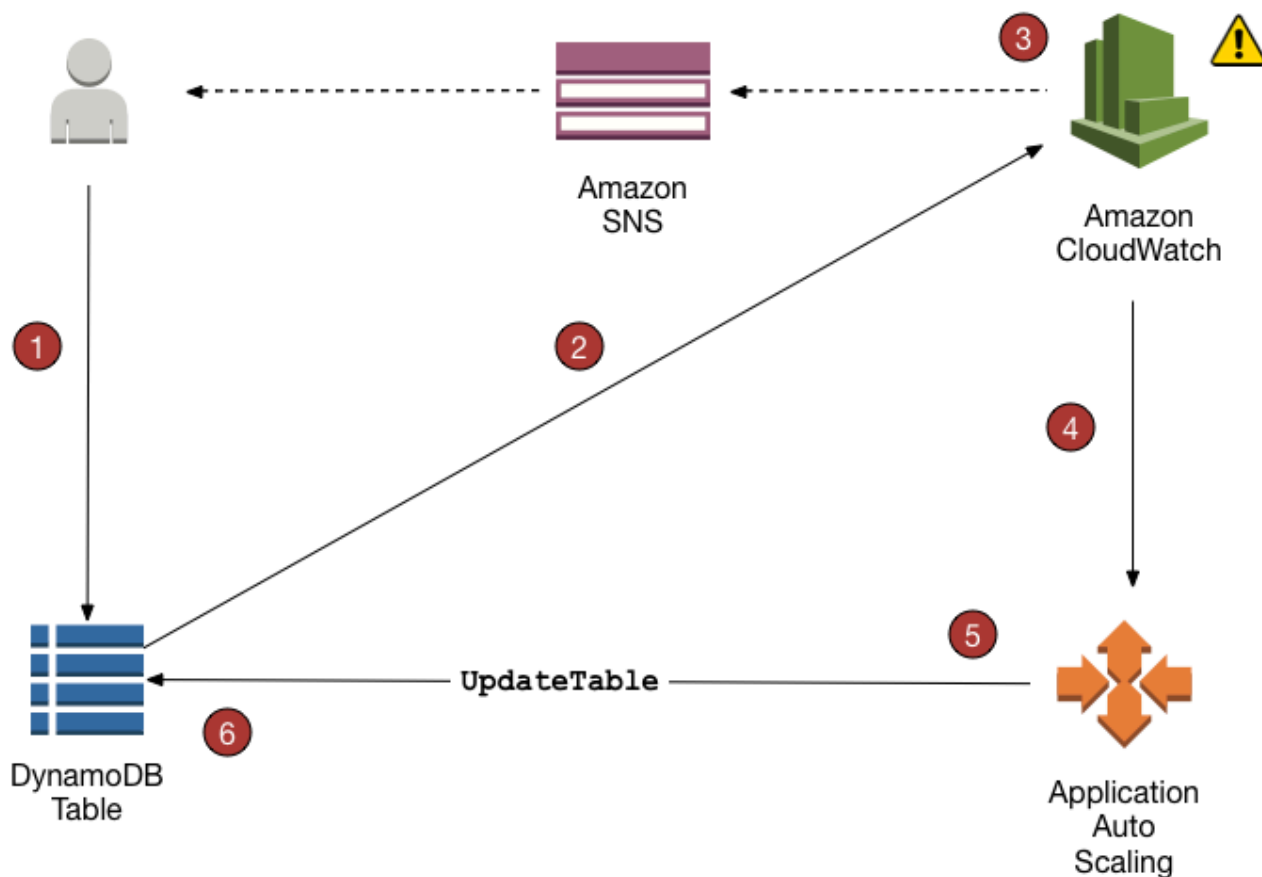
Selain tabel, penskalaan otomatis DynamoDB juga mendukung indeks sekunder global. Setiap indeks sekunder global memiliki kapasitas throughput tersedia miliknya sendiri, terpisah dari tabel dasarnya. Ketika Anda membuat kebijakan penskalaan untuk indeks sekunder global, Application Auto Scaling menyesuaikan pengaturan throughput tersedia untuk indeks guna memastikan bahwa pemanfaatan aktual tetap atau mendekati rasio pemanfaatan yang Anda inginkan.

Cara kerja penskalaan otomatis DynamoDB

Note

Untuk memulai penggunaan penskalaan otomatis DynamoDB dengan cepat, lihat [Menggunakan AWS Management Console dengan penskalaan otomatis DynamoDB](#).

Diagram berikut memberikan ikhtisar secara umum tentang bagaimana penskalaan otomatis DynamoDB mengelola kapasitas throughput untuk tabel.



Langkah-langkah berikut meringkas proses penskalaan otomatis seperti yang ditunjukkan pada diagram sebelumnya:

1. Anda membuat kebijakan Application Auto Scaling untuk tabel DynamoDB Anda.
2. DynamoDB menerbitkan metrik kapasitas yang dikonsumsi ke Amazon. CloudWatch

3. Jika kapasitas konsumsi tabel melebihi pemanfaatan target Anda (atau jatuh di bawah target) untuk jangka waktu tertentu, Amazon CloudWatch memicu alarm. Anda dapat melihat alarm di konsol dan menerima pemberitahuan menggunakan Amazon Simple Notification Service (Amazon SNS).
4. CloudWatch Alarm memanggil Application Auto Scaling untuk mengevaluasi kebijakan penskalaan Anda.
5. Application Auto Scaling mengeluarkan permintaan `UpdateTable` untuk menyesuaikan throughput tersedia pada tabel Anda.
6. DynamoDB memproses permintaan `UpdateTable`, secara dinamis menaikkan (atau menurunkan) kapasitas throughput tersedia pada tabel agar mendekati pemanfaatan target Anda.

Untuk memahami cara kerja penskalaan otomatis DynamoDB, misalnya Anda memiliki tabel bernama `ProductCatalog`. Tabel ini dimuat secara massal dengan data secara jarang, sehingga tidak mengalami aktivitas tulis yang sangat banyak. Namun, tabel tersebut memang mengalami aktivitas baca yang tinggi dan bervariasi dari waktu ke waktu. Dengan memantau CloudWatch metrik `AmazonProductCatalog`, Anda menentukan bahwa tabel memerlukan 1.200 unit kapasitas baca (untuk menghindari DynamoDB membatasi permintaan baca saat aktivitas mencapai puncaknya). Anda juga menentukan bahwa `ProductCatalog` memerlukan minimum 150 unit kapasitas baca, ketika lalu lintas baca berada pada titik terendah. Untuk informasi selengkapnya tentang mencegah throttling, lihat [Memecahkan masalah pembatasan di Amazon DynamoDB](#).

Dalam kisaran 150 sampai 1.200 unit kapasitas baca, Anda memutuskan bahwa pemanfaatan target sebesar 70 persen akan sesuai untuk tabel `ProductCatalog`. Pemanfaatan target adalah rasio unit kapasitas yang dikonsumsi terhadap unit kapasitas yang disediakan, dinyatakan dengan persentase. Application Auto Scaling menggunakan algoritme pelacakan targetnya untuk memastikan bahwa kapasitas baca yang disediakan dari `ProductCatalog` disesuaikan sesuai kebutuhan sehingga pemanfaatan tetap atau mendekati 70 persen.

Note

Penskalaan otomatis DynamoDB memodifikasi pengaturan throughput yang disediakan hanya ketika beban kerja sebenarnya tetap dinaikkan atau ditekan untuk jangka waktu berkelanjutan selama beberapa menit. Algoritme pelacakan target Application Auto Scaling berusaha menjaga pemanfaatan target ini pada atau mendekati nilai yang Anda pilih dalam jangka panjang.

Lonjakan aktivitas yang mendadak dan berdurasi singkat diakomodasi oleh kapasitas lonjakan bawaan tabel. Untuk informasi selengkapnya, lihat [Menggunakan kapasitas lonjakan secara efektif](#).

Untuk mengaktifkan penskalaan otomatis DynamoDB untuk tabel `ProductCatalog`, Anda membuat kebijakan penskalaan. Kebijakan ini menentukan hal berikut:

- Tabel atau indeks sekunder global yang ingin Anda kelola
- Jenis kapasitas mana yang akan dikelola (kapasitas baca atau kapasitas tulis)
- Batas atas dan bawah untuk pengaturan throughput yang tersedia
- Pemanfaatan target Anda

Saat Anda membuat kebijakan penskalaan, Application Auto Scaling akan membuat sepasang alarm CloudWatch Amazon atas nama Anda. Setiap pasangan merepresentasikan batas atas dan bawah untuk pengaturan throughput tersedia Anda. CloudWatch Alarm ini dipicu ketika penggunaan tabel yang sebenarnya menyimpang dari penggunaan target Anda untuk jangka waktu yang berkelanjutan.

Ketika salah satu CloudWatch alarm dipicu, Amazon SNS mengirim Anda pemberitahuan (jika Anda telah mengaktifkannya). CloudWatch Alarm kemudian memanggil Application Auto Scaling, yang pada gilirannya memberi tahu DynamoDB untuk menyesuaikan kapasitas yang disediakan tabel ke atas atau ke bawah `ProductCatalog` sebagaimana mestinya.

Selama acara penskalaan, AWS Config dibebankan per item konfigurasi yang direkam. Ketika peristiwa penskalaan terjadi, empat CloudWatch alarm dibuat untuk setiap peristiwa auto-scaling baca dan tulis: `alarm:`, dan `ProvisionedCapacity alarm:.`, `ProvisionedCapacityLow` `ProvisionedCapacityHigh` `ConsumedCapacity AlarmHigh` `AlarmLow` Ini menghasilkan total delapan alarm. Oleh karena itu, AWS Config catat delapan item konfigurasi untuk setiap peristiwa penskalaan.

Note

Anda juga dapat menjadwalkan penskalaan DynamoDB Anda sehingga terjadi pada waktu-waktu tertentu. Pelajari langkah-langkah dasar [di sini](#).

Catatan penggunaan

Sebelum Anda mulai menggunakan penskalaan otomatis DynamoDB, ketahui hal berikut:

- Penskalaan otomatis DynamoDB dapat meningkatkan kapasitas baca atau kapasitas tulis sesering yang diperlukan, sesuai dengan kebijakan penskalaan otomatis Anda. Semua kuota DynamoDB tetap berlaku, seperti yang dijelaskan dalam [Layanan, akun, dan tabel kuota di Amazon DynamoDB](#).
- Penskalaan otomatis DynamoDB tidak mencegah Anda untuk memodifikasi sendiri pengaturan throughput yang tersedia. Penyesuaian manual ini tidak memengaruhi CloudWatch alarm yang ada yang terkait dengan penskalaan otomatis DynamoDB.
- Jika Anda mengaktifkan penskalaan otomatis DynamoDB untuk tabel yang memiliki satu atau beberapa indeks global sekunder, kami sangat menyarankan Anda agar juga menerapkan penskalaan otomatis yang seragam untuk indeks tersebut. Hal ini akan membantu memastikan performa yang lebih baik untuk penulisan dan pembacaan tabel, serta membantu menghindari throttling. Anda dapat mengaktifkan penskalaan otomatis dengan memilih Terapkan pengaturan yang sama untuk indeks sekunder global di AWS Management Console. Untuk informasi selengkapnya, lihat [Mengaktifkan penskalaan otomatis DynamoDB pada tabel yang ada](#).
- Saat Anda menghapus tabel atau replika tabel global, target yang dapat diskalakan terkait, kebijakan penskalaan, atau CloudWatch alarm tidak akan dihapus secara otomatis dengannya.
- Saat membuat GSI untuk tabel yang ada, penskalaan otomatis tidak diaktifkan untuk GSI. Anda harus mengelola kapasitas secara manual saat GSI sedang dibangun. Setelah backfill pada GSI selesai dan mencapai status aktif, penskalaan otomatis akan beroperasi seperti biasa.

Menggunakan AWS Management Console dengan penskalaan otomatis DynamoDB

Ketika Anda menggunakan AWS Management Console untuk membuat tabel baru, penskalaan otomatis Amazon DynamoDB diaktifkan untuk tabel tersebut secara default. Anda juga dapat menggunakan konsol untuk mengaktifkan penskalaan otomatis untuk tabel yang ada, memodifikasi pengaturan penskalaan otomatis, atau menonaktifkan penskalaan otomatis.

Note


Untuk fitur lanjutan seperti mengatur penskalaan naik dan turun waktu pendinginan, gunakan AWS Command Line Interface (AWS CLI) untuk mengelola penskalaan otomatis DynamoDB. Untuk informasi selengkapnya, lihat [Menggunakan AWS CLI untuk mengelola penskalaan otomatis DynamoDB](#).

Topik

- [Sebelum memulai: Memberikan izin pengguna untuk penskalaan otomatis DynamoDB](#)
- [Membuat tabel baru dengan penskalaan otomatis diaktifkan](#)
- [Mengaktifkan penskalaan otomatis DynamoDB pada tabel yang ada](#)
- [Melihat aktivitas penskalaan otomatis di konsol](#)
- [Memodifikasi atau menonaktifkan pengaturan penskalaan otomatis DynamoDB](#)

Sebelum memulai: Memberikan izin pengguna untuk penskalaan otomatis DynamoDB


Dalam AWS Identity and Access Management (IAM), AWS kebijakan terkelola `DynamoDBFullAccess` menyediakan izin yang diperlukan untuk menggunakan konsol DynamoDB. Namun, untuk penskalaan otomatis DynamoDB, pengguna memerlukan izin tambahan.

 Important

Untuk menghapus tabel berkemampuan skala otomatis, `application-autoscaling:*` izin diperlukan. Yang AWS kebijakan terkelola `DynamoDBFullAccess` termasuk izin tersebut.

Untuk menyiapkan pengguna akses konsol DynamoDB dan penskalaan otomatis DynamoDB, buat peran dan tambahkan `AmazonDynamoDBFullAccess` kebijakan untuk peran itu. Kemudian tetapkan peran ke pengguna.

Membuat tabel baru dengan penskalaan otomatis diaktifkan

 Note

Penskalaan otomatis DynamoDB memerlukan keberadaan peran terkait layanan (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) yang melakukan tindakan penskalaan otomatis atas nama Anda. Peran ini dibuat secara otomatis untuk Anda. Untuk informasi selengkapnya, lihat [Peran yang ditautkan dengan layanan untuk Application Auto Scaling](#) dalam Panduan Pengguna Application Auto Scaling.


Untuk membuat tabel baru dengan penskalaan otomatis diaktifkan

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Pilih Buat tabel.

3. Pada **Buat tabel** halaman, masukkan **Nama tabel** dan kunci utama.
4. Jika **Pengaturan default** dipilih, tabel akan dibuat dengan auto scaling diaktifkan.

Jika tidak, untuk pengaturan khusus:


- a. Pilih **Sesuaikan** pengaturan.
- b. Dalam **Baca/tulis** pengaturan kapasitas bagian, pilih **Disediakan** modus kapasitas dan **set Penskalaan otomatis** kepada **Pada** untuk **Kapasitas baca**, **Kapasitas tulis**, atau keduanya. Untuk masing-masing, atur kebijakan penskalaan yang Anda inginkan untuk tabel dan, secara opsional, semua indeks sekunder global tabel.
 - **Unit kapasitas minimum**- Masukkan batas bawah Anda untuk rentang penskalaan otomatis.
 - **Unit kapasitas maksimum**- Masukkan batas atas Anda untuk rentang penskalaan otomatis.
 - **Pemanfaatan target**- Masukkan persentase pemanfaatan target Anda untuk tabel.

 **Note**

Jika Anda membuat indeks sekunder global untuk tabel baru, kapasitas indeks pada saat pembuatan akan sama dengan kapasitas tabel dasar Anda. Anda dapat mengubah kapasitas indeks dalam pengaturan tabel setelah Anda membuat tabel.

5. Ketika pengaturan seperti yang Anda inginkan, pilih **Buat tabel**. Tabel Anda dibuat dengan parameter penskalaan otomatis.


Mengaktifkan penskalaan otomatis DynamoDB pada tabel yang ada

 **Note**

Penskalaan otomatis DynamoDB memerlukan keberadaan peran terkait layanan (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) yang melakukan tindakan penskalaan otomatis atas nama Anda. Peran ini dibuat secara otomatis untuk Anda. Untuk informasi lebih lanjut, lihat [Peran terkait layanan untuk Penskalaan Otomatis Aplikasi](#).

Untuk mengaktifkan penskalaan otomatis DynamoDB untuk tabel yang ada

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Dalam panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tabel yang ingin Anda kerjakan dan pilih Pengaturan tambah tab.
4. Dalam Kapasitas baca/tulis bagian, pilih Mengedit.
5. Dalam Mode kapasitas bagian, pilih Disediakan.
6. Dalam Kapasitas mejabagian, set Penskalaan otomatis kepada Pada untuk Kapasitas baca, Kapasitas tulis, atau keduanya. Untuk masing-masing, atur kebijakan penskalaan yang Anda inginkan untuk tabel dan, secara opsional, semua indeks sekunder global tabel.
 - Unit kapasitas minimum- Masukkan batas bawah Anda untuk rentang penskalaan otomatis.
 - Unit kapasitas maksimum- Masukkan batas atas Anda untuk rentang penskalaan otomatis.
 - Pemanfaatan target- Masukkan persentase pemanfaatan target Anda untuk tabel.
 - Gunakan pengaturan kapasitas baca/tulis kapasitas yang sama untuk semua indeks sekunder global- Pilih apakah indeks sekunder global harus menggunakan kebijakan penskalaan otomatis yang sama dengan tabel dasar.

 Note

Untuk kinerja terbaik, kami sarankan Anda mengaktifkan Gunakan pengaturan kapasitas baca/tulis yang sama untuk semua indeks sekunder global. Opsi ini memungkinkan penskalaan otomatis DynamoDB untuk menskalakan semua indeks sekunder global pada tabel dasar secara seragam. Ini termasuk indeks sekunder global yang ada, dan setiap indeks lain yang Anda buat untuk tabel ini di masa mendatang.

Dengan mengaktifkan opsi ini, Anda tidak dapat menetapkan kebijakan penskalaan pada indeks sekunder global individual.

7. Saat pengaturan sesuai keinginan Anda, pilih Simpan.

Melihat aktivitas penskalaan otomatis di konsol

Saat drive aplikasi Anda membaca dan menulis lalu lintas ke tabel, penskalaan otomatis DynamoDB secara dinamis memodifikasi pengaturan throughput tabel. Amazon CloudWatch melacak kapasitas

yang disediakan dan dikonsumsi, peristiwa yang dibatasi, latensi, dan metrik lainnya untuk semua tabel DynamoDB dan indeks sekunder Anda.

Untuk melihat metrik ini di konsol DynamoDB, pilih tabel yang ingin Anda gunakan dan pilih **Memantau tab**. Untuk membuat tampilan metrik tabel yang dapat disesuaikan, pilih **Lihat semua di CloudWatch**.

Untuk informasi lebih lanjut tentang **CloudWatch pemantauan** di DynamoDB, lihat [Pemantauan CloudWatch dengan Amazon](#).

Memodifikasi atau menonaktifkan pengaturan penskalaan otomatis DynamoDB

Anda dapat menggunakan AWS Management Console untuk mengubah pengaturan penskalaan otomatis DynamoDB Anda. Untuk melakukan ini, pergi ke **Pengaturan tambah tab** untuk meja Anda, dan pilih **Mengedit** di dalam **Kapasitas baca/tulis bagian**. Untuk informasi selengkapnya tentang pengaturan ini, lihat [Mengaktifkan penskalaan otomatis DynamoDB pada tabel yang ada](#).

Menggunakan AWS CLI untuk mengelola penskalaan otomatis DynamoDB

Alih-alih menggunakan AWS Management Console, Anda dapat menggunakan AWS Command Line Interface (AWS CLI) untuk mengelola penskalaan otomatis Amazon DynamoDB. Tutorial di bagian ini menunjukkan cara menginstal dan mengonfigurasi AWS CLI untuk mengelola penskalaan otomatis DynamoDB. Dalam tutorial ini, Anda melakukan hal-hal berikut:

- Buat tabel DynamoDB bernama `TestTable`. Pengaturan throughput awal adalah 5 unit kapasitas baca dan 5 unit kapasitas tulis.
- Buat kebijakan **Application Auto Scaling** untuk `TestTable`. Kebijakan ini bertujuan untuk mempertahankan rasio target sebesar 50 persen antara kapasitas tulis yang dikonsumsi dan kapasitas tulis yang disediakan. Kisaran untuk metrik ini adalah antara 5 dan 10 unit kapasitas tulis. (**Application Auto Scaling** tidak diizinkan untuk menyesuaikan throughput di luar kisaran ini.)
- Jalankan program Python untuk mendorong lalu lintas tulis ke `TestTable`. Ketika rasio target melebihi 50 persen untuk jangka waktu yang berkelanjutan, **Application Auto Scaling** memberi tahu DynamoDB untuk menyesuaikan throughput `TestTable` ke atas guna mempertahankan pemanfaatan target sebesar 50 persen.
- Verifikasi bahwa DynamoDB telah berhasil menyesuaikan kapasitas tulis yang disediakan untuk `TestTable`.

Note

Anda juga dapat menjadwalkan penskalaan DynamoDB Anda sehingga terjadi pada waktu-waktu tertentu. Pelajari langkah-langkah dasar [di sini](#).

Topik

- [Sebelum Anda memulai](#)
- [Langkah 1: Buat tabel DynamoDB](#)
- [Langkah 2: Daftarkan target yang dapat diskalakan](#)
- [Langkah 3: Buat kebijakan penskalaan](#)
- [Langkah 4: Dorong lalu lintas tulis ke TestTable](#)
- [Langkah 5: Lihat tindakan Application Auto Scaling](#)
- [\(Opsional\) Langkah 6: Hapus](#)

Sebelum Anda memulai

Selesaikan tugas berikut sebelum memulai tutorial.

Instal AWS CLI

Jika belum, Anda harus menginstal dan mengonfigurasi AWS CLI. Untuk melakukannya, ikuti petunjuk berikut di Panduan Pengguna AWS Command Line Interface:

- [Menginstal AWS CLI](#)
- [Mengonfigurasi AWS CLI](#)

Instal Python

Anda perlu menjalankan program Python di bagian tutorial ini (lihat [Langkah 4: Dorong lalu lintas tulis ke TestTable](#)). Jika belum menginstalnya, Anda dapat [mengunduh Python](#).

Langkah 1: Buat tabel DynamoDB

Pada langkah ini, Anda menggunakan AWS CLI untuk membuat TestTable. Kunci primer terdiri dari pk (kunci partisi) dan sk (kunci urutan). Kedua atribut ini berjenis Number. Pengaturan throughput awal adalah 5 unit kapasitas baca dan 5 unit kapasitas tulis.

1. Gunakan perintah AWS CLI berikut untuk membuat tabel.

```
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. Untuk memeriksa status tabel, gunakan perintah berikut.

```
aws dynamodb describe-table \  
  --table-name TestTable \  
  --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

Tabel siap digunakan saat statusnya ACTIVE.

Langkah 2: Daftarkan target yang dapat diskalakan

Selanjutnya Anda mendaftarkan kapasitas tulis tabel sebagai target yang dapat diskalakan dengan Application Auto Scaling. Hal ini memungkinkan Application Auto Scaling untuk menyesuaikan kapasitas tulis yang disediakan untuk TestTable, tetapi hanya dalam kisaran 5-10 unit kapasitas.

Note

Penskalaan otomatis DynamoDB memerlukan kehadiran peran tertaut layanan (AWSServiceRoleForApplicationAutoScaling_DynamoDBTable) yang melakukan tindakan penskalaan otomatis atas nama Anda. Peran ini dibuat secara otomatis untuk Anda. Untuk informasi selengkapnya, lihat [Peran tertaut layanan untuk Application Auto Scaling](#) dalam Panduan Pengguna Application Auto Scaling.

1. Gunakan perintah berikut untuk mendaftarkan target yang dapat diskalakan.

```
aws application-autoscaling register-scalable-target \  
  --service-namespace dynamodb \  
  --resource-id "table/TestTable" \  
  --target-name "TestTable"
```

```
--scalable-dimension "dynamodb:table:WriteCapacityUnits" \  
--min-capacity 5 \  
--max-capacity 10
```

2. Untuk memverifikasi pendaftaran, gunakan perintah berikut.

```
aws application-autoscaling describe-scalable-targets \  
--service-namespace dynamodb \  
--resource-id "table/TestTable"
```

Note

Anda juga dapat mendaftarkan target yang dapat diskalakan terhadap indeks sekunder global. Sebagai contoh, untuk indeks sekunder global ("test-index"), ID sumber daya dan argumen dimensi yang dapat diskalakan diperbarui dengan sesuai.

```
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
```

Langkah 3: Buat kebijakan penskalaan

Pada langkah ini, Anda membuat kebijakan penskalaan untuk TestTable. Kebijakan ini mendefinisikan detail di mana Application Auto Scaling dapat menyesuaikan throughput yang disediakan pada tabel, dan tindakan yang harus diambil ketika melakukannya. Anda mengaitkan kebijakan ini dengan target yang dapat diskalakan yang Anda tetapkan pada langkah sebelumnya (unit kapasitas tulis untuk tabel TestTable).

Kebijakan tersebut berisi elemen berikut:

- **PredefinedMetricSpecification**—Metrik yang dapat disesuaikan oleh Application Auto Scaling. Untuk DynamoDB, nilai berikut adalah nilai yang valid untuk **PredefinedMetricType**:
 - **DynamoDBReadCapacityUtilization**
 - **DynamoDBWriteCapacityUtilization**

- `ScaleOutCooldown`—Jumlah waktu minimum (dalam detik) antara setiap peristiwa Application Auto Scaling yang meningkatkan throughput yang disediakan. Parameter ini memungkinkan Application Auto Scaling untuk secara terus-menerus, tetapi tanpa agresivitas, meningkatkan throughput sebagai respons terhadap beban kerja dunia nyata. Pengaturan default untuk `ScaleOutCooldown` adalah 0.
- `ScaleInCooldown`—Jumlah waktu minimum (dalam detik) antara setiap peristiwa Application Auto Scaling yang mengurangi throughput yang disediakan. Parameter ini memungkinkan Application Auto Scaling untuk mengurangi throughput secara bertahap dan terprediksi. Pengaturan default untuk `ScaleInCooldown` adalah 0.
- `TargetValue`—Application Auto Scaling memastikan bahwa rasio kapasitas yang dikonsumsi terhadap kapasitas yang disediakan tetap atau mendekati nilai ini. Anda mendefinisikan `TargetValue` sebagai persentase.

Note

Untuk lebih memahami cara kerja `TargetValue`, misalkan Anda memiliki sebuah tabel dengan pengaturan throughput yang disediakan sebanyak 200 unit kapasitas tulis. Anda memutuskan untuk membuat kebijakan penskalaan untuk tabel ini, dengan `TargetValue` sebesar 70 persen.

Sekarang anggaplah Anda mulai mendorong lalu lintas tulis ke tabel sehingga throughput tulis aktual adalah sebesar 150 unit kapasitas. `consumed-to-provisioned` Rasionya sekarang (150/200), atau 75 persen. Rasio ini melebihi target Anda, sehingga Application Auto Scaling meningkatkan kapasitas tulis yang disediakan menjadi 215, sehingga rasionya (150 / 215), atau 69,77 persen—sedekat mungkin dengan `TargetValue`, tetapi tidak melebihinya.

Untuk `TestTable`, Anda mengatur `TargetValue` ke 50 persen. Application Auto Scaling menyesuaikan throughput yang disediakan tabel dalam kisaran 5-10 unit kapasitas (lihat [Langkah 2: Daftarkan target yang dapat diskalakan](#)) sehingga rasio tetap pada atau mendekati 50 persen. `consumed-to-provisioned` Anda mengatur nilai untuk `ScaleOutCooldown` dan `ScaleInCooldown` ke 60 detik.

1. Buat file bernama `scaling-policy.json` dengan isi berikut ini.

```
{
  "PredefinedMetricSpecification": {
    "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
```

```
    },  
    "ScaleOutCooldown": 60,  
    "ScaleInCooldown": 60,  
    "TargetValue": 50.0  
  }  
}
```

- Gunakan perintah AWS CLI berikut untuk membuat kebijakan.

```
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
```

- Dalam output, perhatikan bahwa Application Auto Scaling telah menciptakan dua CloudWatch alarm Amazon — masing-masing untuk batas atas dan bawah rentang target penskalaan.
- Gunakan perintah AWS CLI berikut untuk melihat detail lebih lanjut tentang kebijakan penskalaan.

```
aws application-autoscaling describe-scaling-policies \  
  --service-namespace dynamodb \  
  --resource-id "table/TestTable" \  
  --policy-name "MyScalingPolicy"
```

- Dalam output, verifikasi bahwa pengaturan kebijakan sesuai dengan spesifikasi Anda dari [Langkah 2: Daftarkan target yang dapat diskalakan](#) dan [Langkah 3: Buat kebijakan penskalaan](#).

Langkah 4: Dorong lalu lintas tulis ke TestTable

Sekarang Anda dapat menguji kebijakan penskalaan Anda dengan menulis data ke TestTable. Untuk melakukan ini, jalankan program Python.

- Buat file bernama `bulk-load-test-table.py` dengan isi berikut ini.

```
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. Masukkan perintah berikut untuk menjalankan program.

```
python bulk-load-test-table.py
```

Kapasitas tulis yang disediakan untuk TestTable sangat rendah (5 unit kapasitas tulis), sehingga terkadang mengalami kegagalan karena throttling tulis. Ini adalah perilaku yang diharapkan.

Biarkan program terus berjalan sementara Anda melanjutkan ke langkah berikutnya.

Langkah 5: Lihat tindakan Application Auto Scaling

Pada langkah ini, Anda melihat tindakan Application Auto Scaling yang dimulai atas nama Anda. Anda juga memverifikasi bahwa Application Auto Scaling telah memperbarui kapasitas tulis yang disediakan untuk TestTable.

1. Masukkan perintah berikut untuk melihat tindakan Application Auto Scaling.

```
aws application-autoscaling describe-scaling-activities \
    --service-namespace dynamodb
```


Jalankan kembali perintah ini sesekali, sementara program Python sedang berjalan. (Perlu beberapa menit hingga kebijakan penskalaan Anda diinvokasi.) Anda akan melihat output berikut.

```
...
{
  "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"
},
...
```

Hal ini menunjukkan bahwa Application Auto Scaling telah mengeluarkan permintaan `UpdateTable` untuk DynamoDB.

2. Masukkan perintah berikut untuk memverifikasi bahwa DynamoDB meningkatkan kapasitas tulis tabel.

```
aws dynamodb describe-table \
  --table-name TestTable \
  --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

`WriteCapacityUnits` semestinya telah diskalakan dari 5 ke 10.

(Opsional) Langkah 6: Hapus

Dalam tutorial ini, Anda telah membuat beberapa sumber daya. Anda dapat menghapus sumber daya ini jika sudah tidak membutuhkannya lagi.

1. Hapus kebijakan penskalaan untuk `TestTable`.

```
aws application-autoscaling delete-scaling-policy \  
  --service-namespace dynamodb \  
  --resource-id "table/TestTable" \  
  --scalable-dimension "dynamodb:table:WriteCapacityUnits" \  
  --policy-name "MyScalingPolicy"
```

2. Batalkan pendaftaran target yang dapat diskalakan.

```
aws application-autoscaling deregister-scalable-target \  
  --service-namespace dynamodb \  
  --resource-id "table/TestTable" \  
  --scalable-dimension "dynamodb:table:WriteCapacityUnits"
```

3. Hapus tabel TestTable.

```
aws dynamodb delete-table --table-name TestTable
```

Menggunakan SDK AWS untuk mengonfigurasi penskalaan otomatis pada tabel Amazon DynamoDB

Selain menggunakan AWS Management Console dan AWS Command Line Interface (AWS CLI), Anda dapat menulis aplikasi yang berinteraksi dengan penskalaan otomatis Amazon DynamoDB. Bagian ini berisi dua program Java yang dapat Anda gunakan untuk menguji fungsi ini:

- EnableDynamoDBAutoscaling.java
- DisableDynamoDBAutoscaling.java

Mengaktifkan Application Auto Scaling untuk suatu tabel

Program berikut menunjukkan contoh penyiapan kebijakan penskalaan otomatis untuk tabel DynamoDB (TestTable). Hal ini berjalan sebagai berikut:

- Program mendaftarkan unit kapasitas tulis sebagai target yang dapat diskalakan untuk TestTable. Kisaran untuk metrik ini adalah antara 5 dan 10 unit kapasitas tulis.
- Setelah target yang dapat diskalakan dibuat, program ini membangun konfigurasi pelacakan target. Kebijakan ini bertujuan untuk mempertahankan rasio target sebesar 50 persen antara kapasitas tulis yang dikonsumsi dan kapasitas tulis yang disediakan.

- Program ini kemudian membuat kebijakan penskalaan berdasarkan konfigurasi pelacakan target.

Note

Saat menghapus tabel atau replika tabel global secara manual, Anda tidak secara otomatis menghapus target, kebijakan penskalaan, atau alarm terkait yang dapat diskalakan. CloudWatch

Program ini meminta Anda untuk menyediakan Amazon Resource Name (ARN) untuk peran yang tertaut dengan layanan Application Auto Scaling. (Misalnya: `arn:aws:iam::122517410325:role/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`.) Pada program berikut, ganti `SERVICE_ROLE_ARN_GOES_HERE` dengan ARN yang sebenarnya.

```
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.TargetTrackingScalingPolicyConfiguration;

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());
}
```

```
}  
  
}
```

Menonaktifkan Application Auto Scaling untuk suatu tabel

Program berikut membalikkan proses sebelumnya. Program akan menghapus kebijakan penskalaan otomatis dan kemudian membatalkan pendaftaran target yang dapat diskalakan.

```
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());
}

}

}
```

Menambahkan tag dan label ke sumber daya

Anda dapat melabeli sumber daya Amazon DynamoDB menggunakan tag. Tag memungkinkan Anda mengategorikan sumber daya Anda dengan cara yang berbeda, misalnya, menurut tujuan, pemilik, lingkungan, atau kriteria lainnya. Tag dapat membantu Anda melakukan hal berikut:

- Mengidentifikasi sumber daya dengan cepat berdasarkan tag yang Anda tetapkan.
- Melihat tagihan AWS yang diuraikan menurut tag.

Note

Indeks sekunder lokal (LSI) dan indeks sekunder global (GSI) yang terkait dengan tabel yang diberi tag dilabeli dengan tag yang sama secara otomatis. Saat ini, penggunaan DynamoDB Streams tidak dapat diberi tag.

Pemberian tag didukung oleh layanan AWS, seperti Amazon EC2, Amazon S3, DynamoDB, dan banyak lagi. Pemberian tag yang efisien dapat memberikan wawasan biaya dengan memungkinkan Anda membuat laporan di seluruh layanan yang memiliki tag tertentu.

Untuk memulai pemberian tag, lakukan hal berikut:

1. Pahami [Batasan penandaan di DynamoDB](#).
2. Buat tag menggunakan [Pemberian Tag pada Sumber Saya di DynamoDB](#).
3. Gunakan [Laporan alokasi biaya](#) untuk melacak biaya AWS per tag aktif.

Akhirnya, merupakan praktik yang baik untuk mengikuti strategi pemberian tag yang optimal. Untuk informasi, lihat [AWS Strategi penandaan](#).

Batasan penandaan di DynamoDB

Setiap tag terdiri atas sebuah kunci dan sebuah nilai, yang keduanya Anda tentukan. Pembatasan berikut berlaku:

- Setiap tabel DynamoDB hanya dapat memiliki satu tag dengan kunci yang sama. Jika Anda mencoba menambahkan tag yang ada (kunci yang sama), nilai tag yang ada diperbarui ke nilai baru.
- Kunci dan nilai tag peka huruf besar dan kecil.
- Panjang kunci maksimum adalah 128 karakter Unicode.
- Panjang nilai maksimum adalah 256 karakter Unicode.
- Karakter yang diperbolehkan adalah huruf, spasi, dan angka, serta karakter khusus berikut: + - = . _ : /
- Jumlah maksimum tag per sumber daya adalah 50.
- Nama tag AWS-ditugaskan dan nilainya secara otomatis ditetapkan ke awalan `aws :`, yang tidak dapat Anda tetapkan. Nama tag AWS-assigned tidak dihitung menuju batas tag sebanyak 50. Nama tag User-assigned memiliki awalan `user :` dalam laporan alokasi biayanya.
- Anda tidak dapat mengubah ulang aplikasi tag.

Pemberian Tag pada Sumber Saya di DynamoDB

Anda dapat menggunakan konsol Amazon DynamoDB atau AWS Command Line Interface (AWS CLI) untuk menambahkan, mencantumkan, mengedit, atau menghapus tag. Anda kemudian dapat mengaktifkan tag yang ditetapkan pengguna ini sehingga mereka muncul di konsol AWS Billing and Cost Management untuk pelacakan alokasi biaya. Untuk informasi selengkapnya, lihat [Laporan alokasi biaya](#).

Untuk pengeditan massal, Anda juga dapat menggunakan Editor Tag pada AWS Management Console. Untuk informasi selengkapnya, lihat [Bekerja dengan Editor Tag](#).

Untuk menggunakan DynamoDB API sebagai gantinya, lihat operasi berikut di [Referensi Amazon DynamoDB API](#):

- [TagResource](#)
- [UntagResource](#)
- [ListTagsOfResource](#)

Topik

- [Menetapkan izin untuk memfilter menurut tag](#)
- [Menambahkan tag ke tabel baru atau yang sudah ada \(AWS Management Console\)](#)
- [Menambahkan tag ke tabel baru atau yang sudah ada \(AWS CLI\)](#)

Menetapkan izin untuk memfilter menurut tag

Untuk menggunakan tag untuk memfilter daftar tabel Anda di konsol DynamoDB, pastikan kebijakan pengguna Anda mencakup akses ke operasi berikut:

- `tag:GetTagKeys`
- `tag:GetTagValues`

Anda dapat mengakses operasi ini dengan melampirkan kebijakan IAM baru untuk pengguna Anda dengan mengikuti langkah-langkah di bawah ini.

1. Buka [konsol IAM](#) dengan pengguna Admin.
2. Pilih "Kebijakan" di menu navigasi kiri.
3. Pilih "Buat kebijakan."
4. Tempelkan kebijakan berikut ke editor JSON.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "tag:GetTagKeys",
        "tag:GetTagValues"
      ]
    }
  ]
}
```

```
    ],
    "Resource": "*"
  }
]
```

5. Selesaikan panduan dan tetapkan nama untuk kebijakan (misalnya, TagKeysAndValuesReadAccess).
6. Pilih "Pengguna" di menu navigasi kiri.
7. Dari daftar, pilih pengguna yang biasanya Anda gunakan untuk mengakses konsol DynamoDB.
8. Pilih "Tambahkan izin."
9. Pilih "Lampirkan kebijakan yang ada secara langsung."
10. Dari daftar, pilih kebijakan yang Anda buat sebelumnya.
11. Selesaikan panduan.

Menambahkan tag ke tabel baru atau yang sudah ada (AWS Management Console)

Anda dapat menggunakan konsol DynamoDB untuk menambahkan tag ke tabel baru saat Anda membuatnya, atau untuk menambahkan, mengedit, atau menghapus tag untuk tabel yang ada.

Untuk memberi tag pada sumber daya saat pembuatan (konsol)

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi, pilih Tabel, lalu pilih Buat tabel.
3. Pada halaman Buat tabel DynamoDB, masukkan nama dan kunci primer. Di bagian Tag, pilih Tambahkan tag baru dan masukkan tag yang ingin Anda gunakan.

Untuk informasi selengkapnya tentang struktur tag, lihat [Batasan penandaan di DynamoDB](#).

Untuk informasi selengkapnya tentang membuat tabel, lihat [Operasi dasar pada tabel DynamoDB](#).

Untuk memberi tag pada sumber daya yang ada (konsol)

Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.

1. Di panel navigasi, pilih Tabel.

2. Pilih tabel dalam daftar, kemudian pilih tab Pengaturan tambahan. Anda dapat menambahkan, mengedit, atau menghapus tag di bagian Tag di bagian bawah halaman.

Menambahkan tag ke tabel baru atau yang sudah ada (AWS CLI)

Contoh berikut menunjukkan cara menggunakan AWS CLI untuk menentukan tag ketika Anda membuat tabel dan indeks, dan untuk memberi tag pada sumber daya yang ada.

Untuk memberi tag pada sumber daya saat pembuatan (AWS CLI)

- Contoh berikut membuat tabel `Movies` baru dan menambahkan tag `Owner` dengan nilai `blueTeam`:

```
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
```

Untuk memberi tag pada sumber daya yang ada (AWS CLI)

- Contoh berikut menambahkan tag `Owner` dengan nilai `blueTeam` untuk tabel `Movies`:

```
aws dynamodb tag-resource \  
  --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies \  
  --tags Key=Owner,Value=blueTeam
```

Untuk mencantumkan semua tag untuk tabel (AWS CLI)

- Contoh berikut mencantumkan semua tag yang terkait dengan tabel `Movies`:

```
aws dynamodb list-tags-of-resource \  
  --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies
```

Laporan alokasi biaya

AWS menggunakan tag untuk menyusun biaya sumber daya pada laporan alokasi biaya Anda. AWS menyediakan dua jenis tag alokasi biaya:

- Tag AWS-dihasilkan. AWS menentukan, membuat, dan menerapkan tag ini untuk Anda.
- Tag yang ditentukan pengguna. Anda menentukan, membuat, dan menerapkan tag ini.

Anda harus mengaktifkan kedua jenis tag secara terpisah sebelum tag tersebut muncul di Cost Explorer atau laporan alokasi biaya.

Untuk mengaktifkan tag AWS-dihasilkan:

1. Masuk ke AWS Management Console dan buka Konsol Manajemen Penagihan dan Biaya di <https://console.aws.amazon.com/billing/home#/>.
2. Di panel navigasi, pilih Tag Alokasi Biaya.
3. Di bawah Tag Alokasi Biaya AWS-Dihasilkan, pilih Aktifkan.

Untuk mengaktifkan tag yang ditentukan pengguna:

1. Masuk ke AWS Management Console dan buka Konsol Manajemen Penagihan dan Biaya di <https://console.aws.amazon.com/billing/home#/>.
2. Di panel navigasi, pilih Tag Alokasi Biaya.
3. Di bawah Tag Alokasi Biaya yang Ditentukan Pengguna, pilih Aktifkan.

Setelah Anda membuat dan mengaktifkan tag, AWS menghasilkan laporan alokasi biaya dengan penggunaan dan biaya yang dikelompokkan berdasarkan tag aktif Anda. Laporan alokasi biaya mencakup semua biaya AWS untuk setiap periode penagihan. Laporan ini mencakup sumber daya yang diberi tag dan tidak diberi tag, sehingga Anda dapat dengan jelas mengatur biaya untuk sumber daya.

Note

Saat ini, data apa pun yang ditransfer dari DynamoDB tidak akan dipecah menurut tag pada laporan alokasi biaya.

Untuk informasi selengkapnya, lihat [Menggunakan tag alokasi biaya](#).

Bekerja dengan tabel DynamoDB di Java

Anda dapat menggunakan AWS SDK for Java untuk membuat, memperbarui, dan menghapus tabel Amazon DynamoDB, mencantumkan semua tabel di akun Anda, atau mendapatkan informasi tentang tabel tertentu.

Topik

- [Membuat tabel](#)
- [Memperbarui tabel](#)
- [Menghapus tabel](#)
- [Mencantumkan tabel](#)
- [Contoh: Membuat, memperbarui, menghapus, dan mencantumkan tabel menggunakan AWS SDK for Java](#)

Membuat tabel

Untuk membuat tabel, Anda harus memberikan nama tabel, kunci primernya, dan nilai throughput yang disediakan. Potongan kode berikut membuat tabel contoh yang menggunakan ID atribut tipe numerik sebagai kunci primernya.

Untuk membuat tabel menggunakan API AWS SDK for Java

1. Buat instans dari kelas `DynamoDB`.
2. Mulai `CreateTableRequest` untuk memberikan informasi permintaan.

Anda harus memberikan nama tabel, definisi atribut, skema kunci, dan nilai-nilai throughput yang disediakan.

3. Jalankan metode `createTable` dengan menyediakan objek permintaan sebagai parameter.

Contoh kode berikut mendemonstrasikan langkah sebelumnya.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

List<AttributeDefinition> attributeDefinitions= new ArrayList<AttributeDefinition>();
```

```
attributeDefinitions.add(new
    AttributeDefinition().withAttributeName("Id").withAttributeType("N"));

List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
keySchema.add(new
    KeySchemaElement().withAttributeName("Id").withKeyType(KeyType.HASH));

CreateTableRequest request = new CreateTableRequest()
    .withTableName(tableName)
    .withKeySchema(keySchema)
    .withAttributeDefinitions(attributeDefinitions)
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits(5L)
        .withWriteCapacityUnits(6L));

Table table = dynamoDB.createTable(request);

table.waitForActive();
```

Tabel belum siap digunakan hingga DynamoDB membuatnya dan menetapkan statusnya ke AKTIF. Permintaan `createTable` mengembalikan objek `Table` yang dapat Anda gunakan untuk mendapatkan informasi selengkapnya tentang tabel.

Example

```
TableDescription tableDescription =
    dynamoDB.getTable(tableName).describe();

System.out.printf("%s: %s \t ReadCapacityUnits: %d \t WriteCapacityUnits: %d",
    tableDescription.getTableStatus(),
    tableDescription.getTableName(),
    tableDescription.getProvisionedThroughput().getReadCapacityUnits(),
    tableDescription.getProvisionedThroughput().getWriteCapacityUnits());
```

Anda dapat menjalankan metode `describe` klien untuk mendapatkan informasi tabel kapan saja.

Example

```
TableDescription tableDescription = dynamoDB.getTable(tableName).describe();
```

Memperbarui tabel

Anda dapat memperbarui hanya nilai throughput yang disediakan dari tabel yang ada. Bergantung pada persyaratan aplikasi, Anda mungkin perlu memperbarui nilai-nilai ini.

Note

Untuk informasi selengkapnya tentang peningkatan dan penurunan throughput per hari, lihat [Layanan, akun, dan tabel kuota di Amazon DynamoDB](#).

Untuk memperbarui tabel menggunakan API AWS SDK for Java

1. Buat instans kelas `Table`.
2. Buat instans dari kelas `ProvisionedThroughput` untuk menyediakan nilai throughput baru.
3. Jalankan metode `updateTable` dengan menyediakan instans `ProvisionedThroughput` sebagai parameter.

Contoh kode berikut mendemonstrasikan langkah sebelumnya.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

ProvisionedThroughput provisionedThroughput = new ProvisionedThroughput()
    .withReadCapacityUnits(15L)
    .withWriteCapacityUnits(12L);

table.updateTable(provisionedThroughput);

table.waitForActive();
```

Menghapus tabel

Untuk menghapus tabel menggunakan API AWS SDK for Java

1. Buat instans dari kelas `Table`.

2. Buat instans dari kelas `DeleteTableRequest` dan berikan nama tabel yang ingin Anda hapus.
3. Jalankan metode `deleteTable` dengan menyediakan instans `Table` sebagai parameter.

Contoh kode berikut mendemonstrasikan langkah sebelumnya.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

table.delete();

table.waitForDelete();
```

Mencantumkan tabel

Untuk mencantumkan tabel di akun Anda, buat instans dari `DynamoDB` dan jalankan metode `listTables`. [ListTables](#) Operasi tidak memerlukan parameter.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

TableCollection<ListTablesResult> tables = dynamoDB.listTables();
Iterator<Table> iterator = tables.iterator();

while (iterator.hasNext()) {
    Table table = iterator.next();
    System.out.println(table.getTableName());
}
```

Contoh: Membuat, memperbarui, menghapus, dan mencantumkan tabel menggunakan AWS SDK for Java

Contoh kode berikut menggunakan Document API AWS SDK for Java untuk membuat, memperbarui, dan menghapus tabel Amazon DynamoDB (`ExampleTable`). Sebagai bagian dari pembaruan tabel,

ini meningkatkan nilai throughput yang disediakan. Contoh ini juga mencantumkan semua tabel di akun Anda dan mendapatkan deskripsi tabel tertentu. Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihat [Contoh kode Java](#).

```
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.Table;
import com.amazonaws.services.dynamodbv2.document.TableCollection;
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.ListTablesResult;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.TableDescription;

public class DocumentAPITableExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "ExampleTable";

    public static void main(String[] args) throws Exception {

        createExampleTable();
        listMyTables();
        getTableInformation();
        updateExampleTable();

        deleteExampleTable();
    }

    static void createExampleTable() {

        try {
```

```
        List<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
        attributeDefinitions.add(new
AttributeDefinition().withAttributeName("Id").withAttributeType("N"));

        List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
        keySchema.add(new
KeySchemaElement().withAttributeName("Id").withKeyType(KeyType.HASH)); // Partition

        // key

        CreateTableRequest request = new
CreateTableRequest().withTableName(tableName).withKeySchema(keySchema)

.withAttributeDefinitions(attributeDefinitions).withProvisionedThroughput(
new
ProvisionedThroughput().withReadCapacityUnits(5L).withWriteCapacityUnits(6L));

        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();

        getTableInformation();

    } catch (Exception e) {
        System.err.println("CreateTable request failed for " + tableName);
        System.err.println(e.getMessage());
    }
}

static void listMyTables() {

    TableCollection<ListTablesResult> tables = dynamoDB.listTables();
    Iterator<Table> iterator = tables.iterator();

    System.out.println("Listing table names");

    while (iterator.hasNext()) {
        Table table = iterator.next();
```

```
        System.out.println(table.getTableName());
    }
}

static void getTableInformation() {

    System.out.println("Describing " + tableName);

    TableDescription tableDescription = dynamoDB.getTable(tableName).describe();
    System.out.format(
        "Name: %s:\n" + "Status: %s \n" + "Provisioned Throughput (read
capacity units/sec): %d \n"
        + "Provisioned Throughput (write capacity units/sec): %d \n",
        tableDescription.getTableName(), tableDescription.getTableStatus(),
        tableDescription.getProvisionedThroughput().getReadCapacityUnits(),
        tableDescription.getProvisionedThroughput().getWriteCapacityUnits());
}

static void updateExampleTable() {

    Table table = dynamoDB.getTable(tableName);
    System.out.println("Modifying provisioned throughput for " + tableName);

    try {
        table.updateTable(new
ProvisionedThroughput().withReadCapacityUnits(6L).withWriteCapacityUnits(7L));

        table.waitForActive();
    } catch (Exception e) {
        System.err.println("UpdateTable request failed for " + tableName);
        System.err.println(e.getMessage());
    }
}

static void deleteExampleTable() {

    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());
    }
}
}
```

Bekerja dengan tabel DynamoDB di .NET

Anda dapat menggunakan AWS SDK for .NET untuk membuat, memperbarui, dan menghapus tabel, mencantumkan semua tabel di akun Anda, atau mendapatkan informasi tentang tabel tertentu.

Berikut ini adalah langkah-langkah umum untuk operasi tabel Amazon DynamoDB menggunakan AWS SDK for .NET.

1. Buat instans dari kelas `AmazonDynamoDBClient` (klien).
2. Berikan parameter yang diperlukan dan opsional untuk operasi dengan membuat objek permintaan yang sesuai.

Misalnya, buat objek `CreateTableRequest` untuk membuat tabel dan objek `UpdateTableRequest` untuk memperbarui tabel yang ada.

3. Jalankan metode yang sesuai yang disediakan oleh klien yang Anda buat pada langkah sebelumnya.

Note

Contoh dalam bagian ini tidak berfungsi dengan .NET core karena tidak mendukung metode sinkron. Untuk informasi selengkapnya, lihat [AWS asynchronous API untuk .NET](#).

Topik

- [Membuat tabel](#)
- [Memperbarui tabel](#)
- [Menghapus tabel](#)
- [Mencantumkan tabel](#)

- [Contoh: Membuat, memperbarui, menghapus, dan mencantumkan tabel menggunakan API tingkat rendah AWS SDK for .NET](#)

Membuat tabel

Untuk membuat tabel, Anda harus memberikan nama tabel, kunci primernya, dan nilai throughput yang disediakan.

Untuk membuat tabel menggunakan API tingkat rendah AWS SDK for .NET

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Buat instans dari kelas `CreateTableRequest` untuk memberikan informasi permintaan.
Anda harus memberikan nama tabel, kunci primer, dan nilai throughput yang disediakan.
3. Jalankan metode `AmazonDynamoDBClient.CreateTable` dengan menyediakan objek permintaan sebagai parameter.

Contoh #C berikut menunjukkan langkah-langkah sebelumnya. Sampel membuat tabel (`ProductCatalog`) yang menggunakan `Id` sebagai kunci primer dan kumpulan nilai throughput yang disediakan. Bergantung pada persyaratan aplikasi, Anda dapat memperbarui nilai throughput yang disediakan dengan menggunakan API `UpdateTable`.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new CreateTableRequest
{
    TableName = tableName,
    AttributeDefinitions = new List<AttributeDefinition>()
    {
        new AttributeDefinition
        {
            AttributeName = "Id",
            AttributeType = "N"
        }
    },
    KeySchema = new List<KeySchemaElement>()
    {
        new KeySchemaElement
        {
```

```
        AttributeName = "Id",
        KeyType = "HASH" //Partition key
    }
},
ProvisionedThroughput = new ProvisionedThroughput
{
    ReadCapacityUnits = 10,
    WriteCapacityUnits = 5
}
};

var response = client.CreateTable(request);
```

Anda harus menunggu hingga DynamoDB membuat tabel dan menetapkan statusnya menjadi ACTIVE. Respons `CreateTable` mencakup properti `TableDescription` yang menyediakan informasi tabel yang diperlukan.

Example

```
var result = response.CreateTableResult;
var tableDescription = result.TableDescription;
Console.WriteLine("{1}: {0} \t ReadCapacityUnits: {2} \t WriteCapacityUnits: {3}",
    tableDescription.TableStatus,
    tableDescription.TableName,
    tableDescription.ProvisionedThroughput.ReadCapacityUnits,
    tableDescription.ProvisionedThroughput.WriteCapacityUnits);

string status = tableDescription.TableStatus;
Console.WriteLine(tableName + " - " + status);
```

Anda juga dapat memanggil metode `DescribeTable` klien untuk mendapatkan informasi tabel kapan saja.

Example

```
var res = client.DescribeTable(new DescribeTableRequest{TableName = "ProductCatalog"});
```

Memperbarui tabel

Anda dapat memperbarui hanya nilai throughput yang disediakan dari tabel yang ada. Bergantung pada persyaratan aplikasi, Anda mungkin perlu memperbarui nilai-nilai ini.

Note

Anda dapat meningkatkan kapasitas throughput sesering yang diperlukan, dan menurunkannya dalam batasan tertentu. Untuk informasi selengkapnya tentang peningkatan dan penurunan throughput per hari, lihat [Layanan, akun, dan tabel kuota di Amazon DynamoDB](#).

Untuk memperbarui tabel menggunakan API tingkat rendah AWS SDK for .NET

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Buat instans dari kelas `UpdateTableRequest` untuk memberikan informasi permintaan.

Anda harus memberikan nama tabel dan nilai throughput baru yang disediakan.

3. Jalankan metode `AmazonDynamoDBClient.UpdateTable` dengan menyediakan objek permintaan sebagai parameter.

Contoh #C berikut menunjukkan langkah-langkah sebelumnya.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ExampleTable";

var request = new UpdateTableRequest()
{
    TableName = tableName,
    ProvisionedThroughput = new ProvisionedThroughput()
    {
        // Provide new values.
        ReadCapacityUnits = 20,
        WriteCapacityUnits = 10
    }
};
var response = client.UpdateTable(request);
```

Menghapus tabel

Ikuti langkah berikut untuk menghapus tabel menggunakan API tingkat rendah .NET.

Untuk menghapus tabel menggunakan API tingkat rendah AWS SDK for .NET

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Buat instans dari kelas `DeleteTableRequest` dan berikan nama tabel yang ingin Anda hapus.
3. Jalankan metode `AmazonDynamoDBClient.DeleteTable` dengan menyediakan objek permintaan sebagai parameter.

Contoh kode #C berikut menunjukkan langkah-langkah sebelumnya.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ExampleTable";

var request = new DeleteTableRequest{ TableName = tableName };
var response = client.DeleteTable(request);
```

Mencantumkan tabel

Untuk mencantumkan tabel di akun Anda menggunakan API tingkat rendah AWS SDK for .NET, buat sebuah instans dari `AmazonDynamoDBClient` dan jalankan metode `ListTables`.

[ListTables](#) Operasi tidak memerlukan parameter. Namun, Anda dapat menentukan parameter opsional. Misalnya, Anda dapat mengatur parameter `Limit` jika ingin menggunakan paging untuk membatasi jumlah nama tabel per halaman. Hal ini mengharuskan Anda membuat objek `ListTablesRequest` dan memberikan parameter opsional seperti yang ditunjukkan dalam contoh C# berikut. Seiring dengan ukuran halaman, permintaan menetapkan parameter `ExclusiveStartTableName`. Awalnya, `ExclusiveStartTableName` adalah null. Namun, setelah mengambil halaman pertama hasil, untuk mengambil halaman hasil berikutnya, Anda harus menetapkan nilai parameter ini ke properti `LastEvaluatedTableName` hasil saat ini.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

// Initial value for the first page of table names.
string lastEvaluatedTableName = null;
do
{
    // Create a request object to specify optional parameters.
```

```
var request = new ListTablesRequest
{
    Limit = 10, // Page size.
    ExclusiveStartTableName = lastEvaluatedTableName
};

var response = client.ListTables(request);
ListTablesResult result = response.ListTablesResult;
foreach (string name in result.TableNames)
    Console.WriteLine(name);

lastEvaluatedTableName = result.LastEvaluatedTableName;

} while (lastEvaluatedTableName != null);
```

Contoh: Membuat, memperbarui, menghapus, dan mencantumkan tabel menggunakan API tingkat rendah AWS SDK for .NET

Contoh C# berikut membuat, memperbarui, dan menghapus tabel (ExampleTable). Contoh ini juga mencantumkan semua tabel di akun Anda dan mendapatkan deskripsi tabel tertentu. Pembaruan tabel meningkatkan nilai throughput yang disediakan. Untuk step-by-step instruksi untuk menguji contoh berikut, lihat [Contoh kode .NET](#).

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelTableExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        private static string tableName = "ExampleTable";

        static void Main(string[] args)
        {
            try
            {
                CreateExampleTable();
                ListMyTables();
            }
        }
    }
}
```

```
        GetTableInformation();
        UpdateExampleTable();

        DeleteExampleTable();

        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 CreateExampleTable()
{
    Console.WriteLine("\n*** Creating table ***");
    var request = new CreateTableRequest
    {
        AttributeDefinitions = new List<AttributeDefinition>()
        {
            new AttributeDefinition
            {
                AttributeName = "Id",
                AttributeType = "N"
            },
            new AttributeDefinition
            {
                AttributeName = "ReplyDateTime",
                AttributeType = "N"
            }
        },
        KeySchema = new List<KeySchemaElement>
        {
            new KeySchemaElement
            {
                AttributeName = "Id",
                KeyType = "HASH" //Partition key
            },
            new KeySchemaElement
            {
                AttributeName = "ReplyDateTime",
                KeyType = "RANGE" //Sort key
            }
        },
    },
```

```
        ProvisionedThroughput = new ProvisionedThroughput
        {
            ReadCapacityUnits = 5,
            WriteCapacityUnits = 6
        },
        TableName = tableName
    };

    var response = client.CreateTable(request);

    var tableDescription = response.TableDescription;
    Console.WriteLine("{1}: {0} \t ReadsPerSec: {2} \t WritesPerSec: {3}",
        tableDescription.TableStatus,
        tableDescription.TableName,
        tableDescription.ProvisionedThroughput.ReadCapacityUnits,
        tableDescription.ProvisionedThroughput.WriteCapacityUnits);

    string status = tableDescription.TableStatus;
    Console.WriteLine(tableName + " - " + status);

    WaitUntilTableReady(tableName);
}

private static void ListMyTables()
{
    Console.WriteLine("\n*** listing tables ***");
    string lastTableNameEvaluated = null;
    do
    {
        var request = new ListTablesRequest
        {
            Limit = 2,
            ExclusiveStartTableName = lastTableNameEvaluated
        };

        var response = client.ListTables(request);
        foreach (string name in response.TableNames)
            Console.WriteLine(name);

        lastTableNameEvaluated = response.LastEvaluatedTableName;
    } while (lastTableNameEvaluated != null);
}

private static void GetTableInformation()
```

```
{
    Console.WriteLine("\n*** Retrieving table information ***");
    var request = new DescribeTableRequest
    {
        TableName = tableName
    };

    var response = client.DescribeTable(request);

    TableDescription description = response.Table;
    Console.WriteLine("Name: {0}", description.TableName);
    Console.WriteLine("# of items: {0}", description.ItemCount);
    Console.WriteLine("Provision Throughput (reads/sec): {0}",
        description.ProvisionedThroughput.ReadCapacityUnits);
    Console.WriteLine("Provision Throughput (writes/sec): {0}",
        description.ProvisionedThroughput.WriteCapacityUnits);
}

private static void UpdateExampleTable()
{
    Console.WriteLine("\n*** Updating table ***");
    var request = new UpdateTableRequest()
    {
        TableName = tableName,
        ProvisionedThroughput = new ProvisionedThroughput()
        {
            ReadCapacityUnits = 6,
            WriteCapacityUnits = 7
        }
    };

    var response = client.UpdateTable(request);

    WaitUntilTableReady(tableName);
}

private static void DeleteExampleTable()
{
    Console.WriteLine("\n*** Deleting table ***");
    var request = new DeleteTableRequest
    {
        TableName = tableName
    };
};
```

```
        var response = client.DeleteTable(request);

        Console.WriteLine("Table is being deleted...");
    }

    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");
    }
}
```

Tabel global - Replikasi multi-Wilayah untuk DynamoDB

Tabel global Amazon DynamoDB adalah opsi database terkelola penuh, multi-wilayah, dan multi-aktif yang memberikan performa baca dan penulisan performa cepat dan lokal untuk aplikasi global berskala besar.

Tabel global menyediakan solusi terkelola penuh untuk men-deploy basis data multi-Wilayah dan multi-aktif, tanpa harus membangun dan memelihara solusi replikasi Anda sendiri. Anda dapat

menentukan AWS Wilayah tempat Anda ingin tabel tersedia dan DynamoDB akan menyebarkan perubahan data yang sedang berlangsung ke semuanya.

Manfaat khusus penggunaan tabel global meliputi:

- Mereplikasi tabel DynamoDB Anda secara otomatis di seluruh wilayah pilihan Anda AWS
- Menghilangkan kesulitan dalam mereplikasi data antarWilayah dan menyelesaikan konflik pembaruan, sehingga Anda dapat fokus pada logika bisnis aplikasi Anda.
- Membantu aplikasi Anda tetap tersedia bahkan ketika terjadi isolasi atau degradasi di seluruh Wilayah.

Tabel global DynamoDB ideal untuk aplikasi berskala besar dengan pengguna yang tersebar secara global. Di lingkungan seperti itu, pengguna mengharapkan performa aplikasi yang sangat cepat. Tabel global menyediakan replikasi multi-aktif otomatis ke AWS Wilayah di seluruh dunia. Tabel global memungkinkan Anda memberikan akses data latensi rendah kepada pengguna Anda di mana pun mereka berada.

Video berikut akan memberi Anda gambaran pengantar tentang tabel global.

Anda dapat mengatur tabel global di AWS Management Console atau AWS CLI. Tabel global menggunakan API DynamoDB yang sudah ada, sehingga perubahan aplikasi tidak diperlukan. Anda hanya membayar untuk sumber daya yang disediakan tanpa biaya atau komitmen di muka.

[Tabel global untuk replikasi antar-Wilayah](#)

Topik

- [Mereplikasi data secara mulus di Wilayah dengan tabel global](#)
- [Memberikan keamanan dan akses untuk tabel global Anda dengan AWS KMS](#)
- [Tabel global: Cara kerjanya](#)
- [Praktik terbaik dan persyaratan untuk mengelola tabel global](#)
- [Tutorial: Membuat tabel global](#)
- [Memantau tabel global](#)
- [Menggunakan IAM dengan tabel global](#)
- [Menentukan versi tabel global yang Anda gunakan](#)
- [Memutakhirkan tabel global ke versi Saat Ini \(2019.11.21\) dari versi Legacy \(2017.11.29\)](#)

Mereplikasi data secara mulus di Wilayah dengan tabel global

Misalkan Anda memiliki basis pelanggan besar yang tersebar di tiga wilayah geografis—Pantai Timur AS, Pantai Barat AS, dan Eropa Barat. Pelanggan tersebut dapat memperbarui informasi profil mereka menggunakan aplikasi Anda. Untuk memenuhi kasus penggunaan ini, Anda perlu membuat tiga tabel DynamoDB identik bernama `CustomerProfiles`, dalam tiga Wilayah AWS berbeda tempat pelanggan berada. Ketiga tabel ini akan sepenuhnya terpisah satu sama lain—perubahan pada data di satu tabel tidak akan terlihat di tabel lainnya. Tanpa solusi replikasi terkelola, Anda harus menulis kode untuk mereplikasi perubahan data. Namun, melakukan hal ini akan memakan waktu dan tenaga.

Daripada menulis kode Anda sendiri, Anda bisa membuat tabel global yang terdiri dari tiga tabel `CustomerProfiles` khusus Wilayah. DynamoDB kemudian akan otomatis mereplikasi perubahan data di antara tabel tersebut sehingga perubahan pada data `CustomerProfiles` di satu Wilayah akan menyebar ke Wilayah lain dengan lancar. Selain itu, jika salah satu AWS Wilayah menjadi tidak tersedia sementara, pelanggan Anda masih dapat mengakses `CustomerProfiles` data yang sama di Wilayah lain.

Note

- Dukungan Wilayah untuk tabel global [Versi tabel global 2017.11.29 \(Legacy\)](#) terbatas untuk AS Timur (Virginia Utara), AS Timur (Ohio), AS Barat (California Utara), AS Barat (Oregon), Eropa (Irlandia), Eropa (London), Eropa (Frankfurt), Asia Pasifik (Singapura), Asia Pasifik (Sydney), Asia Pasifik (Tokyo), dan Asia Pasifik (Seoul).
- Operasi transaksional memberikan jaminan atomicity, consistency, isolation, dan durability (ACID) hanya di wilayah tempat awal tulis dilakukan. Transaksi tidak didukung di seluruh wilayah dalam tabel global. Misalnya, jika Anda memiliki tabel global dengan replika di wilayah AS Timur (Ohio) dan AS Barat (Oregon) dan melakukan `TransactWriteItems` operasi di Wilayah AS Timur (Virginia N.), Anda dapat mengamati transaksi yang diselesaikan sebagian di Wilayah AS Barat (Oregon) saat perubahan direplikasi. Perubahan hanya akan direplikasi ke wilayah lain setelah perubahan telah dilakukan di wilayah sumber.
- Jika Anda [menonaktifkan AWS Region](#), DynamoDB akan menghapus replika ini dari grup replikasi, 20 jam setelah mendeteksi Wilayah sebagai tidak dapat diakses. AWS Replika tidak akan dihapus dan replikasi dari dan ke wilayah ini akan dihentikan.
- Anda harus menunggu 24 jam setelah Anda menambahkan replika baca agar berhasil menghapus tabel sumber. Jika mencoba menghapus tabel dalam 24 jam pertama setelah

menambahkan replika baca, Anda akan menerima pesan kesalahan yang menyatakan: "Replika tidak dapat dihapus karena telah bertindak sebagai wilayah sumber untuk replika baru yang ditambahkan dalam tabel dalam 24 jam terakhir".

- Tidak ada dampak performa pada wilayah sumber saat menambahkan replika baru.
- Jika Anda mengubah kapasitas baca dan tulis replika, kapasitas tulis baru akan tercermin ke replika lain yang disinkronkan tetapi kapasitas baca baru tidak.

Untuk informasi tentang ketersediaan dan harga AWS Wilayah, lihat Harga [Amazon DynamoDB](#).

Memberikan keamanan dan akses untuk tabel global Anda dengan AWS KMS

- Anda dapat melakukan AWS KMS operasi pada tabel global Anda dengan menggunakan peran `AWSServiceRoleForDynamoDBReplication` terkait layanan terhadap [kunci yang dikelola pelanggan](#) atau yang [Kunci yang dikelola AWS](#) digunakan untuk mengenkripsi replika.
- Jika kunci yang dikelola pelanggan yang digunakan untuk mengenkripsi replika tidak dapat diakses, DynamoDB akan menghapus replika ini dari grup replikasi. Replika tidak akan dihapus dan replikasi dari dan ke wilayah ini akan dihentikan, 20 jam setelah mendeteksi bahwa kunci KMS tidak dapat diakses.
- Jika ingin menonaktifkan [kunci yang dikelola pelanggan](#) yang digunakan untuk mengenkripsi tabel replika, Anda harus melakukannya hanya jika kunci tidak lagi digunakan untuk mengenkripsi tabel replika. Setelah mengeluarkan perintah untuk menghapus tabel replika, Anda harus menunggu operasi penghapusan selesai dan tabel global menjadi `Active` sebelum menonaktifkan kunci. Tidak melakukannya dapat mengakibatkan replikasi data parsial dari dan ke tabel replika.
- Jika ingin mengubah atau menghapus kebijakan peran IAM untuk tabel replika, Anda harus melakukannya ketika tabel replika berada dalam status `Active`. Jika tidak, pembuatan, pembaruan, atau penghapusan tabel replika bisa gagal.
- Tabel global dibuat dengan perlindungan penghapusan dinonaktifkan secara default. Meskipun perlindungan penghapusan diaktifkan untuk tabel global, setiap replika tabel tersebut akan dimulai dengan perlindungan penghapusan dinonaktifkan secara default.
- Saat perlindungan penghapusan dinonaktifkan untuk suatu tabel, tabel tersebut dapat terhapus secara tidak sengaja. Saat perlindungan penghapusan diaktifkan untuk suatu tabel, tidak ada yang bisa menghapusnya.

- Mengubah pengaturan perlindungan penghapusan untuk satu tabel replika tidak akan memperbarui replika lain dalam grup.

Note

Kunci yang dikelola pelanggan tidak didukung di [Versi tabel global 2017.11.29 \(Legacy\)](#). Jika Anda ingin menggunakan kunci yang dikelola pelanggan di DynamoDB Global Table, Anda perlu memutakhirkan tabel ke Tabel [Global versi 2019.11.21 \(Saat Ini\)](#) dan kemudian mengaktifkannya.

Tabel global: Cara kerjanya

Bagian berikut menjelaskan konsep dan perilaku tabel global di Amazon DynamoDB.

Konsep tabel global

Tabel global adalah koleksi dari satu atau beberapa tabel replika, semuanya dimiliki oleh satu akun AWS.

Tabel replika (atau replika, untuk singkatnya) adalah tabel DynamoDB tunggal yang berfungsi sebagai bagian dari tabel global. Setiap replika menyimpan set item data yang sama. Setiap tabel global tertentu hanya dapat memiliki satu tabel replika per Wilayah AWS. Untuk informasi selengkapnya tentang cara mulai menggunakan tabel global, lihat [Tutorial: Membuat tabel global](#).

Saat Anda membuat tabel global DynamoDB, tabel tersebut terdiri dari beberapa tabel replika (satu per Wilayah) yang diperlakukan DynamoDB sebagai satu unit. Setiap replika memiliki nama tabel yang sama dan skema kunci primer yang sama. Ketika aplikasi menulis data ke tabel replika di satu wilayah, DynamoDB menyebarkan aktivitas tulis tersebut ke tabel replika lain di AWS Wilayah lain secara otomatis.

Anda dapat menambahkan tabel replika ke tabel global sehingga dapat tersedia di Wilayah tambahan.

Dengan Versi 2019.11.21 (Terbaru), saat Anda membuat Indeks Sekunder Global di satu Wilayah, indeks tersebut otomatis direplikasi ke Wilayah lain serta di-backfill secara otomatis.

Tugas umum

Tugas umum untuk tabel global berfungsi sebagai berikut.

Anda dapat menghapus tabel replika tabel global sama seperti tabel biasa. Ini akan menghentikan replikasi ke Wilayah itu dan menghapus salinan tabel yang disimpan di Wilayah itu. Anda tidak dapat memutuskan replikasi dan membuat salinan tabel ada sebagai entitas independen. Anda dapat menyalin tabel global ke tabel lokal di Wilayah tersebut, lalu menghapus replika global untuk Wilayah tersebut.

Note

Anda tidak akan dapat menghapus tabel sumber hingga setidaknya 24 jam setelah tabel tersebut digunakan untuk memulai Wilayah baru. Jika mencoba menghapusnya terlalu cepat, Anda akan menerima pesan kesalahan.

Konflik dapat terjadi jika aplikasi memperbarui item yang sama di Wilayah yang berbeda pada waktu yang sama. Untuk membantu memastikan konsistensi akhirnya, tabel global DynamoDB menggunakan metode “penulis terakhir menang” untuk merekonsiliasi antara pembaruan yang dilakukan secara bersamaan. Semua replika akan menyetujui pembaruan terkini dan berkumpul menuju status ketika semua replika memiliki data yang identik.

Note

Ada beberapa cara untuk menghindari konflik, antara lain:

- Hanya mengizinkan penulisan ke tabel dalam satu Wilayah.
- Merutekan lalu lintas pengguna ke Wilayah yang berbeda sesuai dengan kebijakan tulis Anda, untuk memastikan tidak ada konflik.
- Menghindari penggunaan pembaruan non-idempoten seperti `Bookmark = Bookmark + 1`, dan mendukung pembaruan statis seperti `Bookmark=25`.
- Perlu diingat bahwa ketika Anda perlu merutekan tulis atau baca ke satu Wilayah, aplikasi Anda yang akan memastikan penerapan alur tersebut.

Memantau tabel global

Anda dapat menggunakan CloudWatch untuk mengamati metrik `ReplicationLatency`. CloudWatch melacak waktu yang berlalu antara ketika item ditulis ke tabel replika, dan ketika item tersebut muncul di replika lain di tabel global. Waktu tersebut dinyatakan dalam milidetik dan

dipancarkan untuk setiap pasangan Wilayah-sumber dan Wilayah-tujuan. Metrik ini disimpan di Wilayah sumber. Ini adalah satu-satunya CloudWatch metrik yang disediakan oleh Global Tables v2.

Latensi replikasi yang akan Anda alami tergantung pada jarak antara yang Anda pilih Wilayah AWS, serta variabel lainnya. Jika tabel asli Anda berada di Wilayah AS Barat (California Utara) (us-barat-1), replika di Wilayah yang lebih dekat, seperti Wilayah AS Barat (Oregon) (us-barat-2), akan memiliki latensi replikasi yang lebih rendah dibandingkan dengan replika di wilayah yang jauh lebih jauh, seperti Wilayah Afrika (Cape Town) (af-selatan-1). Adalah umum untuk melihat 0,5 hingga 2,5 detik latensi replikasi Wilayah AWS dalam satu hektometer dari satu sama lain.

Note

Latensi replikasi tidak mempengaruhi latensi API. Jika Anda memiliki klien dan tabel di Wilayah A dan menambahkan replika tabel global di Wilayah B, klien dan tabel di Wilayah A akan memiliki latensi yang sama seperti sebelum menambahkan Wilayah B. Jika Anda memanggil operasi [PutItem](#) API di Wilayah B, akhirnya akan tersedia untuk dibaca di Wilayah A setelah penundaan kira-kira `ReplicationLatency` statistik yang tersedia di Amazon. CloudWatch Sebelum direplikasi, Anda akan menerima respons kosong dan setelah direplikasi, Anda akan menerima item; kedua panggilan akan memiliki latensi API yang kira-kira sama.

Waktu Untuk Tayang (TTL)

Anda dapat menggunakan Waktu Untuk Tayang (TTL) untuk menentukan nama atribut yang nilainya menunjukkan waktu kedaluwarsa item tersebut. Nilai ini ditentukan sebagai angka dalam hitungan detik sejak awal jangka waktu Unix. Setelah itu, DynamoDB dapat menghapus item tersebut tanpa menimbulkan biaya tulis.

Dengan tabel global, Anda mengonfigurasi TTL di satu Wilayah, dan pengaturan tersebut otomatis direplikasi ke Wilayah lainnya. Ketika sebuah item dihapus melalui aturan TTL, pekerjaan tersebut dilakukan tanpa menggunakan Unit Tulis pada tabel sumber - tetapi tabel target akan dikenakan biaya Unit Tulis yang Direplikasi.

Perlu diketahui jika tabel sumber dan target memiliki kapasitas tulis yang disediakan yang sangat rendah, hal ini dapat menyebabkan throttling karena penghapusan TTL memerlukan kapasitas tulis.

Aliran dan transaksi dengan tabel global

Setiap tabel global menghasilkan aliran independen berdasarkan semua tulisannya, terlepas dari titik asal untuk penulisan tersebut. Anda dapat memilih untuk menggunakan aliran DynamoDB ini di satu Wilayah atau di semua Wilayah secara independen.

Jika Anda ingin penulisan lokal yang diproses tetapi bukan penulisan yang direplikasi, Anda dapat menambahkan atribut wilayah Anda sendiri ke setiap item. Kemudian, Anda dapat menggunakan filter peristiwa Lambda untuk hanya menginvokasi Lambda untuk penulisan di Wilayah lokal.

Operasi transaksional memberikan jaminan ACID (Atomicity, Consistency, Isolation, dan Durability) HANYA di Wilayah tempat awal tulis dilakukan. Transaksi tidak didukung di seluruh Wilayah dalam tabel global.

Misalnya, jika Anda memiliki tabel global dengan replika di Wilayah AS Timur (Ohio) dan AS Barat (Oregon) dan melakukan `TransactWriteItems` operasi di Wilayah AS Timur (Ohio), Anda dapat mengamati transaksi yang diselesaikan sebagian di Wilayah AS Barat (Oregon) saat perubahan direplikasi. Perubahan hanya akan direplikasi ke Wilayah lain setelah perubahan telah dilakukan di Wilayah sumber.

Note

- Tabel global “write-around” DynamoDB Accelerator dengan memperbarui DynamoDB secara langsung. Akibatnya, DAX tidak akan mengetahui jika sedang menyimpan data usang. Cache DAX hanya akan disegarkan ketika TTL cache kedaluwarsa.
- Tanda pada tabel global tidak disebarikan secara otomatis.

Throughput baca dan tulis

Tabel global mengelola throughput baca dan tulis dengan cara berikut.

- Kapasitas tulis harus sama di semua instans tabel di seluruh Wilayah.
- Dengan Versi 2019.11.21 (Terbaru), jika tabel diatur untuk mendukung penskalaan otomatis atau berada dalam mode sesuai permintaan, maka kapasitas tulis tetap sinkron secara otomatis. Artinya, perubahan kapasitas tulis pada satu tabel akan mereplikasi tabel lainnya.
- Kapasitas baca dapat berbeda antarWilayah karena baca mungkin tidak sama. Saat menambahkan replika global ke tabel, kapasitas Wilayah sumber disebarikan. Setelah pembuatan,

Anda dapat menyesuaikan kapasitas baca untuk satu replika, dan pengaturan baru ini tidak ditransfer ke sisi lain.

Konsistensi dan resolusi konflik

Setiap perubahan yang dibuat pada item apa pun di tabel replika mana pun akan direplikasi ke semua replika lain dalam tabel global yang sama. Dalam tabel global, item yang baru ditulis biasanya disebarkan ke semua tabel replika dalam hitungan detik.

Dengan tabel global, setiap tabel replika menyimpan set item data yang sama. DynamoDB tidak mendukung replikasi parsial hanya beberapa item.

Aplikasi dapat membaca dan menulis data ke tabel replika mana pun. Jika aplikasi Anda hanya menggunakan bacaan akhir konsisten dan hanya mengeluarkan aktivitas baca terhadap satu Wilayah AWS, aplikasi akan berjalan tanpa perubahan apa pun. Namun, jika aplikasi Anda membutuhkan bacaan sangat konsisten, aplikasi harus melakukan semua penulisan dan bacaan sangat konsisten di Wilayah yang sama. DynamoDB tidak mendukung bacaan sangat konsisten di seluruh Wilayah. Oleh karena itu, jika Anda menulis ke satu Wilayah dan membaca dari Wilayah lain, respons baca mungkin menyertakan data usang yang tidak mencerminkan hasil penulisan yang baru saja diselesaikan di Wilayah lain.

Jika aplikasi memperbarui item yang sama di Wilayah yang berbeda pada waktu yang sama, konflik dapat terjadi. Untuk membantu memastikan konsistensi akhir, tabel global DynamoDB menggunakan rekonsiliasi penulis terakhir menang antara pembaruan serentak, dengan DynamoDB melakukan upaya terbaik untuk menentukan penulis terakhir. Ini dilakukan di tingkat item. Dengan mekanisme resolusi konflik ini, semua replika akan menyetujui pembaruan terkini dan berkumpul menuju status ketika semua replika memiliki data yang identik.

Ketersediaan dan daya tahan

Jika satu Wilayah AWS menjadi terisolasi atau terdegradasi, aplikasi Anda dapat mengalihkan ke Wilayah yang berbeda dan melakukan aktivitas baca dan tulis terhadap tabel replika yang berbeda. Anda dapat menerapkan logika bisnis khusus untuk menentukan kapan harus mengalihkan permintaan ke Wilayah lain.

Jika suatu Wilayah menjadi terisolasi atau terdegradasi, DynamoDB melacak setiap penulisan yang telah dilakukan tetapi belum disebarkan ke semua tabel replika. Setelah Wilayah kembali online, DynamoDB melanjutkan penyebaran penulisan yang tertunda dari Wilayah tersebut ke tabel replika

di Wilayah lain. DynamoDB juga melanjutkan penyebaran penulisan dari tabel replika lain ke Wilayah yang saat ini kembali online.

Praktik terbaik dan persyaratan untuk mengelola tabel global

Dengan menggunakan tabel global Amazon DynamoDB, Anda dapat mereplikasi data tabel di AWS Wilayah. Tabel replika dan indeks sekunder di tabel global Anda harus memiliki pengaturan kapasitas tulis yang identik untuk memastikan replikasi data yang tepat.

Untuk kejelasan di masa mendatang, sebaiknya jangan mencantumkan Wilayah dalam nama tabel mana pun yang mungkin suatu saat akan diubah menjadi tabel global.

Warning

Nama tabel untuk setiap tabel global harus unik di dalam akun AWS Anda.

Versi tabel global

Untuk menentukan versi tabel global yang Anda gunakan, lihat [Menentukan versi tabel global yang Anda gunakan](#).

Persyaratan untuk Mengelola Kapasitas

Tabel global harus memiliki kapasitas throughput yang dikonfigurasi dengan salah satu dari dua cara berikut:

1. Mode kapasitas sesuai permintaan, diukur dalam unit permintaan tulis yang direplikasi (rWRU)
2. Mode kapasitas yang disediakan dengan penskalaan otomatis, diukur dalam unit kapasitas tulis yang direplikasi (RWCU)

Penggunaan mode kapasitas yang disediakan dengan penskalaan otomatis atau mode kapasitas sesuai permintaan membantu memastikan tabel global memiliki kapasitas yang memadai untuk melakukan replikasi tulis ke semua wilayah tabel global.

Note

Beralih dari satu mode kapasitas tabel ke mode kapasitas lainnya di Wilayah mana pun akan mengalihkan mode untuk semua replika.

Men-deploy tabel global

Di AWS CloudFormation, setiap tabel global dikontrol oleh tumpukan tunggal di satu Wilayah. Hal ini terlepas dari jumlah replika. Ketika Anda menerapkan template Anda, CloudFormation akan membuat/memperbarui semua replika sebagai bagian dari operasi tumpukan tunggal. Oleh karena itu, Anda tidak boleh men-deploy sumber daya AWS::DynamoDB::GlobalTable yang sama di beberapa Wilayah. Tindakan tersebut tidak didukung dan akan mengakibatkan kesalahan.

Jika men-deploy templat aplikasi di beberapa Wilayah, Anda dapat menggunakan syarat untuk membuat sumber daya hanya di satu Wilayah. Alternatifnya, Anda dapat memilih untuk menentukan sumber daya AWS::DynamoDB::GlobalTable Anda dalam tumpukan yang terpisah dari tumpukan aplikasi Anda dan memastikan sumber daya tersebut hanya disebar ke satu Wilayah. Untuk informasi selengkapnya, lihat [tabel Global di CloudFormation](#)

Tabel DynamoDB disebut sebagai AWS::DynamoDB::Table, dan tabel global adalah AWS::DynamoDB::GlobalTable. CloudFormation Sejauh menyangkut, ini pada dasarnya menjadikan mereka dua sumber daya yang berbeda. Dengan demikian, salah satu pendekatannya adalah membuat semua tabel yang mungkin bersifat global dengan menggunakan konstruksi GlobalTable. Anda kemudian dapat menyimpannya sebagai tabel mandiri untuk memulai, dan menambahkannya nanti ke Wilayah jika diperlukan.

Jika Anda memiliki tabel biasa dan Anda ingin mengonversinya saat menggunakan CloudFormation, metode yang disarankan adalah:

1. Menetapkan kebijakan penghapusan untuk dipertahankan.
2. Menghapus tabel dari tumpukan.
3. Mengonversi tabel menjadi Tabel Global di konsol.
4. Mengimpor tabel global sebagai sumber daya baru ke tumpukan.

Note

Replikasi lintas akun tidak didukung saat ini.

Menggunakan tabel global untuk membantu menangani potensi gangguan Wilayah

Memiliki atau dapat dengan cepat membuat salinan independen dari tumpukan eksekusi Anda di Wilayah alternatif, masing-masing mengakses titik akhir DynamoDB lokalnya.

Gunakan Route53 atau AWS Global Accelerator untuk merutekan ke Wilayah sehat terdekat. Sebagai alternatif, buat klien mengetahui beberapa titik akhir yang mungkin digunakannya.

Gunakan pemeriksaan kondisi di setiap Wilayah yang akan dapat menentukan secara andal apakah ada masalah dengan tumpukan, termasuk apakah DynamoDB mengalami degradasi. Misalnya, jangan hanya melakukan ping bahwa titik akhir DynamoDB sudah aktif. Lakukan panggilan untuk memastikan aliran basis data berhasil sepenuhnya.

Jika pemeriksaan kondisi gagal, lalu lintas dapat merutekan ke Wilayah lain (dengan memperbarui entri DNS dengan Route53, dengan merutekan Akselerator Global secara berbeda, atau dengan meminta klien memilih titik akhir yang berbeda). Tabel global memiliki RPO (sasaran titik pemulihan) yang baik karena data terus disinkronkan dan RTO yang baik (sasaran waktu pemulihan) karena kedua Wilayah selalu menyiapkan tabel untuk lalu lintas baca dan tulis.

Untuk informasi selengkapnya tentang pemeriksaan kondisi, lihat [Jenis pemeriksaan kondisi](#).

Note

DynamoDB adalah layanan inti tempat layanan lain sering membangun operasi bidang kontrolnya, sehingga kecil kemungkinannya Anda akan menghadapi skenario ketika DynamoDB mengalami penurunan layanan di suatu Wilayah sementara layanan lain tidak terkena dampak.

Mencadangkan tabel global

Saat mencadangkan tabel global, pencadangan tabel di satu Wilayah sudah cukup dan mencadangkan semua tabel di semua Wilayah seharusnya tidak diperlukan. Jika tujuannya adalah untuk dapat memulihkan data yang terhapus atau diubah secara keliru, maka PITR di satu Wilayah sudah cukup. Demikian pula, ketika menyimpan snapshot untuk tujuan historis seperti persyaratan peraturan, maka pencadangan di satu Wilayah sudah cukup. Data yang dicadangkan dapat direplikasi ke beberapa Wilayah melalui AWS Backup.

Replika dan menghitung unit tulis

Untuk perencanaan, Anda harus mengambil jumlah penulisan yang akan dilakukan oleh satu Wilayah dan menambahkannya ke jumlah penulisan yang terjadi di setiap Wilayah lainnya. Ini sangat penting karena setiap penulisan yang dilakukan di satu Wilayah juga harus dilakukan di setiap Wilayah

replika. Jika Anda tidak memiliki kapasitas yang memadai untuk menangani semua penulisan, pengecualian kapasitas akan terjadi. Selain itu, waktu tunggu replikasi antarwilayah akan meningkat.

Misalnya, anggaplah Anda mengharapkan 5 aktivitas tulis per detik untuk tabel replika Anda di Ohio, 10 aktivitas tulis per detik untuk tabel replika Anda di Virginia Utara, dan 5 aktivitas tulis per detik untuk tabel replika Anda di Irlandia. Dalam hal ini, Anda akan mengonsumsi 20 rWCU atau rWRU di setiap Wilayah: Ohio, Virginia Utara, dan Irlandia. Dengan kata lain, Anda akan mengonsumsi total 60 rWCU di ketiga Wilayah.

Untuk detail tentang kapasitas yang disediakan dengan penskalaan otomatis dan DynamoDB, lihat [Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB](#).

Note

Jika tabel berjalan dalam mode kapasitas yang disediakan dengan penskalaan otomatis, kapasitas tulis yang disediakan diperbolehkan untuk mengambang dalam pengaturan penskalaan otomatis tersebut untuk setiap Wilayah.

Tutorial: Membuat tabel global

Bagian ini menjelaskan cara membuat tabel menggunakan konsol Amazon DynamoDB atau AWS Command Line Interface (AWS CLI).

Topik

- [Membuat tabel global \(konsol\)](#)
- [Membuat tabel global \(AWS CLI\)](#)
- [Membuat tabel global \(Java\)](#)

Membuat tabel global (konsol)

Ikuti langkah-langkah ini untuk membuat tabel global menggunakan konsol. Contoh berikut membuat tabel global dengan tabel replika di Amerika Serikat dan Eropa.

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/home>. Untuk contoh ini, pilih Wilayah AS Timur (Ohio).
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih Buat Tabel.

- a. Untuk Nama tabel, masukkan **Music**.
- b. Untuk Kunci partisi masukkan **Artist**. Untuk Kunci urutan, masukkan **SongTitle**. (**Artist** dan **SongTitle** harus berupa string.)

Untuk membuat tabel, pilih Buat tabel. Tabel ini berfungsi sebagai tabel replika pertama dalam tabel global baru. Ini adalah prototipe untuk tabel replika lain yang Anda tambahkan nantinya.

4. Pilih tab Tabel Global, lalu pilih Buat replika.
5. Dari dropdown Wilayah replikasi yang Tersedia, pilih AS Barat (Oregon).

Konsol memeriksa untuk memastikan bahwa tidak ada tabel dengan nama yang sama di Wilayah yang dipilih. Jika ada tabel dengan nama yang sama, Anda harus menghapus tabel yang ada sebelum dapat membuat tabel replika baru di Wilayah tersebut.

6. Pilih Buat Replika. Langkah ini memulai proses pembuatan tabel di AS Barat (Oregon);.

Tab Tabel Global untuk tabel yang dipilih (dan untuk setiap tabel replika lainnya) menunjukkan bahwa tabel telah direplikasi di beberapa Wilayah.

7. Sekarang tambahkan Wilayah lain sehingga tabel global Anda direplikasi dan disinkronkan di seluruh Amerika Serikat dan Eropa. Untuk melakukannya, ulangi langkah 5, tetapi kali ini tentukan Eropa (Frankfurt), bukan AS Barat (Oregon).
8. Anda masih harus menggunakan AWS Management Console di Wilayah AS Timur (Ohio). Pilih Item di menu navigasi kiri, pilih tabel Musik, lalu pilih Buat Item.
 - a. Untuk Artist, masukkan **item_1**.
 - b. Untuk SongTitle, masukkan **Song Value 1**.
 - c. Untuk menulis item, pilih Buat item.
9. Setelah beberapa saat, item direplikasi di ketiga Wilayah tabel global Anda. Untuk memverifikasi ini, di pemilih Wilayah di sudut kanan atas di konsol, pilih Eropa (Frankfurt). Tabel Music di Eropa (Frankfurt) akan berisi item baru.
10. Ulangi langkah 9 dan pilih AS Barat (Oregon) untuk memverifikasi replikasi di wilayah itu.

Membuat tabel global (AWS CLI)

Ikuti langkah-langkah ini untuk membuat tabel global Music menggunakan AWS CLI. Contoh berikut membuat tabel global, dengan tabel replika di Amerika Serikat dan Eropa.

1. Buat tabel baru (Music) di AS Timur (Ohio), dengan DynamoDB Streams diaktifkan (NEW_AND_OLD_IMAGES).

```
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 \  
  --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \  
  --region us-east-2
```

2. Buat tabel Music identik di AS Timur (Virginia Utara).

```
aws dynamodb update-table --table-name Music --cli-input-json \  
'{  
  "ReplicaUpdates":  
  [  
    {  
      "Create": {  
        "RegionName": "us-east-1"  
      }  
    }  
  ]  
}' \  
--region=us-east-2
```

3. Ulangi langkah 2 untuk membuat tabel di Eropa (Irlandia) (eu-west-1).
4. Anda dapat melihat daftar replika yang dibuat menggunakan describe-table.

```
aws dynamodb describe-table --table-name Music --region us-east-2
```

5. Untuk memverifikasi bahwa replikasi berfungsi, tambahkan item baru ke tabel Music di AS Timur (Ohio).

```
aws dynamodb put-item \  
  --table-name Music \  
  --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \  
  --region us-east-2
```

```
--region us-east-2
```

6. Tunggu beberapa detik, lalu periksa untuk melihat apakah item tersebut telah berhasil direplikasi ke AS Timur (Virginia Utara) dan Eropa (Irlandia).

```
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
```

7. Hapus tabel replika di Wilayah Eropa (Irlandia).

```
aws dynamodb update-table --table-name Music --cli-input-json \  
'{  
  "ReplicaUpdates":  
  [  
    {  
      "Delete": {  
        "RegionName": "eu-west-1"  
      }  
    }  
  ]  
'
```

Membuat tabel global (Java)

Berikut sampel kode java, buat tabel `Music` di wilayah Eropa (Irlandia), lalu buat replika di wilayah Asia Pasifik (Seoul).

```
package com.amazonaws.codesamples.gtv2  
import java.util.logging.Logger;  
import com.amazonaws.auth.profile.ProfileCredentialsProvider;  
import com.amazonaws.regions.Regions;  
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;  
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;  
import com.amazonaws.services.dynamodbv2.model.AmazonDynamoDBException;
```

```
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.BillingMode;
import com.amazonaws.services.dynamodbv2.model.CreateReplicationGroupMemberAction;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableRequest;
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.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughputOverride;
import com.amazonaws.services.dynamodbv2.model.ReplicaGlobalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.ReplicationGroupUpdate;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.services.dynamodbv2.model.StreamSpecification;
import com.amazonaws.services.dynamodbv2.model.StreamViewType;
import com.amazonaws.services.dynamodbv2.model.UpdateTableRequest;
import com.amazonaws.waiters.WaiterParameters;

public class App
{
    private final static Logger LOGGER = Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);

    public static void main( String[] args )
    {

        String tableName = "Music";
        String indexName = "index1";

        Regions calledRegion = Regions.EU_WEST_1;
        Regions destRegion = Regions.AP_NORTHEAST_2;

        AmazonDynamoDB ddbClient = AmazonDynamoDBClientBuilder.standard()
            .withCredentials(new ProfileCredentialsProvider("default"))
            .withRegion(calledRegion)
            .build();

        LOGGER.info("Creating a regional table - TableName: " + tableName + ",
IndexName: " + indexName + " .....");
        ddbClient.createTable(new CreateTableRequest()
            .withTableName(tableName)
```

```
        .withAttributeDefinitions(
            new AttributeDefinition()

.withAttributeName("Artist").withAttributeType(ScalarAttributeType.S),
            new AttributeDefinition()

.withAttributeName("SongTitle").withAttributeType(ScalarAttributeType.S))
        .withKeySchema(
            new
KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH),
            new
KeySchemaElement().withAttributeName("SongTitle").withKeyType(KeyType.RANGE))
        .withBillingMode(BillingMode.PAY_PER_REQUEST)
        .withGlobalSecondaryIndexes(new GlobalSecondaryIndex()
            .withIndexName(indexName)
            .withKeySchema(new KeySchemaElement()
                .withAttributeName("SongTitle")
                .withKeyType(KeyType.HASH))
            .withProjection(new
Projection().withProjectionType(ProjectionType.ALL)))
        .withStreamSpecification(new StreamSpecification()
            .withStreamEnabled(true)
            .withStreamViewType(StreamViewType.NEW_AND_OLD_IMAGES));

    LOGGER.info("Waiting for ACTIVE table status .....");
    ddbClient.waiters().tableExists().run(new WaiterParameters<>(new
DescribeTableRequest(tableName)));

    LOGGER.info("Testing parameters for adding a new Replica in " + destRegion +
" .....");

    CreateReplicationGroupMemberAction createReplicaAction = new
CreateReplicationGroupMemberAction()
        .withRegionName(destRegion.getName())
        .withGlobalSecondaryIndexes(new ReplicaGlobalSecondaryIndex()
            .withIndexName(indexName)
            .withProvisionedThroughputOverride(new
ProvisionedThroughputOverride()
                .withReadCapacityUnits(15L)));

    ddbClient.updateTable(new UpdateTableRequest()
        .withTableName(tableName))
```

```
        .withReplicaUpdates(new ReplicationGroupUpdate()
            .withCreate(createReplicaAction.withKMSMasterKeyId(null)))));
    }
}
```

Memantau tabel global

Anda dapat menggunakan Amazon CloudWatch untuk memantau perilaku dan kinerja tabel global. Amazon DynamoDB menerbitkan metrik `ReplicationLatency` untuk setiap replika dalam tabel global.

- **ReplicationLatency**—Waktu yang berlalu antara saat item ditulis ke tabel replika, dan saat item tersebut muncul di replika lain dalam tabel global. `ReplicationLatency` dinyatakan dalam milidetik dan dipancarkan untuk setiap pasangan Wilayah sumber dan tujuan.

Selama operasi normal, `ReplicationLatency` akan cukup konstan. Nilai `ReplicationLatency` yang tinggi dapat menunjukkan bahwa pembaruan dari satu replika tidak disebarkan ke tabel replika lain pada waktu yang tepat. Seiring waktu, hal ini dapat mengakibatkan tabel replika lainnya tertinggal karena tidak lagi menerima pembaruan secara konsisten. Dalam kasus ini, Anda harus memverifikasi bahwa unit kapasitas baca (RCU) dan unit kapasitas tulis (WCU) identik untuk masing-masing tabel replika. Selain itu, saat memilih pengaturan WCU, ikuti rekomendasi di [Versi tabel global](#).

`ReplicationLatency` dapat meningkat jika Wilayah AWS terdegradasi dan Anda memiliki tabel replika di wilayah tersebut. Dalam kasus ini, Anda dapat mengalihkan aktivitas baca dan tulis aplikasi Anda ke Wilayah AWS lain untuk sementara waktu.

Lihat informasi yang lebih lengkap di [Dimensi dan Metrik DynamoDB](#).

Menggunakan IAM dengan tabel global

Ketika Anda membuat tabel global untuk pertama kalinya, Amazon DynamoDB otomatis membuat peran tertaut layanan AWS Identity and Access Management (IAM) untuk Anda. Peran ini bernama [AWSServiceRoleForDynamoDBReplication](#), dan memungkinkan DynamoDB untuk mengelola

replikasi lintas-Wilayah untuk tabel global atas nama Anda. Jangan hapus peran tertaut layanan ini. Jika Anda melakukannya, semua tabel global Anda tidak akan lagi berfungsi.

Untuk informasi selengkapnya tentang peran tertaut layanan, lihat [Menggunakan peran tertaut layanan](#) di Panduan Pengguna IAM.

Untuk membuat tabel replika di DynamoDB, Anda harus memiliki izin berikut di wilayah sumber.

- `dynamodb:UpdateTable`

Untuk membuat tabel replika di DynamoDB, Anda harus memiliki izin berikut di wilayah tujuan.

- `dynamodb:CreateTable`
- `dynamodb:CreateTableReplica`
- `dynamodb:Scan`
- `dynamodb:Query`
- `dynamodb:UpdateItem`
- `dynamodb:PutItem`
- `dynamodb:GetItem`
- `dynamodb>DeleteItem`
- `dynamodb:BatchWriteItem`

Untuk menghapus tabel replika di DynamoDB, Anda harus memiliki izin berikut di wilayah tujuan.

- `dynamodb>DeleteTable`
- `dynamodb>DeleteTableReplica`

Untuk memperbarui kebijakan penskalaan otomatis replikaUpdateTableReplicaAutoScaling, Anda harus memiliki izin berikut di semua Wilayah tempat replika tabel ada

- `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`

Untuk menggunakan `UpdateTimeToLive` Anda harus memiliki izin untuk `dynamodb:UpdateTimeToLive` di semua Wilayah tempat replika tabel berada.

Contoh: Menambahkan replika

Kebijakan IAM berikut memberikan izin untuk memungkinkan Anda menambahkan replika ke tabel global.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:DescribeTable",
        "dynamodb:UpdateTable",
        "dynamodb:CreateTableReplica",
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": "*"
    }
  ]
}
```

Contoh: AutoScaling Kebijakan pembaruan

Kebijakan IAM berikut memberikan izin untuk memungkinkan Anda memperbarui kebijakan penskalaan otomatis replika.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```

{
  "Sid": "VisualEditor0",
  "Effect": "Allow",
  "Action": [
    "application-autoscaling:RegisterScalableTarget",
    "application-autoscaling:DeleteScheduledAction",
    "application-autoscaling:DescribeScalableTargets",
    "application-autoscaling:DescribeScalingActivities",
    "application-autoscaling:DescribeScalingPolicies",
    "application-autoscaling:PutScalingPolicy",
    "application-autoscaling:DescribeScheduledActions",
    "application-autoscaling>DeleteScalingPolicy",
    "application-autoscaling:PutScheduledAction",
    "application-autoscaling:DeregisterScalableTarget"
  ],
  "Resource": "*"
}

```

Contoh: Memungkinkan pembuatan replika untuk wilayah dan nama tabel tertentu

Kebijakan IAM berikut memberikan izin yang memungkinkan pembuatan tabel dan replika untuk tabel Customers dengan replika di tiga Wilayah.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:DescribeTable",
        "dynamodb:UpdateTable"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/Customers",
        "arn:aws:dynamodb:us-west-1:123456789012:table/Customers",
        "arn:aws:dynamodb:eu-east-2:123456789012:table/Customers"
      ]
    }
  ]
}

```

```
]
}
```

Menentukan versi tabel global yang Anda gunakan

Ada dua versi tabel global DynamoDB yang tersedia: [Tabel Global versi 2019.11.21 \(Saat Ini\)](#) dan [Versi tabel global 2017.11.29 \(Legacy\)](#). Kami merekomendasikan menggunakan [Tabel Global versi 2019.11.21 \(Saat Ini\)](#). Ini lebih efisien dan mengonsumsi lebih sedikit kapasitas tulis daripada [Versi tabel global 2017.11.29 \(Legacy\)](#). Keuntungan dari versi terbaru meliputi:

- Tabel sumber dan target dipertahankan bersama-sama dan tetap selaras secara otomatis untuk throughput, pengaturan TTL, pengaturan penskalaan otomatis, dan atribut berguna lainnya.
- Indeks sekunder global juga tetap selaras.
- Anda dapat menambahkan tabel replika baru secara dinamis dari tabel yang diisi dengan data
- Atribut metadata yang diperlukan untuk mengontrol replikasi disembunyikan sehingga membantu mencegah penulisan atribut tersebut yang dapat menyebabkan masalah pada replikasi.
- Versi terbaru mendukung lebih banyak Wilayah daripada versi lama, dan memungkinkan Anda menambahkan atau menghapus Wilayah pada tabel yang ada, sedangkan versi lama tidak.
- [Versi Tabel Global 2019.11.21 \(Saat Ini\)](#) lebih efisien dan mengonsumsi kapasitas tulis lebih sedikit daripada [Versi tabel global 2017.11.29 \(Legacy\)](#), dan karenanya lebih hemat biaya. Secara spesifik:
 - Memasukkan item baru di satu Wilayah, lalu mereplikasi ke Wilayah lain memerlukan 2 rWCU per wilayah untuk Versi 2017.11.29 (Lama), tetapi hanya 1 untuk Versi 2019.11.21 (Terbaru).
 - Memperbarui item memerlukan 2 rWCU di Wilayah sumber dan 1 rWCU per Wilayah tujuan di Versi 2017.11.29 (Lama), tetapi hanya 1 rWCU per sumber atau tujuan di Versi 2019.11.21 (Terbaru).
 - Menghapus item memerlukan 1 rWCU di Wilayah sumber dan 2 rWCU per Wilayah tujuan di Versi 2017.11.29 (Lama), tetapi hanya 1 rWCU per sumber atau tujuan di Versi 2019.11.21 (Terbaru).

Untuk informasi selengkapnya, lihat [Haraga Amazon DynamoDB](#).

Menentukan versi melalui CLI

Untuk mengetahui versi tabel global mana yang Anda gunakan melalui AWS CLI, periksa `DescribeTable` dan `DescribeGlobalTable`. `DescribeTable` akan menampilkan versi tabel

jika itu adalah Versi 2019.11.21 (Saat ini), dan DescribeGlobalTable properti akan menampilkan versi tabel jika itu adalah Versi 2017.11.29 (Legacy).

Menentukan versi melalui konsol

Menemukan versi melalui konsol

Untuk mengetahui versi tabel global yang Anda gunakan melalui konsol, lakukan langkah berikut:

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/home>.
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tabel yang ingin Anda gunakan.
4. Pilih tab Tabel Global.
5. Versi tabel global menampilkan versi tabel global yang sedang digunakan:

 You are using global tables version 2019.11.21. If you need to use version 2017.11.29 instead, choose "Create version 2017.11.29 replica." You can create a version 2017.11.29 replica only if you have an empty table.

Create a version 2017.11.29 replica.

Untuk meningkatkan dari tabel global Versi 2017.11.29 (Lama) ke Versi 2019.11.21 (Terbaru), ikuti langkah-langkahnya [di sini](#). Proses peningkatan secara keseluruhan akan berjalan tanpa mengganggu tabel langsung, dan akan selesai dalam waktu kurang dari satu jam. Untuk informasi selengkapnya, lihat [Memperbarui ke Versi 2019.11.21 \(Terbaru\)](#)

Note

- Jika pesan Versi tabel global tidak muncul di konsol, artinya ada tabel lain dengan nama yang sama di Wilayah lain. Dalam hal ini, tabel saat ini tidak dapat dibuat menjadi tabel global. Tabel saat ini harus disalin ke tabel baru dengan nama unik, atau semua tabel lain dengan nama yang sama harus dihapus.
- Jika Anda menggunakan [Global Tables versi 2019.11.21 \(Saat ini\)](#) dari tabel global dan Anda juga menggunakan fitur [Time to Live](#), DynamoDB mereplikasi penghapusan TTL ke

semua tabel replika. Penghapusan TTL awal tidak mengonsumsi kapasitas tulis di wilayah tempat terjadinya kedaluwarsa TTL. Namun, penghapusan TTL yang direplikasi untuk tabel replika mengonsumsi unit kapasitas tulis yang direplikasi ketika menggunakan kapasitas yang disediakan, atau kapasitas tulis yang direplikasi ketika menggunakan mode kapasitas sesuai permintaan, di masing-masing wilayah replika dan biaya akan berlaku.

- Dalam [Tabel Global versi 2019.11.21 \(Saat ini\)](#), ketika penghapusan TTL terjadi, itu direplikasi ke semua wilayah replika. Tulisan yang direplikasi ini tidak mengandung properti `type` atau `principalID`. Hal ini dapat menyulitkan untuk membedakan penghapusan TTL dari penghapusan pengguna dalam tabel yang direplikasi.

Memutakhirkan tabel global ke versi Saat Ini (2019.11.21) dari versi Legacy (2017.11.29)

Ada dua versi tabel global DynamoDB yang tersedia: [Tabel Global versi 2019.11.21 \(Saat Ini\)](#) dan [Versi tabel global 2017.11.29 \(Legacy\)](#). Pelanggan harus menggunakan versi 2019.11.21 (Terbaru) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Lama). Untuk menentukan versi yang Anda gunakan, lihat [Menentukan versi tabel global yang Anda gunakan](#).

Bagian ini menjelaskan cara memutakhirkan tabel global Anda ke Versi 2019.11.21 (Saat ini) menggunakan konsol DynamoDB. Memutakhirkan dari Versi 2017.11.29 (Legacy) ke Versi 2019.11.21 (Saat Ini) adalah tindakan satu kali dan Anda tidak dapat membalikkannya. Saat ini, Anda dapat memutakhirkan tabel global hanya menggunakan konsol.

Topik

- [Perbedaan perilaku antara versi Legacy dan Current](#)
- [Prasyarat peningkatan](#)
- [Izin yang diperlukan untuk peningkatan tabel global](#)
- [Apa yang diharapkan selama upgrade](#)
- [Perilaku DynamoDB Streams sebelum, selama, dan setelah peningkatan](#)
- [Memutakhirkan ke Versi 2019.11.21 \(Saat ini\)](#)

Perbedaan perilaku antara versi Legacy dan Current

Daftar berikut menjelaskan perbedaan perilaku antara versi Legacy dan Current dari tabel global.

- Versi 2019.11.21 (Saat ini) mengkonsumsi lebih sedikit kapasitas tulis untuk beberapa operasi DynamoDB dibandingkan dengan Versi 2017.11.29 (Legacy), dan oleh karena itu, lebih hemat biaya bagi sebagian besar pelanggan. Perbedaan untuk operasi DynamoDB ini adalah sebagai berikut:
 - [PutItem](#) Memanggil item 1KB di Wilayah dan mereplikasi ke Wilayah lain memerlukan 2 rWRU per wilayah untuk 2017.11.29 (Warisan), tetapi hanya 1 rWRU untuk 2019.11.21 (Saat Ini).
 - [UpdateItem](#) Memanggil item 1KB membutuhkan 2 rWRU di Wilayah sumber dan 1 RWRU per Wilayah tujuan untuk 2017.11.29 (Warisan), tetapi hanya 1 rWRU untuk Wilayah sumber dan tujuan untuk 2019.11.21 (Saat Ini).
 - [DeleteItem](#) Memanggil item 1KB membutuhkan 1 rWRU di Wilayah sumber dan 2 rWRU per Wilayah tujuan untuk 2017.11.29 (Warisan), tetapi hanya 1 rWRU untuk Wilayah sumber atau tujuan untuk 2019.11.21 (Saat Ini).

Tabel berikut menunjukkan konsumsi RWRU untuk tabel 2017.11.29 (Legacy) dan 2019.11.21 (Saat Ini).

Konsumsi RWRU 2017.11.29 (Legacy) dan 2019.11.21 (Saat Ini) tabel untuk item 1KB di dua Wilayah

Operasi	2017.11.29 (Warisan)	2019.11.21 (Saat ini)	Penghematan
PutItem	4 RWRU	2 RWRU	50%
UpdateItem	3 RWRU	2 RWRU	33%
DeleteItem	3 RWRU	2 RWRU	33%

- Versi 2017.11.29 (Legacy) hanya tersedia di 11. Wilayah AWS Namun, Versi 2019.11.21 (Saat Ini) tersedia di semua. Wilayah AWS
- Anda membuat tabel global Versi 2017.11.29 (Legacy) dengan terlebih dahulu membuat satu set tabel Regional kosong, lalu memanggil [CreateGlobalTable](#) API untuk membentuk tabel global. Anda membuat tabel global Versi 2019.11.21 (Saat ini) dengan menjalankan [UpdateTable](#) API untuk menambahkan replika ke tabel Regional yang ada.

- Versi 2017.11.29 (Legacy) mengharuskan Anda untuk mengosongkan semua replika dalam tabel sebelum menambahkan replika di Wilayah baru (termasuk selama pembuatan). Versi 2019.11.21 (Saat ini) mendukung Anda untuk menambah dan menghapus replika ke Wilayah pada tabel yang sudah berisi data.
- Versi 2017.11.29 (Legacy) menggunakan set khusus API bidang kontrol berikut untuk mengelola replika:
 - [CreateGlobalTable](#)
 - [DescribeGlobalTable](#)
 - [DescribeGlobalTableSettings](#)
 - [ListGlobalTables](#)
 - [UpdateGlobalTable](#)
 - [UpdateGlobalTableSettings](#)

Versi 2019.11.21 (Saat ini) menggunakan [UpdateTable](#) API [DescribeTable](#) dan untuk mengelola replika.

- Versi 2017.11.29 (Legacy) menerbitkan dua catatan DynamoDB Streams untuk setiap penulisan. Versi 2019.11.21 (Saat ini) hanya menerbitkan satu catatan DynamoDB Streams untuk setiap penulisan.
- Versi 2017.11.29 (Legacy) mengisi dan memperbarui, dan atribut. `aws:rep:deleting` `aws:rep:updateregion` `aws:rep:updatetime` Versi 2019.11.21 (Saat ini) tidak mengisi atau memperbarui atribut ini.
- Versi 2017.11.29 (Legacy) tidak menyinkronkan [Waktu untuk Hidup \(TTL\)](#) pengaturan di seluruh replika. Versi 2019.11.21 (Saat ini) menyinkronkan pengaturan TTL di seluruh replika.
- Versi 2017.11.29 (Legacy) tidak mereplikasi penghapusan TTL ke replika lain. Versi 2019.11.21 (Saat ini) mereplikasi penghapusan TTL ke semua replika.
- Versi 2017.11.29 (Legacy) tidak menyinkronkan pengaturan penskalaan [otomatis](#) di seluruh replika. Versi 2019.11.21 (Saat ini) menyinkronkan pengaturan penskalaan otomatis di seluruh replika.
- Versi 2017.11.29 (Legacy) tidak menyinkronkan pengaturan [indeks sekunder global \(GSI\)](#) di seluruh replika. Versi 2019.11.21 (Saat ini) menyinkronkan pengaturan GSI di seluruh replika.
- Versi 2017.11.29 (Legacy) tidak menyinkronkan [enkripsi pada pengaturan istirahat di](#) seluruh replika. Versi 2019.11.21 (Saat ini) menyinkronkan enkripsi pada pengaturan istirahat di seluruh replika.

- Versi 2017.11.29 (Legacy) menerbitkan metrik. PendingReplicationCount Versi 2019.11.21 (Saat ini) tidak mempublikasikan metrik ini.

Prasyarat peningkatan

Sebelum Anda mulai meningkatkan ke Versi 2019.11.21 (Saat ini) tabel global, Anda harus memenuhi prasyarat berikut:

- [Waktu untuk Hidup \(TTL\)](#) pengaturan pada replika konsisten di seluruh Wilayah.
- Definisi [indeks sekunder global \(GSI\)](#) pada replika konsisten di seluruh Wilayah.
- [Enkripsi pada pengaturan istirahat](#) pada replika konsisten di seluruh Wilayah.
- DynamoDB auto scaling diaktifkan untuk WCU untuk semua replika, atau mode kapasitas sesuai permintaan diaktifkan untuk semua replika.
- Aplikasi tidak memerlukan kehadiran dari `aws:rep:deleting`, `aws:rep:updateregion`, dan `aws:rep:updatetime` atribut dalam item tabel.

Izin yang diperlukan untuk peningkatan tabel global

Untuk meningkatkan ke Versi 2019.11.21 (Saat Ini), Anda harus memiliki `dynamodb:UpdateGlobalTableVersion` izin di semua Wilayah dengan replika. Izin ini diperlukan selain izin yang diperlukan untuk mengakses konsol DynamoDB dan tabel tampilan.

Kebijakan IAM berikut memberikan izin untuk memutakhirkan tabel global apa pun ke Versi 2019.11.21 (Saat Ini).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:UpdateGlobalTableVersion",
      "Resource": "*"
    }
  ]
}
```

Kebijakan IAM berikut memberikan izin untuk memutakhirkan hanya tabel Music global dengan replika di dua Wilayah ke Versi 2019.11.21 (Saat Ini).

```
{
  "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"
      ]
    }
  ]
}
```

Apa yang diharapkan selama upgrade

- Semua replika tabel global akan terus memproses lalu lintas baca dan tulis saat meningkatkan.
- Proses upgrade membutuhkan antara beberapa menit hingga beberapa jam tergantung pada ukuran tabel dan jumlah replika.
- Selama proses upgrade, nilai [TableStatus](#) akan berubah dari ACTIVE keUPDATING. Anda dapat melihat status tabel dengan menjalankan [DescribeTable](#) API, atau dengan tampilan Tabel di konsol [DynamoDB](#).
- Penskalaan otomatis tidak akan menyesuaikan pengaturan kapasitas yang disediakan untuk tabel global saat tabel sedang ditingkatkan. Kami sangat menyarankan Anda mengatur tabel ke mode kapasitas [sesuai permintaan](#) selama peningkatan.
- Jika Anda memilih untuk menggunakan mode kapasitas [yang disediakan dengan](#) penskalaan otomatis selama peningkatan, Anda harus meningkatkan throughput baca dan tulis minimum pada kebijakan Anda untuk mengakomodasi peningkatan lalu lintas yang diharapkan untuk menghindari pembatasan selama peningkatan.
- Ketika proses upgrade selesai, status tabel Anda akan berubah menjadiACTIVE.

Perilaku DynamoDB Streams sebelum, selama, dan setelah peningkatan

Operasi	Wilayah Replika	Perilaku sebelum upgrade	Perilaku selama peningkatan	Perilaku setelah peningkatan
Masukkan atau Perbarui	Sumber	Populasi stempel waktu terjadi menggunakan. UpdateItem	Populasi stempel waktu terjadi menggunakan. PutItem	Tidak ada stempel waktu yang terlihat pelanggan yang dihasilkan.
		Dua catatan Streams dihasilkan. Catatan pertama berisi atribut tertulis pelanggan. Catatan kedua berisi <code>aws:rep:*</code> atribut.	Dua catatan Streams dihasilkan. Catatan pertama berisi atribut tertulis pelanggan. Catatan kedua berisi <code>aws:rep:*</code> atribut.	Catatan Streams tunggal dihasilkan yang berisi atribut yang ditulis pelanggan.
		Dua RWCu dikonsumsi untuk setiap pelanggan menulis.	Dua RWCu dikonsumsi untuk setiap pelanggan menulis.	Satu RWCu dikonsumsi untuk setiap pelanggan menulis.
		ReplicationLatency dan PendingReplicationCount metrik dipublikasikan di CloudWatch.	ReplicationLatency dan PendingReplicationCount metrik dipublikasikan di CloudWatch.	ReplicationLatency metrik diterbitkan di CloudWatch.

Operasi	Wilayah Replika	Perilaku sebelum upgrade	Perilaku selama peningkatan	Perilaku setelah peningkatan
	Tujuan	Replikasi terjadi dengan menggunakan PutItem.	Replikasi terjadi dengan menggunakan PutItem.	Replikasi terjadi dengan menggunakan PutItem.
		Catatan Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan dan atribut. <code>aws:rep:*</code>	Catatan Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan dan atribut. <code>aws:rep:*</code>	Rekaman Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan saja dan tidak ada atribut replikasi.
		Satu RWCu dikonsumsi jika item tersebut ada di Wilayah tujuan. Dua RWCU dikonsumsi jika item tidak ada di Wilayah tujuan.	Satu RWCu dikonsumsi jika item tersebut ada di Wilayah tujuan. Dua RWCU dikonsumsi jika item tidak ada di Wilayah tujuan.	Satu RWCu dikonsumsi untuk setiap pelanggan menulis.
		ReplicationLatency dan PendingReplicationCount metrik dipublikasikan di CloudWatch.	ReplicationLatency dan PendingReplicationCount metrik dipublikasikan di CloudWatch.	ReplicationLatency metrik diterbitkan di CloudWatch.

Operasi	Wilayah Replika	Perilaku sebelum upgrade	Perilaku selama peningkatan	Perilaku setelah peningkatan
Hapus	Sumber	Hapus item apa pun dengan stempel waktu yang lebih kecil menggunakan. DeleteItem	Hapus item apa pun dengan stempel waktu yang lebih kecil menggunakan. DeleteItem	Hapus item apa pun dengan stempel waktu yang lebih kecil menggunakan. DeleteItem
		Catatan Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan dan atribut. <code>aws:rep:*</code>	Catatan Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan dan atribut. <code>aws:rep:*</code>	Rekaman Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan.
		Satu RWCu dikonsumsi untuk setiap penghapusan pelanggan.	Satu RWCu dikonsumsi untuk setiap penghapusan pelanggan.	Satu RWCu dikonsumsi untuk setiap penghapusan pelanggan.
		ReplicationLatency dan PendingReplicationCount metrik dipublikasikan di CloudWatch.	ReplicationLatency dan PendingReplicationCount metrik dipublikasikan di CloudWatch.	ReplicationLatency metrik diterbitkan di CloudWatch.

Operasi	Wilayah Replika	Perilaku sebelum upgrade	Perilaku selama peningkatan	Perilaku setelah peningkatan
	Tujuan	Penghapusan dua fase terjadi: <ul style="list-style-type: none"> • Di Fase 1, UpdateItem atur bendera penghapusan. • Di Fase 2, DeleteItem menghapus item. 	Menghapus item menggunakan DeleteItem.	Menghapus item menggunakan DeleteItem.
		Dua catatan Streams dihasilkan. Catatan pertama berisi perubahan ke <code>aws:rep:d</code> <code>eleting</code> bidang. Catatan kedua berisi atribut yang ditulis pelanggan dan atribut. <code>aws:rep:*</code>	Rekaman Stream tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan.	Rekaman Stream tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan.
		Dua RWCu dikonsumsi untuk setiap penghapusan pelanggan.	Satu RWCu dikonsumsi untuk setiap penghapusan pelanggan.	Satu RWCu dikonsumsi untuk setiap penghapusan pelanggan.

Operasi	Wilayah Replika	Perilaku sebelum upgrade	Perilaku selama peningkatan	Perilaku setelah peningkatan
		ReplicationLatency dan PendingReplication Count metrik dipublikasikan di CloudWatch.	ReplicationLatency metrik diterbitkan di CloudWatch.	ReplicationLatency metrik diterbitkan di CloudWatch.

Memutakhirkan ke Versi 2019.11.21 (Saat ini)

Lakukan langkah-langkah berikut untuk memutakhirkan versi tabel global DynamoDB Anda menggunakan AWS Management Console

Untuk meningkatkan tabel global ke Versi 2019.11.21 (Saat ini)

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/home>.
2. Di panel navigasi di sisi kiri konsol, pilih Tabel, lalu pilih tabel global yang ingin Anda tingkatkan ke Versi 2019.11.21 (Saat Ini).
3. Pilih tab Tabel Global.
4. Pilih Perbarui versi.

The screenshot shows the AWS Management Console interface for DynamoDB Global Tables. The navigation bar at the top includes 'Overview', 'Indexes', 'Monitor', 'Global tables' (highlighted with a red circle), 'Backups', and 'Exports and streams'. Below the navigation bar, the 'Replicas (1)' section is displayed, indicating the current version is 2017.11.29. There are four buttons: a refresh icon, 'Remove replication Region', 'Create replica', and 'Update version' (circled in red). A table below shows the replication details for the 'Europe (Frankfurt)' region, which is 'Active' with a status of 'Active' and a capacity of '1-10, auto scaling at 70%' for both read and write.

5. Baca dan setuju persyaratan baru, lalu pilih Perbarui versi.
6. Setelah proses pemutakhiran selesai, versi tabel global yang muncul di konsol berubah menjadi 2019.11.21.

Bekerja dengan operasi baca dan tulis

Anda dapat melakukan operasi baca dan tulis dengan DynamoDB API atau PartiQL untuk DynamoDB. Operasi ini akan memungkinkan Anda berinteraksi dengan item di tabel Anda untuk melakukan fungsionalitas dasar buat, baca, perbarui, dan hapus (CRUD).

Bagian berikut membahas lebih mendalam tentang topik ini.

Topik

- [DynamoDB API](#)
- [PartiQL - Bahasa kueri yang kompatibel dengan SQL untuk Amazon DynamoDB](#)

DynamoDB API

Topik

- [Bekerja dengan item dan atribut](#)
- [Koleksi item - cara memodelkan one-to-many hubungan di DynamoDB](#)
- [Bekerja dengan pemindaian di DynamoDB](#)

Bekerja dengan item dan atribut

Di Amazon DynamoDB, item adalah koleksi atribut. Setiap atribut mempunyai nama dan nilai. Nilai atribut dapat berupa skalar, himpunan, atau tipe dokumen. Untuk informasi selengkapnya, lihat [Amazon DynamoDB: Cara kerjanya](#).

DynamoDB menyediakan empat operasi untuk fungsionalitas dasar buat, baca, perbarui, dan hapus (CRUD). Semua operasi ini bersifat atom.

- `PutItem` — Membuat item.
- `GetItem` — Membaca item
- `UpdateItem` — Memperbarui item
- `DeleteItem` — Menghapus item

Setiap operasi ini mengharuskan Anda menentukan kunci primer item yang ingin Anda gunakan. Misalnya, untuk membaca item menggunakan `GetItem`, Anda harus menentukan kunci partisi dan kunci urutan (jika ada) untuk item tersebut.

Selain empat operasi CRUD dasar, DynamoDB juga menyediakan hal berikut ini:

- `BatchGetItem` — Membaca hingga 100 item dari satu tabel atau lebih.
- `BatchWriteItem` — Membuat atau menghapus hingga 25 item dalam satu atau beberapa tabel.

Operasi batch ini menggabungkan beberapa operasi CRUD ke dalam satu permintaan. Selain itu, operasi batch membaca dan menulis item secara paralel untuk meminimalkan latensi respons.

Bagian ini menjelaskan cara menggunakan operasi ini dan mencakup topik terkait, seperti pembaruan bersyarat dan penghitung atom. Bagian ini juga mencakup contoh kode yang menggunakan AWS SDK.

Topik

- [Membaca item](#)
- [Menulis item](#)
- [Nilai yang ditampilkan](#)
- [Operasi batch](#)
- [Penghitung atom](#)
- [Penulisan bersyarat](#)
- [Menggunakan ekspresi di DynamoDB](#)
- [Waktu untuk Hidup \(TTL\)](#)
- [Bekerja dengan item: Java](#)
- [Bekerja dengan item: .NET](#)

Membaca item

Untuk membaca item dari tabel DynamoDB, gunakan operasi `GetItem`. Anda harus memberikan nama tabel, beserta kunci primer item yang Anda inginkan.

Example

AWS CLI Contoh berikut menunjukkan cara membaca item dari `ProductCatalog` tabel.

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"1"}}'
```

Note

Dengan `GetItem`, Anda harus menentukan seluruh kunci primer, bukan hanya sebagian darinya. Misalnya, jika tabel memiliki kunci primer gabungan (kunci partisi dan kunci urutan), Anda harus menyediakan nilai untuk kunci partisi dan nilai untuk kunci urutan.

Pada akhirnya, permintaan `GetItem` melakukan pembacaan konsisten secara default. Sebagai gantinya, Anda dapat menggunakan parameter `ConsistentRead` untuk meminta bacaan sangat konsisten. (Ini mengkonsumsi unit kapasitas baca tambahan, tetapi mengembalikan sebagian besar up-to-date versi item.)

`GetItem` mengembalikan semua atribut item. Anda dapat menggunakan ekspresi proyeksi untuk mengembalikan sebagian atribut saja. Untuk informasi selengkapnya, lihat [Ekspresi proyeksi](#).

Untuk mengembalikan jumlah unit kapasitas baca yang dikonsumsi oleh `GetItem`, tetapkan parameter `ReturnConsumedCapacity` menjadi `TOTAL`.

Example

Berikut AWS Command Line Interface (AWS CLI) contoh menunjukkan beberapa `GetItem` parameter opsional.

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"1"}}' \  
  --consistent-read \  
  --projection-expression "Description, Price, RelatedItems" \  
  --return-consumed-capacity TOTAL
```

Menulis item

Untuk membuat, memperbarui, atau menghapus item dalam tabel DynamoDB, gunakan salah satu operasi berikut:

- `PutItem`

- UpdateItem
- DeleteItem

Untuk setiap operasi ini, Anda harus menentukan seluruh kunci primer, bukan hanya sebagian saja. Misalnya, jika tabel memiliki kunci primer gabungan (kunci partisi dan kunci urutan), Anda harus memberikan nilai untuk kunci partisi dan nilai untuk kunci urutan.

Untuk mengembalikan jumlah unit kapasitas tulis yang digunakan oleh salah satu operasi ini, atur parameter `ReturnConsumedCapacity` ke salah satu dari berikut ini:

- TOTAL — Mengembalikan jumlah total unit kapasitas tulis yang dikonsumsi.
- INDEXES — Mengembalikan jumlah total unit kapasitas tulis yang digunakan, dengan subtotal untuk tabel dan indeks sekunder apa pun yang terpengaruh oleh operasi.
- NONE — Tidak ada detail kapasitas tulis dikembalikan. (Ini adalah pengaturan default.)

PutItem

`PutItem` membuat item baru. Jika item dengan kunci yang sama sudah ada di tabel, item tersebut akan diganti dengan item baru.

Example

Tulis item baru ke tabel `Thread`. Kunci primer untuk `Thread` terdiri dari `ForumName` (kunci partisi) dan `Subject` (kunci urutan).

```
aws dynamodb put-item \  
  --table-name Thread \  
  --item file://item.json
```

Argumen untuk `--item` disimpan dalam file `item.json`.

```
{  
  "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

Jika item dengan kunci tertentu tidak ada, UpdateItem akan membuat item baru. Jika tidak, ini akan mengubah atribut item yang sudah ada.

Anda menggunakan ekspresi pembaruan untuk menentukan atribut yang ingin Anda ubah dan nilai barunya. Untuk informasi selengkapnya, lihat [Ekspresi pembaruan](#).

Dalam ekspresi pembaruan, Anda menggunakan nilai atribut ekspresi sebagai pengganti nilai aktual. Untuk informasi selengkapnya, lihat [Nilai atribut ekspresi](#).

Example

Modifikasi berbagai atribut dalam item Thread. Parameter ReturnValues opsional menunjukkan item seperti yang muncul setelah pembaruan. Untuk informasi selengkapnya, lihat [Nilai yang ditampilkan](#).

```
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
```

Argumen untuk --key disimpan dalam file key.json.

```
{  
  "ForumName": {"S": "Amazon DynamoDB"},  
  "Subject": {"S": "New discussion thread"}  
}
```

Argumen untuk --expression-attribute-values disimpan dalam file expression-attribute-values.json.

```
{  
  ":zero": {"N": "0"},  
  ":lastpostedby": {"S": "barney@example.com"}  
}
```

DeleteItem

DeleteItem menghapus item dengan kunci tertentu.

Example

AWS CLI Contoh berikut menunjukkan cara menghapus Thread item.

```
aws dynamodb delete-item \  
  --table-name Thread \  
  --key file:///key.json
```

Nilai yang ditampilkan

Dalam beberapa kasus, Anda mungkin ingin DynamoDB mengembalikan nilai-nilai atribut tertentu seperti nilai yang muncul sebelum atau setelah Anda memodifikasi mereka. Operasi PutItem, UpdateItem, dan DeleteItem memiliki parameter ReturnValues yang dapat Anda gunakan untuk mengembalikan nilai atribut sebelum atau setelah dimodifikasi.

Nilai default untuk ReturnValues adalah NONE, yang berarti DynamoDB tidak mengembalikan informasi apa pun tentang atribut yang dimodifikasi.

Berikut ini adalah pengaturan valid lainnya untuk ReturnValues, yang diatur oleh operasi DynamoDB API.

PutItem

- ReturnValues: ALL_OLD
 - Jika Anda menimpa item yang ada, ALL_OLD akan mengembalikan seluruh item seperti item yang muncul sebelum penempatan.
 - Jika Anda menulis item yang tidak ada, ALL_OLD tidak akan berpengaruh.

UpdateItem

Penggunaan yang paling umum untuk UpdateItem adalah untuk memperbarui item yang ada. Namun, UpdateItem benar-benar melakukan upsert, yang berarti bahwa ini secara otomatis membuat item jika sudah tidak ada.

- ReturnValues: ALL_OLD

- Jika Anda memperbarui item yang ada, `ALL_OLD` akan mengembalikan seluruh item seperti item yang muncul sebelum pembaruan.
- Jika Anda memperbarui item yang tidak ada (upsert), `ALL_OLD` tidak akan berpengaruh.
- `ReturnValues: ALL_NEW`
 - Jika Anda memperbarui item yang ada, `ALL_NEW` menampilkan seluruh item seperti item yang muncul setelah pembaruan.
 - Jika Anda memperbarui item tidak ada (upsert), `ALL_NEW` akan mengembalikan seluruh item.
- `ReturnValues: UPDATED_OLD`
 - Jika Anda memperbarui item yang ada, `UPDATED_OLD` hanya mengembalikan atribut yang diperbarui seperti atribut yang muncul sebelum pembaruan.
 - Jika Anda memperbarui item yang tidak ada (upsert), `UPDATED_OLD` tidak akan berpengaruh.
- `ReturnValues: UPDATED_NEW`
 - Jika Anda memperbarui item yang ada, `UPDATED_NEW` hanya menampilkan atribut yang terpengaruh, seperti yang muncul setelah pembaruan.
 - Jika Anda memperbarui item tidak ada (upsert), `UPDATED_NEW` hanya mengembalikan atribut diperbarui, seperti yang muncul setelah pembaruan.

DeleteItem

- `ReturnValues: ALL_OLD`
 - Jika Anda menghapus item yang ada, `ALL_OLD` akan mengembalikan seluruh item seperti yang muncul sebelum Anda menghapusnya.
 - Jika Anda menghapus item yang tidak ada, `ALL_OLD` tidak akan menampilkan data apa pun.

Operasi batch

Untuk aplikasi yang memerlukan membaca atau menulis beberapa item, DynamoDB menyediakan operasi `BatchGetItem` dan `BatchWriteItem`. Dengan menggunakan operasi ini, Anda dapat mengurangi jumlah perjalanan bolak-balik jaringan dari aplikasi Anda ke DynamoDB. Selain itu, DynamoDB melakukan operasi baca atau tulis individu secara paralel. Aplikasi Anda mendapatkan keuntungan dari paralelisme ini tanpa perlu mengelola konkurensi atau threading.

Operasi batch pada dasarnya merupakan pembungkus di sekitar beberapa permintaan baca atau tulis. Misalnya, jika permintaan `BatchGetItem` berisi lima item, DynamoDB melakukan lima operasi

GetItem atas nama Anda. Demikian pula, jika permintaan BatchWriteItem berisi dua permintaan put dan empat permintaan penghapusan, DynamoDB akan melakukan dua permintaan PutItem dan empat DeleteItem.

Secara umum, operasi batch tidak gagal kecuali semua permintaan dalam batch mengalami kegagalan. Misalnya, bayangkan Anda melakukan operasi BatchGetItem, tetapi salah satu permintaan GetItem individu dalam batch mengalami kegagalan. Dalam kasus ini, BatchGetItem mengembalikan kunci dan data dari permintaan GetItem yang gagal. Permintaan GetItem lainnya dalam batch tidak akan terpengaruh.

BatchGetItem

Satu operasi BatchGetItem dapat berisi hingga 100 permintaan GetItem individu dapat mengambil data hingga 16 MB. Selain itu, operasi BatchGetItem dapat mengambil item dari beberapa tabel.

Example

Ambil dua item dari tabel Thread, menggunakan ekspresi proyeksi untuk mengembalikan beberapa atribut saja.

```
aws dynamodb batch-get-item \  
  --request-items file://request-items.json
```

Argumen untuk `--request-items` disimpan dalam file `request-items.json`.

```
{  
  "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

Operasi `BatchWriteItem` dapat berisi hingga 25 permintaan `PutItem` dan `DeleteItem` individu, serta dapat menulis data hingga 16 MB. (Ukuran maksimum item individu adalah sebesar 400 KB.) Selain itu, operasi `BatchWriteItem` dapat menempatkan atau menghapus item dalam beberapa tabel.

Note

`BatchWriteItem` tidak mendukung permintaan `UpdateItem`.

Example

Tulis dua item ke tabel `ProductCatalog`.

```
aws dynamodb batch-write-item \  
  --request-items file://request-items.json
```

Argumen untuk `--request-items` disimpan dalam file `request-items.json`.

```
{  
  "ProductCatalog": [  
    {  
      "PutRequest": {  
        "Item": {  
          "Id": { "N": "601" },  
          "Description": { "S": "Snowboard" },  
          "QuantityOnHand": { "N": "5" },  
          "Price": { "N": "100" }  
        }  
      }  
    },  
    {  
      "PutRequest": {  
        "Item": {  
          "Id": { "N": "602" },  
          "Description": { "S": "Snow shovel" }  
        }  
      }  
    }  
  ]  
}
```



```
}
```

Penghitung atom

Anda dapat menggunakan operasi `UpdateItem` untuk mengimplementasikan penghitung atom—atribut numerik yang bertambah, tanpa syarat, tanpa mengganggu permintaan tulis lainnya. (Semua permintaan tulis diterapkan sesuai urutan penerimaannya.) Dengan penghitung atom, pembaruan tidak idempotent. Dengan kata lain, nilai numerik bertambah atau berkurang setiap kali Anda menelepon. `UpdateItem` Jika nilai kenaikan yang digunakan untuk memperbarui penghitung atom positif, maka itu dapat menyebabkan penghitungan berlebihan. Jika nilai kenaikan negatif, maka dapat menyebabkan kekurangan perhitungan.

Anda dapat menggunakan penghitung atom untuk melacak jumlah pengunjung ke situs web. Dalam hal ini, aplikasi Anda akan menaikkan nilai numerik, terlepas dari nilai saat ini. Jika operasi `UpdateItem` gagal, aplikasi hanya dapat mencoba ulang operasi. Tindakan ini akan berisiko memperbarui penghitung dua kali, tetapi Anda mungkin bisa menoleransi sedikit kelebihan penghitungan atau kekurangan penghitungan dari pengunjung situs web.

Penghitung atom tidak akan cocok ketika kelebihan penghitungan atau kekurangan penghitungan tidak dapat ditoleransi (misalnya, dalam aplikasi perbankan). Dalam kasus ini, lebih aman menggunakan pembaruan bersyarat, bukannya penghitung atom.

Untuk informasi selengkapnya, lihat [Menambahkan dan mengurangi atribut numerik](#).

Example

AWS CLI Contoh berikut menambah produk `Price` sebesar 5. Untuk contoh ini, item diketahui ada sebelum penghitung diperbarui. Karena `UpdateItem` tidak idempotent, `Price` meningkat setiap kali Anda menjalankan kode ini.

```
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
```

Penulisan bersyarat

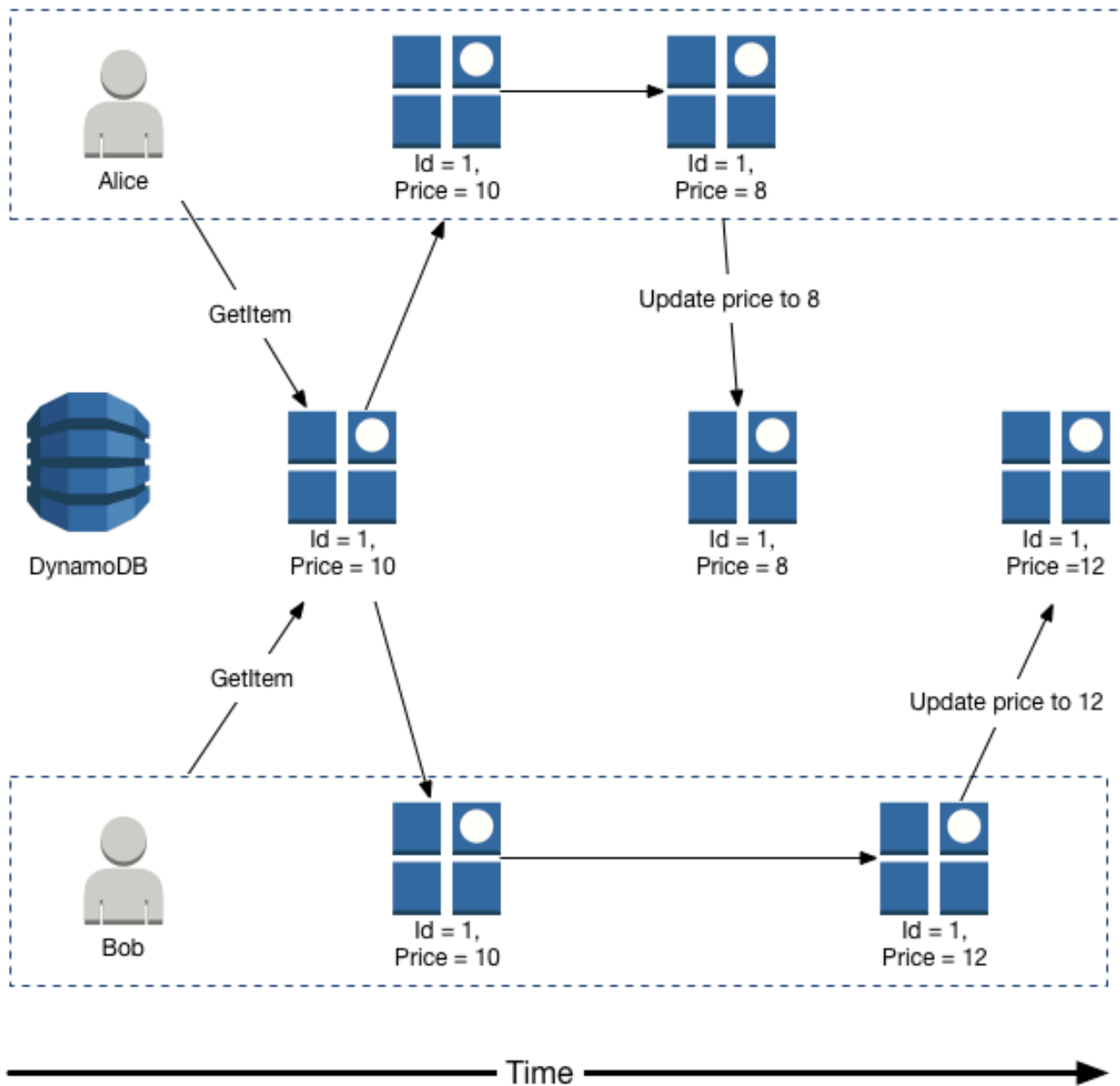
Secara default, operasi tulis DynamoDB (`PutItem`, `UpdateItem`, `DeleteItem`) bersifat tanpa syarat: Setiap operasi menimpa item yang ada yang memiliki kunci primer tertentu.

DynamoDB secara opsional mendukung penulisan bersyarat untuk operasi ini. Penulisan bersyarat akan berhasil hanya jika atribut item memenuhi satu atau beberapa kondisi yang diharapkan. Jika tidak, operasi ini akan mengembalikan kesalahan.

Penulisan bersyarat memeriksa kondisinya terhadap versi item yang terbaru diperbarui. Perhatikan bahwa jika item sebelumnya tidak ada atau jika operasi sukses terbaru terhadap item tersebut adalah penghapusan, maka penulisan bersyarat tidak akan menemukan item sebelumnya.

Penulisan bersyarat sangat membantu dalam banyak situasi. Misalnya, Anda mungkin ingin operasi `PutItem` bisa berhasil hanya jika sudah tidak ada item dengan kunci primer yang sama. Atau Anda dapat mencegah operasi `UpdateItem` memodifikasi item jika salah satu atributnya memiliki nilai tertentu.

Penulisan bersyarat berguna jika beberapa pengguna mencoba mengubah item yang sama. Perhatikan diagram berikut, di mana dua pengguna (Alice dan Bob) bekerja dengan item yang sama dari tabel DynamoDB.



Misalkan Alice menggunakan AWS CLI untuk memperbarui `Price` atribut ke 8.

```
aws dynamodb update-item \
  --table-name ProductCatalog \
  --key '{"Id":{"N":"1"}}' \
  --update-expression "SET Price = :newval" \
```

```
--expression-attribute-values file://expression-attribute-values.json
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `expression-attribute-values.json`:

```
{
  ":newval":{"N":"8"}
}
```

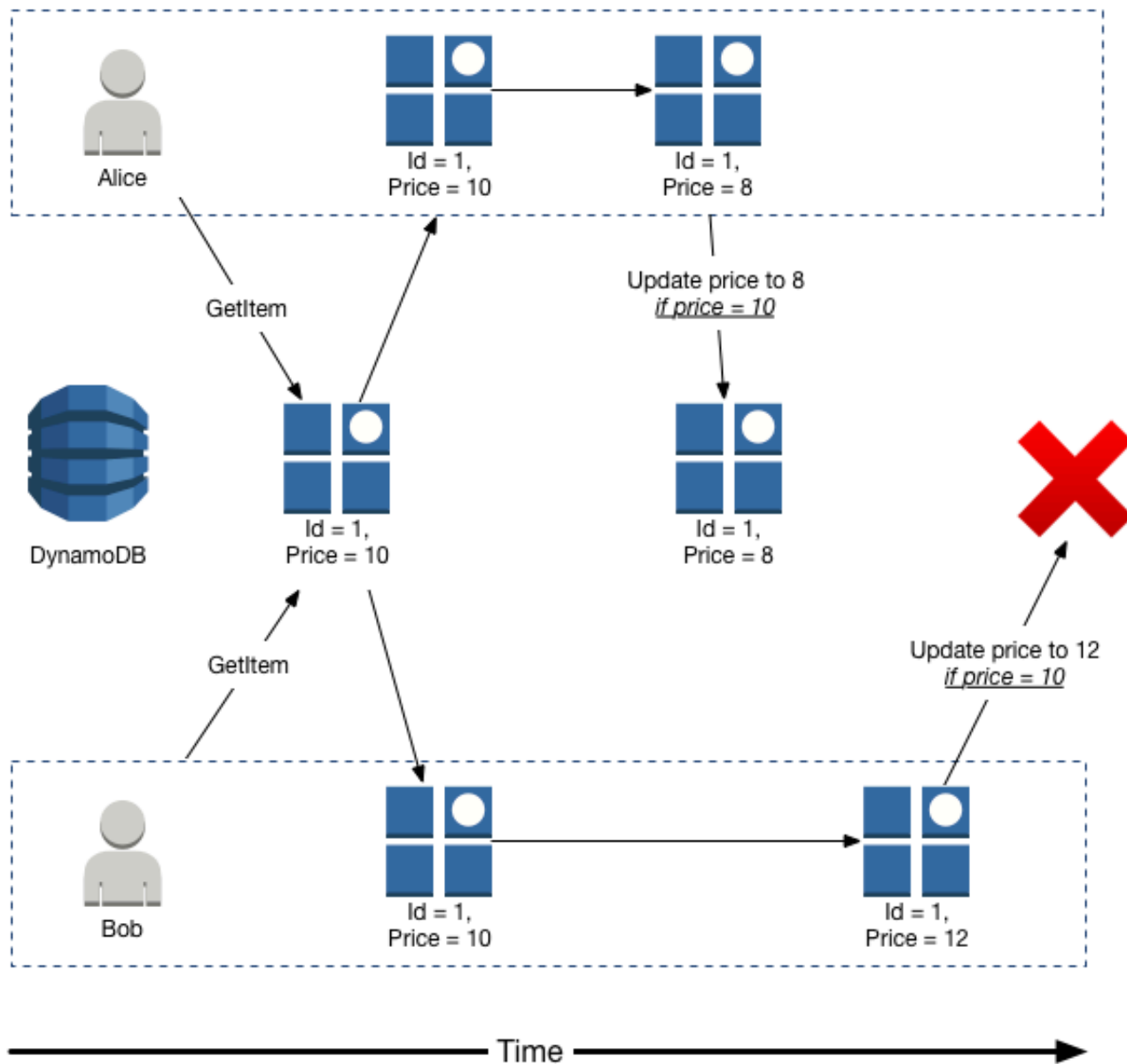
Sekarang anggaplah Bob mengeluarkan permintaan `UpdateItem` yang serupa kemudian, tetapi mengubah `Price` menjadi 12. Untuk Bob, parameter `--expression-attribute-values` akan terlihat seperti berikut ini.

```
{
  ":newval":{"N":"12"}
}
```

Permintaan Bob berhasil, tetapi pembaruan Alice sebelumnya hilang.

Untuk meminta `PutItem`, `DeleteItem`, atau `UpdateItem` bersyarat, Anda harus menentukan ekspresi kondisi. Ekspresi kondisi adalah string yang berisi nama atribut, operator bersyarat, dan fungsi bawaan. Seluruh ekspresi harus bernilai benar. Jika tidak, operasi gagal.

Sekarang pertimbangkan diagram berikut yang menunjukkan bagaimana penulisan bersyarat akan mencegah pembaruan Alice ditimpa.



Alice mencoba pertama kali untuk memperbarui Price menjadi 8, tetapi hanya jika Price saat ini adalah 10.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `expression-attribute-values.json`.

```
{  
  ":newval":{"N":"8"},  
  ":currval":{"N":"10"}  
}
```

Pembaruan Alice berhasil karena kondisi bernilai benar.

Berikutnya, Bob mencoba memperbarui `Price` menjadi 12, tetapi hanya jika `Price` saat ini adalah 10. Untuk Bob, parameter `--expression-attribute-values` akan terlihat seperti berikut ini.

```
{  
  ":newval":{"N":"12"},  
  ":currval":{"N":"10"}  
}
```

Karena Alice sebelumnya telah mengubah `Price` ke 8, ekspresi kondisi bernilai SALAH, dan pembaruan Bob mengalami kegagalan.

Untuk informasi selengkapnya, lihat [Ekspresi kondisi](#).

Idempotensi penulisan bersyarat

Penulisan bersyarat dapat menjadi idempotent jika pemeriksaan bersyarat berada pada atribut yang sama dan sedang diperbarui. Ini berarti bahwa DynamoDB melakukan permintaan tulis yang diberikan hanya jika nilai atribut tertentu dalam item cocok dengan tempat mereka berada pada saat permintaan sesuai harapan Anda.

Misalnya, misalnya Anda mengeluarkan permintaan `UpdateItem` untuk meningkatkan `Price` dari item sebesar 3, tetapi hanya jika `Price` saat ini adalah 20. Setelah Anda mengirimkan permintaan, tetapi sebelum Anda mendapatkan hasil kembali, terjadi kesalahan jaringan, dan Anda tidak tahu apakah permintaan tersebut berhasil. Karena penulisan bersyarat ini idempotent, Anda dapat mencoba kembali permintaan `UpdateItem` yang sama, dan DynamoDB memperbarui item hanya jika `Price` saat ini adalah 20.

Unit kapasitas yang digunakan oleh penulisan bersyarat

Jika `ConditionExpression` bernilai `false` selama penulisan bersyarat, DynamoDB masih menggunakan kapasitas tulis dari tabel. Jumlah yang dikonsumsi tergantung dari besar kecilnya barang yang ada (atau minimal 1). Misalnya, jika item yang ada berukuran 300kb dan item baru yang Anda coba buat atau perbarui berukuran 310kb, unit kapasitas tulis yang digunakan akan menjadi 300 jika kondisi gagal, dan 310 jika kondisi berhasil. Jika ini adalah item baru (belum ada item yang ada), maka unit kapasitas tulis yang digunakan adalah 1 jika kondisi gagal dan 310 jika kondisi berhasil.

Note

Operasi tulis hanya mengonsumsi unit kapasitas tulis. Operasi tersebut tidak pernah mengonsumsi unit kapasitas baca.

Penulisan bersyarat yang gagal akan menampilkan `ConditionalCheckFailedException`. Ketika hal ini terjadi, Anda tidak menerima informasi apa pun sebagai respons tentang kapasitas tulis yang dikonsumsi. Namun, Anda dapat melihat `ConsumedWriteCapacityUnits` metrik untuk tabel di Amazon CloudWatch. Untuk informasi selengkapnya, lihat [Metrik DynamoDB](#) di [Pencatatan dan pemantauan di DynamoDB](#).

Untuk mengembalikan jumlah unit kapasitas tulis yang dikonsumsi selama penulisan bersyarat, Anda menggunakan parameter `ReturnConsumedCapacity`:

- **TOTAL** — Mengembalikan jumlah total unit kapasitas tulis yang dikonsumsi.
- **INDEXES** — Mengembalikan jumlah total unit kapasitas tulis yang digunakan, dengan subtotal untuk tabel dan indeks sekunder apa pun yang terpengaruh oleh operasi.
- **NONE** — Tidak ada detail kapasitas tulis dikembalikan. (Ini menjadi opsi default.)

Note

Tidak seperti indeks sekunder global, indeks sekunder lokal berbagi kapasitas throughput yang disediakan dengan tabelnya. Aktivitas baca dan tulis pada indeks sekunder lokal mengonsumsi kapasitas throughput yang disediakan dari tabel.

Menggunakan ekspresi di DynamoDB

Di Amazon DynamoDB, Anda menggunakan ekspresi untuk menunjukkan atribut yang ingin Anda baca dari suatu item. Anda juga menggunakan ekspresi ketika menulis item untuk menunjukkan kondisi yang harus dipenuhi (juga dikenal sebagai pembaruan bersyarat), dan untuk menunjukkan bagaimana atribut yang akan diperbarui. Bagian ini menjelaskan tata bahasa ekspresi dasar dan jenis ekspresi yang tersedia.

Note

Untuk kompatibilitas ke belakang, DynamoDB juga mendukung parameter bersyarat yang tidak menggunakan ekspresi. Untuk informasi selengkapnya, lihat [Parameter bersyarat lama](#). Aplikasi baru harus menggunakan ekspresi, bukan parameter warisan.

Topik

- [Menentukan atribut item saat menggunakan ekspresi](#)
- [Ekspresi proyeksi](#)
- [Nama atribut ekspresi di DynamoDB](#)
- [Nilai atribut ekspresi](#)
- [Ekspresi kondisi](#)
- [Ekspresi pembaruan](#)

Menentukan atribut item saat menggunakan ekspresi

Bagian ini menjelaskan cara merujuk ke atribut item dalam suatu ekspresi di Amazon DynamoDB. Anda dapat bekerja dengan atribut apa pun, meskipun itu sangat besar dalam beberapa daftar dan peta.

Topik

- [Atribut tingkat atas](#)
- [Atribut bersarang](#)
- [Jalur dokumen](#)

Item Sampel: ProductCatalog

Berikut ini adalah representasi item dalam tabel ProductCatalog. (Tabel ini dijelaskan dalam [Contoh tabel dan data.](#))

```
{
  "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"
}
```

Perhatikan hal-hal berikut:

- Nilai kunci partisi (Id) adalah 123. Tidak ada kunci urutan.
- Sebagian besar atribut memiliki tipe daya skalar, seperti String, Number, Boolean, dan Null.
- Satu atribut (Color) adalah String Set.

- Atribut berikut adalah tipe daya dokumen:
 - Daftar dari `RelatedItems`. Setiap elemen adalah `Id` untuk produk terkait.
 - Peta dari `Pictures`. Setiap elemen adalah deskripsi singkat dari gambar, bersama dengan URL untuk file gambar yang sesuai.
 - Peta dari `ProductReviews`. Setiap elemen mewakili penilaian dan daftar ulasan yang sesuai dengan penilaian tersebut. Awalnya, peta ini diisi dengan ulasan bintang lima dan bintang satu.

Atribut tingkat atas

Suatu atribut dikatakan tingkat atas jika tidak melekat pada atribut lain. Untuk item `ProductCatalog`, atribut tingkat atas adalah sebagai berikut:

- `Id`
- `Title`
- `Description`
- `BicycleType`
- `Brand`
- `Price`
- `Color`
- `ProductCategory`
- `InStock`
- `QuantityOnHand`
- `RelatedItems`
- `Pictures`
- `ProductReviews`
- `Comment`
- `Safety.Warning`

Semua atribut tingkat atas ini bersifat skalar, kecuali untuk `Color` (daftar), `RelatedItems` (daftar), `Pictures` (peta), dan `ProductReviews` (peta).

Atribut bersarang

Suatu atribut dikatakan bersarang jika melekat pada atribut lain. Untuk mengakses atribut bersarang, Anda menggunakan operator dereferensi:

- `[n]` — untuk elemen daftar
- `.` (titik) — untuk elemen peta

Mengakses elemen daftar

Operator dereferensi untuk elemen daftar adalah `[N]`, dengan `n` adalah nomor elemen. Elemen daftar berbasis nol, sehingga `[0]` mewakili elemen pertama dalam daftar, `[1]` mewakili elemen kedua, dan seterusnya. Berikut ini adalah beberapa contohnya:

- `MyList[0]`
- `AnotherList[12]`
- `ThisList[5][11]`

Elemen `ThisList[5]` itu sendiri adalah daftar bersarang. Karena itu, `ThisList[5][11]` merujuk pada elemen ke-12 dalam daftar itu.

Angka di dalam tanda kurung siku harus berupa bilangan bulat bukan negatif. Oleh karena itu, ekspresi berikut ini tidak valid:

- `MyList[-1]`
- `MyList[0.4]`

Mengakses elemen peta

Operator dereferensi untuk elemen peta adalah `.` (titik). Gunakan satu titik sebagai pemisah antara unsur-unsur dalam peta:

- `MyMap.nestedField`
- `MyMap.nestedField.deeplyNestedField`

Jalur dokumen

Dalam sebuah ekspresi, Anda menggunakan jalur dokumen untuk memberi tahu DynamoDB lokasi penemuan atribut. Untuk atribut tingkat atas, jalur dokumen hanyalah nama atribut. Untuk atribut bersarang, Anda mengonstruksi jalur dokumen menggunakan operator dereferensi.

Berikut adalah beberapa contoh dari jalur dokumen. (Lihat item yang ditampilkan dalam [Menentukan atribut item saat menggunakan ekspresi.](#))

- Atribut skalar tingkat atas.

`Description`

- Atribut daftar tingkat atas. (Ini mengembalikan seluruh daftar, bukan hanya beberapa elemen.)

`RelatedItems`

- Elemen ketiga dari daftar `RelatedItems`. (Ingat bahwa elemen daftar berbasis nol.)

`RelatedItems[2]`

- Gambar tampilan depan produk.

`Pictures.FrontView`

- Semua ulasan bintang lima.

`ProductReviews.FiveStar`

- Yang pertama dari ulasan bintang lima.

`ProductReviews.FiveStar[0]`

Note

Kedalaman maksimum untuk jalur dokumen adalah 32. Oleh karena itu, jumlah operator dereferensi dalam suatu jalur tidak dapat melebihi batas ini.

Anda dapat menggunakan nama atribut apa pun di jalur dokumen asalkan memenuhi persyaratan berikut:

- Nama atribut harus dimulai dengan tanda pound (#)
- Karakter pertama adalah a-z atau A-Z dan atau 0-9

- Karakter kedua (jika ada) adalah a-z, A-Z

Note

Jika nama atribut tidak memenuhi persyaratan ini, Anda harus menentukan nama atribut ekspresi sebagai placeholder.

Untuk informasi selengkapnya, lihat [Nama atribut ekspresi di DynamoDB](#).

Ekspresi proyeksi

Untuk membaca data dari tabel, Anda menggunakan operasi seperti `GetItem`, `Query`, atau `Scan`. Amazon DynamoDB mengembalikan semua atribut item secara default. Untuk mendapatkan beberapa saja, bukan semua atribut, gunakan ekspresi proyeksi.

Ekspresi proyeksi adalah string yang mengidentifikasi atribut yang Anda inginkan. Untuk mengambil atribut tunggal, tentukan namanya. Untuk beberapa atribut, nama harus dipisahkan koma.

Berikut adalah beberapa contoh ekspresi proyeksi, berdasarkan item `ProductCatalog` dari [Menentukan atribut item saat menggunakan ekspresi](#):

- Atribut tingkat atas tunggal.

`Title`

- Tiga atribut tingkat atas. DynamoDB mengambil seluruh set `Color`.

`Title, Price, Color`

- Empat atribut tingkat atas. DynamoDB mengembalikan seluruh konten `RelatedItems` dan `ProductReviews`.

`Title, Description, RelatedItems, ProductReviews`

DynamoDB memiliki daftar kata-kata khusus dan karakter khusus. Anda dapat menggunakan nama atribut dalam ekspresi proyeksi, asalkan karakter pertama adalah a-z atau A-Z dan karakter kedua (jika ada) adalah a-z, A-Z, atau 0-9. Jika nama atribut tidak memenuhi persyaratan ini, Anda harus menentukan nama atribut ekspresi sebagai placeholder. Untuk daftar lengkap, lihat [Disimpan kata-kata di DynamoDB](#). Selain itu, karakter berikut memiliki arti khusus dalam DynamoDB: `#` (hash) dan `:` (titik dua).

Meskipun DynamoDB mengizinkan Anda menggunakan kata-kata khusus dan karakter khusus ini untuk nama, kami menyarankan agar Anda menghindari melakukannya karena Anda harus menentukan variabel placeholder setiap kali Anda menggunakan nama-nama ini dalam sebuah ekspresi. Untuk informasi selengkapnya, lihat [Nama atribut ekspresi di DynamoDB](#).

AWS CLI Contoh berikut menunjukkan bagaimana menggunakan ekspresi proyeksi dengan GetItem operasi. Ekspresi proyeksi ini mengambil atribut skalar tingkat atas (Description), elemen pertama dalam daftar (RelatedItems[0]), dan daftar bersarang dalam peta (ProductReviews.FiveStar).

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key file://key.json \  
  --projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"
```

JSON berikut akan dikembalikan untuk contoh ini.

```
{  
  "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"  
        }  
      ]  
    }  
  }  
}
```

```
}  
  }  
}
```

Argumen untuk `--key` disimpan dalam file `key.json`.

```
{  
  "Id": { "N": "123" }  
}
```

Untuk contoh kode khusus bahasa pemrograman, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Nama atribut ekspresi di DynamoDB

Nama atribut ekspresi adalah placeholder yang Anda gunakan dalam ekspresi Amazon DynamoDB sebagai alternatif untuk nama atribut aktual. Nama atribut ekspresi harus dimulai dengan tanda pagar (`#`), dan diikuti oleh satu atau lebih karakter alfanumerik dan karakter garis bawah (`_`).

Bagian ini menjelaskan beberapa situasi ketika Anda harus menggunakan nama atribut ekspresi.

Note

Contoh di bagian ini menggunakan AWS Command Line Interface (AWS CLI). Untuk contoh kode khusus bahasa pemrograman, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Topik

- [Kata yang dicadangkan](#)
- [Nama atribut yang berisi karakter khusus](#)
- [Atribut bersarang](#)
- [Mengulangi nama atribut](#)

Kata yang dicadangkan

Terkadang Anda mungkin perlu menulis ekspresi yang berisi nama atribut yang bertentangan dengan kata terpesan DynamoDB. (Untuk daftar lengkap kata-kata yang dicadangkan, lihat [Disimpan kata-kata di DynamoDB](#).)

Misalnya, AWS CLI contoh berikut akan gagal karena `COMMENT` merupakan kata yang dicadangkan.

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "Comment"
```

Untuk mengatasi hal ini, Anda dapat mengganti `Comment` dengan nama atribut ekspresi seperti `#c`. `#` (tanda pagar) diperlukan dan menunjukkan bahwa ini adalah placeholder untuk nama atribut. AWS CLI Contohnya sekarang akan terlihat seperti berikut.

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "#c" \  
  --expression-attribute-names '{"#c":"Comment"}'
```

Note

Jika nama atribut dimulai dengan nomor, berisi spasi, atau berisi kata terpesan, Anda harus menggunakan nama atribut ekspresi untuk menggantikan nama atribut dalam ekspresi tersebut.

Nama atribut yang berisi karakter khusus

Dalam suatu ekspresi, titik (".") ditafsirkan sebagai karakter pemisah pada jalur dokumen. Namun, DynamoDB juga memungkinkan Anda menggunakan karakter titik dan karakter khusus lainnya, seperti tanda hubung ("-") sebagai bagian dari nama atribut. Dalam beberapa kasus, ini bisa menjadi ambigu. Untuk menggambarkan, misalkan bahwa Anda ingin mengambil atribut `Safety.Warning` dari item `ProductCatalog` (lihat [Menentukan atribut item saat menggunakan ekspresi](#)).

Misalkan Anda ingin mengakses `Safety.Warning` menggunakan ekspresi proyeksi.

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "Safety.Warning"
```

DynamoDB akan mengembalikan hasil kosong, bukan string yang diharapkan ("`Always wear a helmet`"). Hal ini karena DynamoDB menafsirkan titik dalam ekspresi sebagai pemisah jalur

dokumen. Dalam hal ini, Anda harus menentukan nama atribut ekspresi (seperti `#sw`) sebagai pengganti untuk `Safety.Warning`. Anda kemudian dapat menggunakan ekspresi proyeksi berikut.

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "#sw" \  
  --expression-attribute-names '{"#sw":"Safety.Warning"}'
```

DynamoDB kemudian akan mengembalikan hasil yang benar.

Note

Jika nama atribut berisi titik (".") atau tanda hubung ("-"), Anda harus menggunakan nama atribut ekspresi untuk menggantikan nama atribut dalam ekspresi tersebut.

Atribut bersarang

Misalkan Anda ingin mengakses atribut bersarang `ProductReviews.OneStar`, menggunakan ekspresi proyeksi berikut.

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "ProductReviews.OneStar"
```

Hasilnya akan berisi semua ulasan produk bintang satu, seperti yang diharapkan.

Tapi bagaimana jika Anda memutuskan menggunakan nama atribut ekspresi sebagai gantinya? Misalnya, apa yang akan terjadi jika Anda mendefinisikan `#pr1star` sebagai pengganti `ProductReviews.OneStar`?

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "#pr1star" \  
  --expression-attribute-names '{"#pr1star":"ProductReviews.OneStar"}'
```

DynamoDB akan mengembalikan hasil kosong bukan peta yang diharapkan dari ulasan bintang satu. Hal ini karena DynamoDB menafsirkan titik dalam nama atribut ekspresi sebagai karakter

dalam nama atribut. Ketika DynamoDB mengevaluasi ekspresi nama atribut `#pr1star`, itu menentukan bahwa `ProductReviews.OneStar` merujuk pada atribut skalar—yang bukan apa yang dimaksudkan.

Pendekatan yang benar akan mendefinisikan nama atribut ekspresi untuk setiap elemen dalam jalur dokumen:

- `#pr` – `ProductReviews`
- `#1star` – `OneStar`

Anda kemudian dapat menggunakan `#pr.#1star` untuk ekspresi proyeksi.

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "#pr.#1star" \  
  --expression-attribute-names '{"#pr":"ProductReviews", "#1star":"OneStar"}'
```

DynamoDB kemudian akan mengembalikan hasil yang benar.

Mengulangi nama atribut

Nama atribut ekspresi sangat membantu ketika Anda perlu merujuk ke nama atribut yang sama berulang-ulang. Misalnya, pertimbangkan ekspresi berikut untuk mengambil beberapa ulasan dari item `ProductCatalog`.

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "ProductReviews.FiveStar, ProductReviews.ThreeStar,  
ProductReviews.OneStar"
```

Untuk membuat hal ini lebih jelas, Anda dapat mengganti `ProductReviews` dengan nama atribut ekspresi seperti `#pr`. Ekspresi yang direvisi kini akan terlihat seperti berikut ini.

- `#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"
```

```
--key '{"Id":{"N":"123"}}' \  
--projection-expression "#pr.FiveStar, #pr.ThreeStar, #pr.OneStar" \  
--expression-attribute-names '{"#pr":"ProductReviews"}
```

Jika Anda menentukan nama atribut ekspresi, Anda harus menggunakannya secara konsisten di seluruh ekspresi. Selain itu, Anda tidak dapat menghilangkan simbol #.

Nilai atribut ekspresi

Jika Anda perlu membandingkan atribut dengan nilai, tentukan nilai atribut ekspresi sebagai placeholder. Nilai atribut ekspresi dalam Amazon DynamoDB adalah pengganti untuk nilai-nilai aktual yang ingin Anda bandingkan—nilai yang mungkin tidak Anda ketahui hingga runtime. Nilai atribut ekspresi harus dimulai dengan titik dua (:) dan diikuti oleh satu atau beberapa karakter alfanumerik.

Misalnya, anggaplah Anda ingin menampilkan semua item ProductCatalog yang tersedia dalam Black dan berbiaya 500 atau lebih sedikit. Anda dapat menggunakan operasi Scan dengan ekspresi filter, seperti dalam contoh AWS Command Line Interface (AWS CLI).

```
aws dynamodb scan \  
  --table-name ProductCatalog \  
  --filter-expression "contains(Color, :c) and Price <= :p" \  
  --expression-attribute-values file://values.json
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{  
  ":c": { "S": "Black" },  
  ":p": { "N": "500" }  
}
```

Note

Operasi Scan membaca setiap item dalam tabel. Jadi Anda harus menghindari penggunaan Scan dengan tabel besar.

Ekspresi filter diterapkan ke hasil Scan, dan item yang tidak cocok dengan ekspresi filter dibuang.

Jika Anda menentukan nilai atribut ekspresi, Anda harus menggunakannya secara konsisten di seluruh ekspresi. Selain itu, Anda tidak dapat menghilangkan simbol :.

Nilai atribut ekspresi digunakan dengan ekspresi kondisi kunci, ekspresi kondisi, ekspresi pembaruan, dan ekspresi filter.

Note

Untuk contoh kode khusus bahasa pemrograman, lihat [Memulai dengan DynamoDB dan SDK AWS](#).

Ekspresi kondisi

Untuk memanipulasi data dalam tabel Amazon DynamoDB, Anda menggunakan operasi `PutItem`, `UpdateItem`, dan `DeleteItem`. (Anda juga dapat menggunakan `BatchWriteItem` untuk melakukan beberapa operasi `PutItem` atau `DeleteItem` dalam satu panggilan.)

Untuk operasi manipulasi data ini, Anda dapat menentukan ekspresi kondisi untuk menentukan item mana yang harus dimodifikasi. Jika ekspresi kondisi bernilai `true`, operasi berhasil; jika tidak, operasi akan gagal.

`DeleteItem`, `OperasiPutItem`, `UpdateItem`, dan memiliki `ReturnValues` parameter yang dapat Anda gunakan untuk mengembalikan nilai atribut seperti yang muncul sebelum atau setelah Anda memodifikasinya. Untuk informasi lebih lanjut, lihat [ReturnValues](#).

Berikut ini adalah beberapa AWS Command Line Interface (AWS CLI) contoh penggunaan ekspresi kondisi. Contoh-contoh ini berdasarkan tabel `ProductCatalog`, yang diperkenalkan dalam [Menentukan atribut item saat menggunakan ekspresi](#). Kunci partisi untuk tabel ini adalah `Id`, dan tidak ada kunci urutan. Operasi `PutItem` berikut membuat item `ProductCatalog` sampel yang dirujuk oleh contoh.

```
aws dynamodb put-item \  
  --table-name ProductCatalog \  
  --item file://item.json
```

Argumen untuk `--item` disimpan dalam file `item.json`. (Untuk kemudahan, hanya beberapa atribut item yang digunakan.)

```
{  
  "Id": {"N": "456" },  
  "ProductCategory": {"S": "Sporting Goods" },  
  "Price": {"N": "650" }
```

```
}
```

Topik

- [Put bersyarat](#)
- [Penghapusan bersyarat](#)
- [Pembaruan bersyarat](#)
- [Contoh ekspresi bersyarat](#)
- [Operator perbandingan dan referensi fungsi](#)

Put bersyarat

Operasi `PutItem` menimpa item dengan kunci primer yang sama (jika ada). Jika Anda ingin menghindari hal ini, gunakan ekspresi kondisi. Hal ini memungkinkan penulisan untuk dilanjutkan hanya jika item yang dimaksud belum memiliki kunci primer yang sama.

Contoh berikut digunakan `attribute_not_exists()` untuk memeriksa apakah kunci primer ada dalam tabel sebelum mencoba operasi tulis.

Note

Jika kunci primer Anda terdiri dari kunci partisi (pk) dan kunci urutan (sk), parameter akan memeriksa apakah `attribute_not_exists(pk)` DAN `attribute_not_exists(sk)` mengevaluasi benar atau salah sebelum mencoba operasi tulis.

```
aws dynamodb put-item \  
  --table-name ProductCatalog \  
  --item file://item.json \  
  --condition-expression "attribute_not_exists(Id)"
```

Jika ekspresi kondisi bernilai false, DynamoDB mengembalikan pesan kesalahan berikut: Permintaan bersyarat gagal.

Note

Untuk informasi selengkapnya tentang fungsi `attribute_not_exists` dan lainnya, lihat [Operator perbandingan dan referensi fungsi](#).

Penghapusan bersyarat

Untuk melakukan penghapusan bersyarat, Anda menggunakan operasi `DeleteItem` dengan ekspresi kondisi. Ekspresi kondisi harus bernilai `true` agar operasi berhasil; jika tidak, operasi akan gagal.

Pertimbangkan item dari [Ekspresi kondisi](#).

```
{
  "Id": {
    "N": "456"
  },
  "Price": {
    "N": "650"
  },
  "ProductCategory": {
    "S": "Sporting Goods"
  }
}
```

Misalkan Anda ingin menghapus item, tetapi hanya dalam kondisi berikut:

- `ProductCategory` adalah "Alat Olahraga" atau "Perlengkapan Berkebun."
- `Price` adalah antara 500 dan 600.

Contoh berikut mencoba menghapus versi item.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
  ":cat1": {"S": "Sporting Goods"},
  ":cat2": {"S": "Gardening Supplies"},
  ":lo": {"N": "500"},
  ":hi": {"N": "600"}
```

```
}
```

Note

Dalam ekspresi kondisi, `:` (karakter titik dua) menunjukkan nilai atribut ekspresi—placeholder untuk nilai aktual. Untuk informasi selengkapnya, lihat [Nilai atribut ekspresi](#).

Untuk informasi selengkapnya tentang IN, AND, dan kata kunci lainnya, lihat [Operator perbandingan dan referensi fungsi](#).

Dalam contoh ini, perbandingan `ProductCategory` bernilai `true`, tetapi perbandingan `Price` bernilai `false`. Hal ini menyebabkan ekspresi kondisi untuk bernilai `false` dan operasi `DeleteItem` gagal.

Pembaruan bersyarat

Untuk melakukan pembaruan bersyarat, Anda menggunakan operasi `UpdateItem` dengan ekspresi kondisi. Ekspresi kondisi harus bernilai `true` agar operasi berhasil; jika tidak, operasi akan gagal.

Note

`UpdateItem` juga mendukung ekspresi pembaruan, di mana Anda menentukan modifikasi yang ingin Anda buat ke item. Untuk informasi selengkapnya, lihat [Ekspresi pembaruan](#).

Misalkan Anda memulai dengan item yang ditampilkan dalam [Ekspresi kondisi](#).

```
{
  "Id": { "N": "456"},
  "Price": {"N": "650"},
  "ProductCategory": {"S": "Sporting Goods"}
}
```

Contoh berikut melakukan operasi `UpdateItem`. Operasi ini mencoba untuk mengurangi `Price` dari produk sebesar 75—tetapi ekspresi kondisi mencegah pembaruan jika `Price` saat ini kurang dari atau sama dengan 500.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{  
  ":discount": { "N": "75"},  
  ":limit": { "N": "500"}  
}
```

Jika `Price` awal adalah 650, operasi `UpdateItem` mengurangi `Price` menjadi 575. Jika Anda menjalankan operasi `UpdateItem` lagi, `Price` berkurang menjadi 500. Jika Anda menjalankannya untuk kali ketiga, ekspresi kondisi bernilai `false`, dan pembaruan gagal.

Note

Dalam ekspresi kondisi, `:` (karakter titik dua) menunjukkan nilai atribut ekspresi—placeholder untuk nilai aktual. Untuk informasi selengkapnya, lihat [Nilai atribut ekspresi](#).

Untuk informasi selengkapnya tentang `>` dan operator lainnya, lihat [Operator perbandingan dan referensi fungsi](#).

Contoh ekspresi bersyarat

Untuk informasi selengkapnya tentang fungsi yang digunakan dalam contoh berikut, lihat [Operator perbandingan dan referensi fungsi](#). Jika Anda ingin tahu lebih banyak tentang cara menentukan jenis atribut yang berbeda dalam suatu ekspresi, lihat [Menentukan atribut item saat menggunakan ekspresi](#).

Memeriksa atribut dalam item

Anda dapat memeriksa keberadaan (atau ketiadaan) atribut apa pun. Jika ekspresi kondisi bernilai `true`, operasi berhasil; jika tidak, operasi akan gagal.

Contoh berikut menggunakan `attribute_not_exists` untuk menghapus produk hanya jika operasi tidak memiliki atribut `Price`.

```
aws dynamodb delete-item \  

```



```
--table-name ProductCatalog \  
--key '{"Id": {"N": "456"}}' \  
--condition-expression "attribute_not_exists(Price)"
```

DynamoDB juga menyediakan fungsi `attribute_exists`. Contoh berikut menghapus produk hanya jika operasi telah menerima ulasan buruk.

```
aws dynamodb delete-item \  
  --table-name ProductCatalog \  
  --key '{"Id": {"N": "456"}}' \  
  --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

Memeriksa jenis atribut

Anda dapat memeriksa tipe daya nilai atribut dengan menggunakan fungsi `attribute_type`. Jika ekspresi kondisi bernilai `true`, operasi berhasil; jika tidak, operasi akan gagal.

Contoh berikut menggunakan `attribute_type` untuk menghapus produk hanya jika memiliki atribut `Color` jenis `Set String`.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam `expression-attribute-values` file.json.

```
{  
  ":v_sub":{"S":"SS"}  
}
```

Memeriksa nilai awal string

Anda dapat memeriksa apakah nilai atribut `String` dimulai dengan substring tertentu dengan menggunakan fungsi `begins_with`. Jika ekspresi kondisi bernilai `true`, operasi berhasil; jika tidak, operasi akan gagal.

Contoh berikut menggunakan `begins_with` untuk menghapus produk hanya jika elemen `FrontView` dari peta `Pictures` dimulai dengan nilai tertentu.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam `expression-attribute-values` `file.json`.

```
{  
  ":v_sub":{"S":"http://"}  
}
```

Memeriksa elemen dalam set

Anda dapat memeriksa elemen dalam set atau mencari substring dalam string dengan menggunakan fungsi `contains`. Jika ekspresi kondisi bernilai `true`, operasi berhasil; jika tidak, operasi akan gagal.

Contoh berikut menggunakan `contains` untuk menghapus produk hanya jika elemen Set String `Color` memiliki elemen dengan nilai tertentu.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam `expression-attribute-values` `file.json`.

```
{  
  ":v_sub":{"S":"Red"}  
}
```

Memeriksa ukuran nilai atribut

Anda dapat memeriksa ukuran nilai atribut dengan menggunakan fungsi `size`. Jika ekspresi kondisi bernilai `true`, operasi berhasil; jika tidak, operasi akan gagal.

Contoh berikut menggunakan `size` untuk menghapus produk hanya jika ukuran atribut `VideoClip` lebih besar dari `64000` byte.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam `expression-attribute-values file.json`.

```
{  
  ":v_sub":{"N":"64000"}  
}
```

Operator perbandingan dan referensi fungsi

Bagian ini mencakup fungsi dan kata kunci bawaan untuk menulis ekspresi filter dan ekspresi kondisi di Amazon DynamoDB. Untuk informasi lebih rinci mengenai fungsi dan pemrograman dengan DynamoDB, lihat [Pemrograman dengan DynamoDB dan SDK AWS](#) dan [Referensi API DynamoDB](#).

Topik

- [Sintaks untuk ekspresi filter dan kondisi](#)
- [Membuat perbandingan](#)
- [Fungsi](#)
- [Evaluasi logika](#)
- [Tanda kurung](#)
- [Prioritas dalam kondisi](#)

Sintaks untuk ekspresi filter dan kondisi

Dalam ringkasan sintaks berikut, *operand* dapat berupa yang berikut ini:

- Nama atribut tingkat atas, seperti `Id`, `Title`, `Description`, atau `ProductCategory`
- Jalur dokumen yang mereferensikan atribut bersarang

```
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)
```

Membuat perbandingan

Gunakan pembanding ini untuk membandingkan operan dengan rentang nilai atau daftar nilai yang disebutkan:

- $a = b$ — True jika a sama dengan b
- $a <> b$ — True jika a tidak sama dengan b
- $a < b$ — True jika a kurang dari b
- $a <= b$ — True jika a kurang dari atau sama dengan b
- $a > b$ — True jika a lebih besar dari b
- $a >= b$ — True jika a lebih besar dari atau sama dengan b

Gunakan kata kunci BETWEEN dan IN untuk membandingkan operan dengan rentang nilai atau daftar nilai yang disebutkan:

- a BETWEEN b AND c - True jika a lebih besar atau sama dengan b , dan kurang dari atau sama dengan c .

- $a \text{ IN } (b, c, d)$ – True jika a sama dengan nilai apa pun dalam daftar—misalnya, salah satu dari b , c , atau d . Daftar ini dapat berisi hingga 100 nilai, dipisahkan dengan koma.

Fungsi

Gunakan fungsi berikut untuk menentukan apakah atribut berada dalam item, atau untuk mengevaluasi nilai atribut. Nama fungsi ini peka huruf besar/kecil. Untuk atribut bersarang, Anda harus menyediakan jalur dokumen yang lengkap.

Fungsi	Deskripsi
<code>attribute_exists (<i>path</i>)</code>	<p>True jika item berisi atribut yang ditentukan oleh path.</p> <p>Contoh: Periksa apakah item dalam tabel Product memiliki tampilan tampak samping.</p> <ul style="list-style-type: none"> • <code>attribute_exists (#Picture s.#SideView)</code>
<code>attribute_not_exists (<i>path</i>)</code>	<p>True jika atribut yang ditentukan oleh path tidak ada dalam item.</p> <p>Contoh: Periksa apakah item memiliki atribut <code>Manufacturer</code> .</p> <ul style="list-style-type: none"> • <code>attribute_not_exists (Manufacturer)</code>
<code>attribute_type (<i>path</i>, <i>type</i>)</code>	<p>True jika atribut di jalur yang ditentukan merupakan tipe daya tertentu. Parameter <code>type</code> harus salah satu dari hal berikut:</p> <ul style="list-style-type: none"> • S – String • SS – Set string •

Fungsi	Deskripsi
	<ul style="list-style-type: none">N – Nomor•NS – Set nomor•B – Biner•BS – Set biner•BOOL – Boolean•NULL – Null•L – Daftar•M – Peta <p>Anda harus menggunakan nilai atribut ekspresi untuk parameter type.</p> <p>Contoh: Periksa apakah atribut <code>QuantityOnHand</code> berjenis Daftar. Dalam contoh ini, <code>:v_sub</code> adalah placeholder untuk string L.</p> <ul style="list-style-type: none">• <code>attribute_type (ProductReviews.FiveStar, :v_sub)</code> <p>Anda harus menggunakan nilai atribut ekspresi untuk parameter type.</p>

Fungsi	Deskripsi
<code>begins_with (<i>path</i>, <i>substr</i>)</code>	<p>True jika atribut yang ditentukan oleh <code>path</code> dimulai dengan substring tertentu.</p> <p>Contoh: Periksa apakah beberapa karakter pertama dari URL gambar tampilan depan adalah <code>http://</code>.</p> <ul style="list-style-type: none"><code>begins_with (Pictures.FrontView, :v_sub)</code> <p>Nilai atribut ekspresi <code>:v_sub</code> adalah placeholder untuk <code>http://</code>.</p>

Fungsi	Deskripsi
<code>contains (<i>path</i>, <i>operand</i>)</code>	<p>True jika atribut yang ditentukan oleh <code>path</code> adalah salah satu hal berikut:</p> <ul style="list-style-type: none">• <code>String</code> yang berisi substring tertentu.• <code>Set</code> yang berisi elemen tertentu dalam set.• <code>List</code> yang berisi elemen tertentu dalam daftar. <p>Jika atribut yang ditentukan oleh <code>path</code> adalah <code>String</code>, <code>operand</code> harus <code>String</code>. Jika atribut yang ditentukan oleh <code>path</code> adalah <code>Set</code>, <code>operand</code> harus merupakan jenis elemen set.</p> <p>Jalan dan operan harus berbeda. Artinya, <code>contains (a, a)</code> mengembalikan kesalahan .</p> <p>Contoh: Periksa apakah atribut <code>Brand</code> berisi substring <code>Company</code>.</p> <ul style="list-style-type: none">• <code>contains (Brand, :v_sub)</code> <p>Nilai atribut ekspresi <code>:v_sub</code> adalah placeholder untuk <code>Company</code>.</p> <p>Contoh: Periksa apakah produk tersedia dalam warna merah.</p> <ul style="list-style-type: none">• <code>contains (Color, :v_sub)</code>

Fungsi	Deskripsi
	Nilai atribut ekspresi :v_sub adalah placeholder untuk Red.

Fungsi	Deskripsi
<code>size (<i>path</i>)</code>	<p>Mengembalikan angka yang mewakili ukuran atribut ini. Berikut ini adalah tipe daya yang valid untuk digunakan dengan <code>size</code>.</p> <p>Jika atribut adalah jenis <code>String</code>, <code>size</code> menampilkan panjang string.</p> <p>Contoh: Periksa apakah string <code>Brand</code> kurang dari atau sama dengan 20 karakter. Nilai atribut ekspresi <code>:v_sub</code> adalah placeholder untuk 20.</p> <ul style="list-style-type: none"><code>size (Brand) <= :v_sub</code> <p>Jika atribut adalah jenis <code>Binary</code>, <code>size</code> menghasilkan jumlah byte dalam nilai atribut.</p> <p>Contoh: Misalkan <code>ProductCatalog</code> item memiliki atribut binari yang disebut <code>VideoClip</code> yang berisi video singkat dari produk yang digunakan. Ekspresi berikut memeriksa apakah <code>VideoClip</code> melebihi 64.000 byte. Nilai atribut ekspresi <code>:v_sub</code> adalah placeholder untuk 64000.</p> <ul style="list-style-type: none"><code>size(VideoClip) > :v_sub</code> <p>Jika atribut adalah tipe daya <code>Set</code>, <code>size</code> menghasilkan jumlah elemen dalam set.</p>

Fungsi	Deskripsi
	<p>Contoh: Periksa apakah produk tersedia dalam lebih dari satu warna. Nilai atribut ekspresi <code>:v_sub</code> adalah placeholder untuk 1.</p> <ul style="list-style-type: none"> <code>size (Color) < :v_sub</code> <p>Jika atribut adalah jenis <code>List</code> atau <code>Map</code>, <code>size</code> menghasilkan jumlah elemen turunan.</p> <p>Contoh: Periksa apakah jumlah ulasan <code>OneStar</code> telah melampaui ambang batas tertentu. Nilai atribut ekspresi <code>:v_sub</code> adalah placeholder untuk 3.</p> <ul style="list-style-type: none"> <code>size(ProductReviews.OneStar) > :v_sub</code>

Evaluasi logika

Gunakan kata kunci `AND`, `OR`, dan `NOT` untuk melakukan evaluasi logika. Dalam daftar berikut, `a` dan `b` mewakili kondisi yang akan dievaluasi.

- `a AND b` – True jika `a` dan `b` adalah True.
- `a OR b` – True jika `a` atau `b` (atau keduanya) adalah True.
- `NOT a` – True jika `a` adalah false. False jika `a` adalah true.

Berikut ini adalah contoh kode `AND` dalam suatu operasi.

```
dynamodb-local (*)> select * from exprtest where a > 3 and a < 5;
```

Tanda kurung

Gunakan tanda kurung untuk mengubah prioritas evaluasi logis. Sebagai contoh, misalkan kondisi *a* dan *b* adalah True, dan kondisi *c* adalah False. Ekspresi berikut bernilai True:

- *a* OR *b* AND *c*

Namun, jika Anda mengapit kondisi dalam tanda kurung, kondisi tersebut akan dievaluasi terlebih dahulu. Misalnya, nilai berikut bernilai false:

- (*a* OR *b*) AND *c*

Note

Anda dapat menyanggah tanda kurung dalam sebuah ekspresi. Bagian paling dalam akan dievaluasi terlebih dahulu.

Berikut ini adalah contoh kode dengan tanda kurung dalam evaluasi logis.

```
dynamodb-local (*)> select * from exprtest where attribute_type(b, string)
or ( a = 5 and c = "coffee");
```

Prioritas dalam kondisi

DynamoDB mengevaluasi kondisi dari kiri ke kanan menggunakan aturan prioritas berikut:

- = <> < <= > >=
- IN
- BETWEEN
- attribute_exists attribute_not_exists begins_with contains
- Tanda kurung
- NOT
- AND
- OR

Ekspresi pembaruan

Operasi `UpdateItem` memperbarui item yang sudah ada, atau menambahkan item baru ke tabel jika belum ada. Anda harus memberikan kunci item yang ingin Anda perbarui. Anda juga harus menyediakan ekspresi pembaruan, yang menunjukkan atribut yang ingin Anda ubah dan nilai yang ingin Anda tetapkan padanya.

Ekspresi pembaruan menentukan cara `UpdateItem` mengubah atribut item—misalnya, menetapkan nilai skalar atau menghapus elemen dari daftar atau peta.

Berikut ini adalah ringkasan sintaksis untuk ekspresi pembaruan.

```
update-expression ::=  
[ SET action [, action] ... ]  
[ REMOVE action [, action] ... ]  
[ ADD action [, action] ... ]  
[ DELETE action [, action] ... ]
```

Ekspresi pembaruan terdiri dari satu atau lebih klausa. Setiap klausa dimulai dengan kata kunci SET, REMOVE, ADD, atau DELETE. Anda dapat memasukkan salah satu klausa ini dalam ekspresi pembaruan, dalam urutan apa pun. Namun, setiap kata kunci tindakan hanya dapat muncul satu kali.

Dalam setiap klausa, ada satu atau lebih tindakan yang dipisahkan dengan koma. Setiap tindakan mewakili modifikasi data.

Contoh dalam bagian ini didasarkan pada item `ProductCatalog` yang ditampilkan dalam [Ekspresi proyeksi](#).

Topik di bawah ini mencakup beberapa kasus penggunaan yang berbeda untuk SET tindakan tersebut.

Topik

- [SET — mengubah atau menambahkan atribut item](#)
- [REMOVE—menghapus atribut dari item](#)
- [ADD — memperbarui momor dan set](#)
- [DELETE — menghapus elemen dari set](#)
- [Menggunakan beberapa ekspresi pembaruan](#)

SET — mengubah atau menambahkan atribut item

Gunakan tindakan SET dalam ekspresi pembaruan untuk menambahkan satu atau beberapa atribut ke item. Jika salah satu atribut ini sudah ada, atribut tersebut akan ditimpa dengan nilai baru.

Anda juga dapat menggunakan SET untuk menambahkan atau mengurangi dari atribut yang berjenis `Number`. Untuk melakukan beberapa tindakan SET, pisahkan dengan koma.

Dalam ringkasan sintaks berikut:

- Elemen *jalur* adalah jalur dokumen ke item.
- Elemen *operand* dapat berupa jalur dokumen ke item atau fungsi.

```
set-action ::=
  path = value

value ::=
  operand
  | operand '+' operand
  | operand '-' operand

operand ::=
  path | function
```

Operasi `PutItem` berikut membuat item sampel yang dirujuk oleh contoh.

```
aws dynamodb put-item \
  --table-name ProductCatalog \
  --item file://item.json
```

Argumen untuk `--item` disimpan dalam file `item.json`. (Untuk kemudahan, hanya beberapa atribut item yang digunakan.)

```
{
  "Id": {"N": "789"},
  "ProductCategory": {"S": "Home Improvement"},
  "Price": {"N": "52"},
  "InStock": {"BOOL": true},
  "Brand": {"S": "Acme"}
}
```

Topik

- [Memodifikasi atribut](#)
- [Menambahkan daftar dan peta](#)
- [Menambahkan elemen ke daftar](#)
- [Menambahkan atribut peta bersarang](#)
- [Menambahkan dan mengurangi atribut numerik](#)
- [Melekatkan elemen ke daftar](#)
- [Mencegah penimpaan atribut yang ada](#)

Memodifikasi atribut

Example

Perbarui atribut ProductCategory dan Price.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{  
  ":c": { "S": "Hardware" },  
  ":p": { "N": "60" }  
}
```

Note

Dalam operasi `UpdateItem`, `--return-values ALL_NEW` menyebabkan DynamoDB untuk mengembalikan item seperti yang muncul setelah pembaruan.

Menambahkan daftar dan peta

Example

Tambahkan daftar baru dan peta baru.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{  
  ":ri": {  
    "L": [  
      { "S": "Hammer" }  
    ]  
  },  
  ":pr": {  
    "M": {  
      "FiveStar": {  
        "L": [  
          { "S": "Best product ever!" }  
        ]  
      }  
    }  
  }  
}
```

Menambahkan elemen ke daftar

Example

Tambahkan atribut baru ke daftar `ReLatedItems`. (Ingatlah bahwa elemen daftar berbasis nol, sehingga `[0]` mewakili elemen pertama dalam daftar, `[1]` mewakili elemen kedua, dan seterusnya.)

```
aws dynamodb update-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"789"}}' \  
  --update-expression "SET RelatedItems[0] = :ri"
```



```
--update-expression "SET RelatedItems[1] = :ri" \  
--expression-attribute-values file://values.json \  
--return-values ALL_NEW
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{  
  ":ri": { "S": "Nails" }  
}
```

Note

Saat Anda menggunakan SET untuk memperbarui elemen daftar, konten elemen akan diganti dengan data baru yang Anda tentukan. Jika elemen tersebut belum ada, SET menambahkan elemen baru pada akhir daftar.

Jika Anda menambahkan beberapa elemen dalam satu operasi SET, elemen diurutkan berdasarkan nomor elemen.

Menambahkan atribut peta bersarang

Example

Tambahkan beberapa atribut peta bersarang.

```
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
```

Argumen untuk `--expression-attribute-names` disimpan dalam file `names.json`.

```
{  
  "#pr": "ProductReviews",  
  "#5star": "FiveStar",  
  "#3star": "ThreeStar"  
}
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
  ":r5": { "S": "Very happy with my purchase" },
  ":r3": {
    "L": [
      { "S": "Just OK - not that great" }
    ]
  }
}
```

Menambahkan dan mengurangi atribut numerik

Anda dapat menambahkan atau mengurangi dari atribut numerik yang sudah ada. Untuk melakukannya, gunakan operator `+` (plus) dan `-` (minus).

Example

Kurangi Price dari item.

```
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
```

Untuk meningkatkan Price, Anda akan menggunakan operator `+` dalam ekspresi pembaruan.

Melekatkan elemen ke daftar

Anda dapat menambahkan elemen ke akhir daftar. Untuk melakukannya, gunakan SET dengan fungsi `list_append`. (Nama fungsi ini peka huruf besar/kecil.) Fungsi `list_append` khusus untuk tindakan SET dan hanya dapat digunakan dalam ekspresi pembaruan. Sintaksnya adalah sebagai berikut.

- `list_append (list1, list2)`

Fungsi ini mengambil dua daftar sebagai input dan menambahkan semua elemen dari *list2* ke *list1*.

Example

Dalam [Menambahkan elemen ke daftar](#), Anda mencantumkan RelatedItems dan mengisinya dengan dua elemen: Hammer dan Nails. Sekarang Anda menambahkan dua elemen lagi ke akhir RelatedItems.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{  
  ":vals": {  
    "L": [  
      { "S": "Screwdriver" },  
      { "S": "Hacksaw" }  
    ]  
  }  
}
```

Akhirnya, Anda menambahkan satu elemen lagi ke awal dari RelatedItems. Untuk melakukan hal ini, tukar urutan elemen `list_append`. (Ingatlah bahwa `list_append` mengambil dua daftar sebagai input dan menambahkan daftar kedua ke yang pertama.)

```
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
```

Atribut RelatedItems yang dihasilkan sekarang berisi lima elemen, dalam urutan berikut: Chisel, Hammer, Nails, Screwdriver, Hacksaw.

Mencegah penimpaan atribut yang ada

Jika Anda ingin menghindari penimpaan atribut yang ada, Anda dapat menggunakan SET dengan fungsi `if_not_exists`. (Nama fungsi ini peka huruf besar/kecil.) Fungsi `if_not_exists` khusus untuk tindakan SET dan hanya dapat digunakan dalam ekspresi pembaruan. Sintaksnya adalah sebagai berikut.

- `if_not_exists` (*path*, *value*)

Jika item tidak berisi atribut pada *path* yang ditentukan, `if_not_exists` mengevaluasi ke *value*; jika tidak, itu mengevaluasi *path*.

Example

Tetapkan `Price` dari suatu item, tetapi hanya jika item tersebut belum memiliki atribut `Price`. (Jika `Price` sudah ada, tidak ada yang terjadi.)

```
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—menghapus atribut dari item

Gunakan tindakan REMOVE dalam ekspresi pembaruan untuk menghapus satu atau beberapa atribut dari item di Amazon DynamoDB. Untuk melakukan beberapa tindakan REMOVE, pisahkan dengan koma.

Berikut ini adalah ringkasan sintaks untuk REMOVE dalam ekspresi pembaruan. Satu-satunya operand adalah jalur dokumen untuk atribut yang ingin Anda hapus.

```
remove-action ::=  
  path
```

Example

Hapus beberapa atribut dari item. (Jika atribut tidak ada, tidak ada yang terjadi.)

```
aws dynamodb update-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"789"}}' \  
  --update-expression "REMOVE Brand, InStock, QuantityOnHand" \  
  --return-values ALL_NEW
```

Menghapus elemen dari daftar

Anda dapat menggunakan REMOVE untuk menghapus elemen individu dari daftar.

Example

Dalam [Melekatkan elemen ke daftar](#), Anda mengubah atribut daftar (RelatedItems) sehingga berisi lima elemen:

- [0]—Chisel
- [1]—Hammer
- [2]—Nails
- [3]—Screwdriver
- [4]—Hacksaw

Berikut AWS Command Line Interface (AWS CLI) contoh menghapus Hammer dan Nails dari daftar.

```
aws dynamodb update-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"789"}}' \  
  --update-expression "REMOVE RelatedItems[1], RelatedItems[2]" \  
  --return-values ALL_NEW
```

Setelah Hammer dan Nails dihapus, elemen yang tersisa digeser. Daftarnya sekarang berisi hal-hal berikut:

- [0]—Chisel
- [1]—Screwdriver
- [2]—Hacksaw

ADD — memperbarui momor dan set

Note

Secara umum, sebaiknya gunakan SET daripada ADD.

Gunakan tindakan ADD dalam ekspresi pembaruan untuk menambahkan atribut baru dan nilainya untuk item.

Jika atribut sudah ada, perilaku ADD bergantung pada tipe daya atribut ini:

- Jika atribut adalah angka, dan nilai yang Anda tambahkan juga angka, nilai matematis ditambahkan ke atribut yang ada. (Jika nilai adalah angka negatif, nilai tersebut dikurangi dari atribut yang ada.)
- Jika atribut adalah suatu set, dan nilai yang Anda tambahkan juga suatu set, nilai ditambahkan ke set yang ada.

Note

Tindakan ADD hanya mendukung tipe daya angka dan set.

Untuk melakukan beberapa tindakan ADD, pisahkan dengan koma.

Dalam ringkasan sintaks berikut:

- Elemen *jalur* adalah jalur dokumen ke atribut. Atribut harus berupa Number atau tipe daya set.
- Elemen *nilai* adalah angka yang ingin Anda tambahkan ke atribut (untuk tipe daya Number), atau suatu set untuk ditambahkan ke atribut (untuk jenis set).

```
add-action ::=  
  path value
```

Topik di bawah ini mencakup beberapa kasus penggunaan yang berbeda untuk ADD tindakan tersebut.

Topik

- [Menambahkan nomor](#)
- [Menambahkan elemen ke set](#)

Menambahkan nomor

Anggaplah atribut `QuantityOnHand` tidak ada. AWS CLI Contoh berikut ditetapkan `QuantityOnHand` ke 5.

```
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
```

Kini setelah `QuantityOnHand` ada, Anda dapat menjalankan kembali contoh untuk menaikkan `QuantityOnHand` sebesar 5 setiap kali melakukannya.

Menambahkan elemen ke set

Anggaplah atribut `Color` tidak ada. Contoh AWS CLI berikut menetapkan `Color` untuk satu set string dengan dua elemen.

```
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
```

Kini setelah `Color` ada, Anda dapat menambahkan lebih banyak elemen untuknya.

```
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 — menghapus elemen dari set

Important

Tindakan DELETE hanya mendukung tipe daya Set.

Gunakan tindakan DELETE dalam ekspresi pembaruan untuk menghapus satu atau beberapa elemen dari satu set. Untuk melakukan beberapa tindakan DELETE, pisahkan dengan koma.

Dalam ringkasan sintaks berikut:

- Elemen *jalur* adalah jalur dokumen ke atribut. Atribut harus berupa tipe daya set.
- *Subset* adalah satu atau beberapa elemen yang ingin Anda hapus dari *jalur*. Anda harus menentukan *subset* sebagai jenis set.

```
delete-action ::=  
  path subset
```

Example

Dalam [Menambahkan elemen ke set](#), Anda membuat set string Color. Contoh ini akan menghapus beberapa elemen dari set tersebut.

```
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
```

Menggunakan beberapa ekspresi pembaruan

Anda dapat menggunakan beberapa ekspresi pembaruan dalam pernyataan tunggal.

Example

Jika Anda ingin mengubah nilai atribut dan menghapus atribut lain sepenuhnya, Anda dapat menggunakan tindakan SET dan REMOVE dalam satu pernyataan. Operasi ini akan mengurangi Price nilai menjadi 15 sekaligus menghapus InStock atribut dari item.


```
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

Jika Anda ingin menambahkan ke daftar sambil juga mengubah nilai atribut lain, Anda bisa menggunakan dua tindakan SET dalam satu pernyataan. Operasi ini akan menambahkan “Nails” ke atribut daftar RelatedItems dan juga menetapkan nilai Price ke 21.

```
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
```

Waktu untuk Hidup (TTL)

Time To Live (TTL) untuk DynamoDB adalah metode hemat biaya untuk menghapus item yang tidak lagi relevan. TTL memungkinkan Anda untuk menentukan stempel waktu kedaluwarsa per item yang menunjukkan kapan item tidak lagi diperlukan. DynamoDB secara otomatis menghapus item yang kedaluwarsa dalam beberapa hari dari waktu kedaluwarsa, tanpa menghabiskan throughput penulisan.

Untuk menggunakan TTL, pertama-tama aktifkan pada tabel dan kemudian tentukan atribut tertentu untuk menyimpan stempel waktu kedaluwarsa TTL. Stempel waktu harus disimpan dalam [format waktu epoch Unix](#) pada perincian detik. Setiap kali item dibuat atau diperbarui, Anda dapat menghitung waktu kedaluwarsa dan menyimpannya di atribut TTL.

Item dengan atribut TTL yang valid dan kedaluwarsa dapat dihapus oleh sistem kapan saja, biasanya dalam beberapa hari setelah kedaluwarsa. Anda masih dapat memperbarui item kedaluwarsa yang menunggu penghapusan, termasuk mengubah atau menghapus atribut TTL mereka. Saat memperbarui item yang kedaluwarsa, kami menyarankan Anda menggunakan ekspresi kondisi untuk memastikan item tersebut belum dihapus selanjutnya. Gunakan ekspresi filter untuk menghapus item kedaluwarsa dari hasil [Pindai](#) dan [Kueri](#).

Item yang dihapus bekerja sama dengan yang dihapus melalui operasi penghapusan tipikal. Setelah dihapus, item masuk ke DynamoDB Streams sebagai penghapusan layanan, bukan penghapusan pengguna, dan dihapus dari indeks sekunder lokal dan indeks sekunder global seperti operasi penghapusan lainnya.

Jika Anda menggunakan [Tabel Global versi 2019.11.21 \(Saat ini\)](#) dari tabel global dan Anda juga menggunakan fitur TTL, DynamoDB mereplikasi penghapusan TTL ke semua tabel replika. Penghapusan TTL awal tidak menggunakan Write Capacity Units (WCU) di wilayah di mana TTL kedaluwarsa terjadi. Namun, penghapusan TTL yang direplikasi ke tabel replika menggunakan Unit Kapasitas Tulis yang direplikasi saat menggunakan kapasitas yang disediakan, atau Unit Tulis Replikasi saat menggunakan mode kapasitas sesuai permintaan, di setiap wilayah replika dan biaya yang berlaku akan berlaku.

Untuk informasi selengkapnya tentang TTL, lihat topik berikut:

Topik

- [Mengaktifkan waktu untuk hidup \(TTL\)](#)
- [Menghitung waktu untuk hidup \(TTL\)](#)
- [Bekerja dengan item kedaluwarsa](#)

Mengaktifkan waktu untuk hidup (TTL)

TTL dapat diaktifkan di Amazon DynamoDB Console, AWS Command Line Interface, (), AWS CLI atau menggunakan Referensi [Amazon DynamoDB API dengan SDK mana pun](#) yang seharusnya. AWS Dibutuhkan sekitar satu jam untuk TTL diaktifkan di semua partisi.

Aktifkan DynamoDB TTL menggunakan konsol AWS

1. Masuk ke AWS Management Console, lalu buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Pilih Tabel, lalu pilih tabel yang ingin Anda ubah.
3. Di tab Pengaturan tambahan, di bagian Time to Live (TTL), pilih Aktifkan untuk mengaktifkan TTL.

Read/write capacity Edit

The read/write capacity mode controls how you are charged for read and write throughput and how you manage capacity.

Capacity mode
Provisioned

Table capacity

Read capacity auto scaling On	Write capacity auto scaling On
Provisioned read capacity units 5	Provisioned write capacity units 5
Provisioned range for reads 1 - 10	Provisioned range for writes 1 - 10
Target read capacity utilization 70%	Target write capacity utilization 70%

► Estimated read/write capacity cost

Auto scaling activities (0) Refresh

Recent events of automatic scaling. [Learn more](#)

Find events

Start time	End time	Target	Capacity unit	Description	Status
No auto scaling activities found					

There are no auto scaling activities for the table or its global secondary indexes.

Time to Live (TTL) [Info](#) Run preview Turn on

Automatically delete expired items from a table.

TTL status
Off

4. Saat mengaktifkan TTL pada tabel, DynamoDB mengharuskan Anda mengidentifikasi nama atribut tertentu yang akan dicari layanan saat menentukan apakah suatu item memenuhi syarat untuk kedaluwarsa. Nama atribut TTL, yang ditunjukkan di bawah ini, peka huruf besar/kecil dan harus cocok dengan atribut yang ditentukan dalam operasi baca dan tulis Anda. Ketidakcocokan akan menyebabkan item yang kedaluwarsa tidak dihapus. Mengganti nama atribut TTL mengharuskan Anda untuk menonaktifkan TTL dan kemudian mengaktifkannya kembali dengan atribut baru di masa mendatang. TTL akan terus memproses penghapusan selama kurang lebih 30 menit setelah dinonaktifkan. TTL harus dikonfigurasi ulang pada tabel yang dipulihkan.

[DynamoDB](#) > [Tables](#) > [Music](#) > Turn on Time to Live (TTL)

Turn on Time to Live (TTL) [Info](#)

TTL settings

TTL attribute name

The name of the attribute that will be stored in the TTL timestamp.

Between 1 and 255 characters.

Preview

Confirm that your TTL attribute and values are working properly by specifying a date and time, and reviewing a sample of the items that will be deleted by then. Note that preview may show only some of the relevant items.


Simulated date and time

Specify the date and time to simulate which items would be expired.

Epoch time value ▼

September 13, 2023, 15:28:52 (UTC-06:00)

Run preview

 Activating TTL can take up to one hour to be applied across all partitions. You will not be able to make additional TTL changes until this update is complete.

Cancel

Turn on TTL

5. (Opsional) Anda dapat melakukan tes dengan mensimulasikan tanggal dan waktu kedaluwarsa dan mencocokkan beberapa item. Ini memberi Anda daftar sampel item dan mengonfirmasi bahwa ada item yang berisi nama atribut TTL yang disediakan bersama dengan waktu kedaluwarsa.

Turn on Time to Live (TTL) [Info](#)

TTL settings

TTL attribute name

The name of the attribute that will be stored in the TTL timestamp.

Between 1 and 255 characters.

Preview

Confirm that your TTL attribute and values are working properly by specifying a date and time, and reviewing a sample of the items that will be deleted by then. Note that preview may show only some of the relevant items.

Simulated date and time

Specify the date and time to simulate which items would be expired.

Epoch time value ▼

December 11, 2023, 16:58:01 (UTC-07:00)

Run preview

Items to be deleted (32)

artist	album	createdAt	expireAt (TTL)
f91897e5-0...	72499653-...	1694559481	<u>1702339081</u>
7d38838f-e...	64b6999b-...	1694559479	<u>1702339079</u>
6734d779-...	52d667bd-...	1694559481	<u>1702339081</u>
4553fb30-...	bb2cc547-e...	1694559481	<u>1702339081</u>
ea7c0eeb-5...	840b3c7b-...	1694559478	<u>1702339078</u>

Setelah TTL diaktifkan, atribut TTL ditandai TTL saat Anda melihat item di konsol DynamoDB. Anda dapat melihat tanggal dan waktu kedaluwarsa suatu item dengan mengarahkan kursor ke atribut tersebut.

Items returned (100)

Actions Create item

< 1 > ⚙️

<input type="checkbox"/>	artist (String)	album (String)	createdAt	expireAt (TTL)
<input type="checkbox"/>	f91897e5-0a7e-4ee8-a9be-561ec...	72499653-50fd-454f-9ed0-496...	1694559481	1702339081
<input type="checkbox"/>	7d38838f-e904-4673-96ba-ab29c...	64b6999b-80aa-46d6-b567-c6f...	1694559479	1702339079
<input type="checkbox"/>	9da8f8a1-d920-41e2-8469-88fa8...	e8cb4ef3-8d22-4f5b-96f3-e79c...	1694559478	1702339078
<input type="checkbox"/>	6734d779-5d71-47f3-ae4a-4b617...	52d667bd-cd9d-48a4-9a66-3bf...	1694559477	1702339077
<input type="checkbox"/>	cdb74466-0b36-41cd-9b39-cbe41...	52965e04-cb1a-4089-b891-9a1...	1694559476	1702339076
<input type="checkbox"/>	70aba065-a9d3-40f3-bd64-0d34c...	3272c168-4de2-4edf-a253-e02...	1694559475	1702339075
<input type="checkbox"/>	54caf925-843f-4966-b1e3-95530...	5e723d06-877d-4572-808b-e8d...	1694559474	1702339074
<input type="checkbox"/>	4af50ef7-8c8e-4cc3-ad61-9eb3b5...	8c3dfc04-7091-4557-b287-67ca...	1694559486	1702339086
<input type="checkbox"/>	f4d6f592-2b42-4b88-9551-ebad3...	0f9c7f08-667a-4577-997a-ee51...	1694559487	1702339087

UTC

December 11, 2023 23:58:06 UTC

Local

December 11, 2023 16:58:06 MST

Region (N. Virginia)

December 11, 2023 18:58:06 EST

Aktifkan DynamoDB TTL menggunakan API

Python

Anda dapat mengaktifkan TTL dengan kode, menggunakan [UpdateTimeToLive](#) operasi.

```
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')

```

Anda dapat mengonfirmasi TTL diaktifkan dengan menggunakan [DescribeTimeToLive](#) operasi, yang menggambarkan status TTL di atas meja. `TimeToLiveStatusnya` adalah salah satu `ENABLED` atau `DISABLED`.

```

# 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

Anda dapat mengaktifkan TTL dengan kode, menggunakan [UpdateTimeToLiveCommand](#) operasi.

```

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');
```

Aktifkan Waktu untuk Hidup menggunakan AWS CLI

1. Aktifkan TTL pada tabel TTLExample.

```
aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
specification "Enabled=true, AttributeName=ttl"
```

2. Jelaskan TTL pada tabel TTLExample.

```
aws dynamodb describe-time-to-live --table-name TTLExample
{
  "TimeToLiveDescription": {
    "AttributeName": "ttl",
    "TimeToLiveStatus": "ENABLED"
  }
}
```

3. Tambahkan item ke tabel TTLExample dengan atribut Waktu untuk Tayang yang diatur menggunakan shell BASH dan AWS CLI.

```
EXP=`date -d '+5 days' +%s`
```



```
aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl": {"N": "'$EXP'}}'
```

Contoh ini dimulai dengan tanggal saat ini dan menambahkan 5 hari untuk membuat waktu kedaluwarsa. Kemudian, contoh ini mengubah waktu kedaluwarsa menjadi format jangka waktu yang pada akhirnya menambahkan item ke "TTLExample".

Note

Salah satu cara untuk menetapkan nilai kedaluwarsa Waktu untuk Tayang adalah dengan menghitung jumlah detik yang ditambahkan ke waktu kedaluwarsa. Misalnya, 5 hari adalah 432.000 detik. Namun, seringkali lebih baik memulai dengan suatu tanggal dan bekerja dari sana.

Cukup mudah untuk mendapatkan waktu saat ini dalam format jangka waktu, seperti pada contoh berikut.

- Terminal Linux: `date +%s`
- Python: `import time; int(time.time())`
- Java: `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

Aktifkan DynamoDB TTL menggunakan AWS CloudFormation

1. Aktifkan TTL pada tabel TTLExample.

```
aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-specification "Enabled=true, AttributeName=ttl"
```

2. Jelaskan TTL pada tabel TTLExample.

```
aws dynamodb describe-time-to-live --table-name TTLExample
{
  "TimeToLiveDescription": {
    "AttributeName": "ttl",
    "TimeToLiveStatus": "ENABLED"
  }
}
```

```
}
```

3. Tambahkan item ke tabel `TTLExample` dengan atribut Waktu untuk Tayang yang diatur menggunakan shell BASH dan AWS CLI.

```
EXP=`date -d '+5 days' +%s`  
aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl": {"N": "'$EXP'"}'}
```

Contoh ini dimulai dengan tanggal saat ini dan menambahkan 5 hari untuk membuat waktu kedaluwarsa. Kemudian, contoh ini mengubah waktu kedaluwarsa menjadi format jangka waktu yang pada akhirnya menambahkan item ke "TTLExample".

Note

Salah satu cara untuk menetapkan nilai kedaluwarsa Waktu untuk Tayang adalah dengan menghitung jumlah detik yang ditambahkan ke waktu kedaluwarsa. Misalnya, 5 hari adalah 432.000 detik. Namun, seringkali lebih baik memulai dengan suatu tanggal dan bekerja dari sana.

Cukup mudah untuk mendapatkan waktu saat ini dalam format jangka waktu, seperti pada contoh berikut.

- Terminal Linux: `date +%s`
- Python: `import time; int(time.time())`
- Java: `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

Menghitung waktu untuk hidup (TTL)

Cara umum untuk mengimplementasikan TTL adalah dengan menetapkan waktu kedaluwarsa untuk item berdasarkan kapan mereka dibuat atau terakhir diperbarui. Ini dapat dilakukan dengan menambahkan waktu ke `createdAt` dan `updatedAt` stempel waktu. Misalnya, TTL untuk item yang baru dibuat dapat diatur `createdAt` ke +90 hari. Saat item diperbarui, TTL dapat dihitung ulang menjadi +90 hari. `updatedAt`

Waktu kedaluwarsa yang dihitung harus dalam format epoch, dalam hitungan detik. Untuk dipertimbangkan untuk kedaluwarsa dan penghapusan, TTL tidak boleh lebih dari lima tahun yang lalu. Jika Anda menggunakan format lain, proses TTL akan mengabaikan item. Jika Anda menetapkan tanggal kedaluwarsa ke sometime in the future ketika Anda ingin item kedaluwarsa, item akan kedaluwarsa setelah waktu itu. Misalnya, Anda menetapkan tanggal kedaluwarsa ke 1724241326 (yaitu Senin, 21 Agustus 2024 11:55:26 (GMT)). Item akan kedaluwarsa setelah waktu yang ditentukan.

Topik

- [Buat item dan atur Waktu ke Hidup](#)
- [Perbarui item dan segarkan Waktu untuk Hidup](#)

Buat item dan atur Waktu ke Hidup

Contoh berikut menunjukkan bagaimana menghitung waktu kedaluwarsa saat membuat item baru, menggunakan `expireAt` sebagai nama atribut TTL. Pernyataan penugasan memperoleh waktu saat ini sebagai variabel. Dalam contoh, waktu kedaluwarsa dihitung sebagai 90 hari dari waktu saat ini. Waktu kemudian dikonversi ke format epoch dan disimpan sebagai tipe data integer dalam atribut TTL.

Python

```
import boto3
from datetime import datetime, timedelta

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,
    'createdAt': current_time,
    'expiredAt': expiration_time
}

table.put_item(Item=item)

print("Item created successfully.")
except Exception as e:
    print(f"Error creating item: {e}")
    raise

# Use your own values
create_dynamodb_item('your-table-name', 'us-west-2', 'your-partition-key-value',
    'your-sort-key-value')
```

Javascript

Dalam permintaan ini kami menambahkan logika untuk menghitung waktu kedaluwarsa item yang baru dibuat:

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

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, ex);
    throw err;
  } else {
    console.log("Item created successfully: %s.", data);
    return data;
  }
});
}

// use your own values
createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
'your-sort-key-value');
```

Perbarui item dan segarkan Waktu untuk Hidup

Contoh ini merupakan kelanjutan dari yang dari [bagian sebelumnya](#). Waktu kedaluwarsa dapat dihitung ulang jika item diperbarui. Contoh berikut menghitung ulang `expireAt` stempel waktu menjadi 90 hari dari waktu saat ini.

Python

```
import boto3
from datetime import datetime, timedelta

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')
```

Output dari operasi pembaruan menunjukkan bahwa, sementara `createdAt` waktu tidak berubah, `updatedAt` dan `expireAt` waktu telah diperbarui. `expireAt` Waktu sekarang diatur ke 90 hari dari waktu pembaruan terakhir, yaitu Kamis, 19 Oktober 2023 pukul 13:27:15.

partition_key	createdAt	updatedAt	Kedaluwarsa	atribut_1	atribut_2
beberapa_ nilai	2023-07-17 14:11:05.322 323	2023-07-19 13:27:15.213 423	1697722035	new_value	beberapa_ nilai

Javascript

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

async function updateDynamoDBItem(tableName, region, partitionKey, sortKey) {
  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); //is there a better way to do this?

  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;
}

//enter your values here
updateDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
  'your-sort-key-value');
```

Contoh TTL yang dibahas dalam pendahuluan ini menunjukkan metode untuk memastikan hanya item yang baru diperbarui disimpan dalam tabel. Item yang diperbarui memiliki masa pakainya diperpanjang, sedangkan item yang tidak diperbarui pasca-pembuatan kedaluwarsa dan dihapus tanpa biaya, mengurangi penyimpanan dan mempertahankan tabel bersih.

Bekerja dengan item kedaluwarsa

Item kedaluwarsa yang tertunda penghapusan dapat difilter dari operasi baca dan tulis. Ini berguna dalam skenario ketika data kedaluwarsa tidak lagi valid dan tidak boleh digunakan. Jika tidak difilter, mereka akan terus ditampilkan dalam operasi baca dan tulis sampai dihapus oleh proses latar belakang.

Note

Barang-barang ini masih diperhitungkan untuk penyimpanan dan biaya baca sampai dihapus.

Penghapusan TTL dapat diidentifikasi di DynamoDB Streams, tetapi hanya di Wilayah tempat penghapusan terjadi. Penghapusan TTL yang direplikasi ke wilayah tabel global tidak dapat diidentifikasi dalam aliran DynamoDB di wilayah tempat penghapusan direplikasi.

Filter item kedaluwarsa dari operasi baca

Untuk operasi baca seperti [Pindai](#) dan [Kueri](#), ekspresi filter dapat memfilter item kedaluwarsa yang menunggu penghapusan. Seperti yang ditunjukkan pada cuplikan kode di bawah ini, ekspresi filter

dapat menyaring item di mana waktu TTL sama dengan atau kurang dari waktu saat ini. Hal ini dilakukan dengan pernyataan penugasan yang memperoleh waktu saat ini sebagai variabel (`now`), yang dikonversi ke format `int` waktu epoch.

Python

```
import boto3
from datetime import datetime

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')

```

Output dari operasi pembaruan menunjukkan bahwa, sementara `createdAt` waktu tidak berubah, `updatedAt` dan `expireAt` waktu telah diperbarui. `expireAt` waktu sekarang diatur ke 90 hari dari waktu pembaruan terakhir, yaitu Kamis, 19 Oktober 2023 pukul 13:27:15.

partition_key	createdAt	updatedAt	Kedaluwarsa	atribut_1	atribut_2
beberapa_ nilai	2023-07-17 14:11:05.322 323	2023-07-19 13:27:15.213 423	1697722035	new_value	beberapa_ nilai

Javascript

```

import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

async function queryDynamoDBItems(tableName, region, primaryKey) {
    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;
    }
}

//enter your own values here
queryDynamoDBItems('your-table-name', 'your-partition-key-value');

```

Menulis secara kondisional ke item yang kedaluwarsa

Ekspresi kondisi dapat digunakan untuk menghindari penulisan terhadap item yang kedaluwarsa. Cuplikan kode di bawah ini adalah pembaruan bersyarat yang memeriksa apakah waktu kedaluwarsa lebih besar dari waktu saat ini. Jika benar, operasi tulis akan berlanjut.

Python

```

import boto3
from datetime import datetime, timedelta
from botocore.exceptions import ClientError

def update_dynamodb_item(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 'ttlExpirationDate' is not
expired
    condition_expression = "ttlExpirationDate > :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 'ttlExpirationDate' 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('your-table-name', 'us-east-1', 'your-partition-key-value',
  'your-sort-key-value',
  'your-ttl-attribute-value')
```

Output dari operasi pembaruan menunjukkan bahwa, sementara `createdAt` waktu tidak berubah, `updatedAt` dan `expireAt` waktu telah diperbarui. `expireAt` Waktu sekarang diatur ke 90 hari dari waktu pembaruan terakhir, yaitu Kamis, 19 Oktober 2023 pukul 13:27:15.

partition_key	createdAt	updatedAt	Kedaluwarsa	atribut_1	atribut_2
beberapa_ nilai	2023-07-17 14:11:05.322 323	2023-07-19 13:27:15.213 423	1697722035	new_value	beberapa_ nilai

Javascript

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

// Example function to update an item in a DynamoDB table.
// The function should take the table name, region, partition key, sort key, and
// new attribute as arguments.
// The function should use the DynamoDB client to update the item.
// The function should return the updated item.
// The function should handle errors and exceptions.
const updateDynamoDBItem = async (tableName, region, partitionKey, sortKey,
  newAttribute) => {
  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;
}
};

// Enter your values here
updateDynamoDBItem('your-table-name', "us-east-1", 'your-partition-key-value', 'your-sort-key-value', 'your-new-attribute-value');
```

Mengidentifikasi item yang dihapus di DynamoDB Streams

Catatan stream berisi bidang identitas pengguna `Records[<index>].userIdentity`. Item yang dihapus oleh proses TTL memiliki bidang berikut:

```
Records[<index>].userIdentity.type
"Service"
```

```
Records[<index>].userIdentity.principalId
"dynamodb.amazonaws.com"
```

JSON berikut menunjukkan bagian yang relevan dari catatan aliran tunggal:

```
"Records": [  
  {  
    ...  
    "userIdentity": {  
      "type": "Service",  
      "principalId": "dynamodb.amazonaws.com"  
    }  
    ...  
  }  
]
```

Bekerja dengan item: Java

Anda dapat menggunakan AWS SDK for Java Document API untuk melakukan operasi pembuatan, pembacaan, pembaruan, dan penghapusan (CRUD) yang biasa pada item Amazon DynamoDB dalam tabel.

Note

SDK untuk Java juga menyediakan model persistensi objek, memungkinkan Anda memetakan kelas sisi klien ke tabel DynamoDB. Pendekatan ini dapat mengurangi jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat [Java 1.x: DynamoDBMapper](#).

Bagian ini berisi contoh Java untuk melakukan beberapa tindakan item Java Document API dan beberapa contoh kerja yang lengkap.

Topik

- [Menempatkan item](#)
- [Mendapatkan item](#)
- [Penulisan batch: Menempatkan dan menghapus beberapa item](#)
- [Batch get: Mendapatkan beberapa item](#)
- [Memperbarui Item](#)
- [Menghapus item](#)
- [Contoh: Operasi CRUD menggunakan AWS SDK for Java document API](#)

- [Contoh: Operasi batch menggunakan AWS SDK for Java document API](#)
- [Contoh: Penanganan atribut jenis biner menggunakan AWS SDK for Java document API](#)

Menempatkan item

Metode `putItem` menyimpan item dalam tabel. Jika item ada, item tersebut akan menggantikan keseluruhan item. Alih-alih mengganti seluruh item, jika Anda hanya ingin memperbaiki atribut tertentu, Anda dapat menggunakan metode `updateItem`. Untuk informasi selengkapnya, lihat [Memperbarui Item](#).

Ikuti langkah-langkah ini:

1. Buat instans kelas `DynamoDB`.
2. Buat instans kelas `Table` untuk mewakili tabel yang ingin Anda gunakan.
3. Buat instans kelas `Item` untuk mewakili item baru. Anda harus menentukan kunci primer item baru dan atributnya.
4. Panggil metode `putItem` dari objek `Table` dengan menggunakan `Item` yang Anda buat pada langkah sebelumnya.

Contoh kode Java berikut menunjukkan tugas sebelumnya. Kode tersebut menulis item baru ke tabel `ProductCatalog`.

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);
```

Pada contoh sebelumnya, item memiliki atribut berupa skalar (`String`, `Number`, `Boolean`, `Null`), set (`String Set`), dan jenis dokumen (`List`, `Map`).

Menentukan parameter opsional

Selain parameter yang diperlukan, Anda juga dapat menentukan parameter opsional untuk metode `putItem`. Misalnya, contoh kode Java berikut menggunakan parameter opsional untuk menentukan kondisi untuk mengunggah item. Jika kondisi yang Anda tentukan tidak terpenuhi, AWS SDK for Java akan meluncurkan `ConditionalCheckFailedException`. Contoh kode menentukan parameter opsional berikut dalam metode `putItem`:

- `ConditionExpression` yang mendefinisikan kondisi permintaan. Kode tersebut mendefinisikan kondisi bahwa item yang ada dengan kunci primer yang sama diganti hanya jika item tersebut memiliki atribut ISBN yang sama dengan nilai tertentu.
- Peta untuk `ExpressionAttributeValues` yang digunakan dalam kondisi tersebut. Dalam kasus ini, hanya ada satu substitusi yang diperlukan: placeholder `:val` dalam ekspresi kondisi diganti pada saat runtime dengan nilai ISBN aktual yang akan diperiksa.

Contoh berikut menambahkan item buku baru menggunakan parameter opsional.

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 dan dokumen JSON

Anda dapat menyimpan dokumen JSON sebagai atribut dalam tabel DynamoDB. Untuk melakukannya, gunakan metode `withJSON` Item. Metode ini mengurai dokumen JSON dan memetakan setiap elemen untuk tipe daya DynamoDB asli.

Misalkan Anda ingin menyimpan dokumen JSON berikut, yang berisi vendor yang dapat memenuhi pesanan untuk produk tertentu.

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"
  ]
}
}
```

Anda dapat menggunakan metode `withJSON` untuk menyimpan ini dalam tabel `ProductCatalog`, dalam atribut Map yang disebut `VendorInfo`. Contoh kode Java berikut mendemonstrasikan cara melakukannya.

```
// 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);
```

Mendapatkan item

Untuk mengambil satu item, gunakan metode `getItem` dari objek `Table`. Ikuti langkah-langkah ini:

1. Buat instans kelas `DynamoDB`.
2. Buat instans kelas `Table` untuk mewakili tabel yang ingin Anda gunakan.
3. Panggil metode `getItem` dari instans `Table`. Anda harus menentukan kunci item primer yang ingin Anda ambil.

Contoh kode Java berikut menunjukkan langkah sebelumnya. Kode mendapatkan item yang memiliki kunci partisi yang ditentukan.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Item item = table.getItem("Id", 210);
```

Menentukan parameter opsional

Selain parameter yang diperlukan, Anda juga dapat menentukan parameter opsional untuk metode `getItem`. Misalnya, contoh kode Java berikut menggunakan metode opsional untuk mengambil hanya daftar atribut tertentu dan menentukan bacaan sangat konsisten. (Untuk mempelajari selengkapnya tentang konsistensi baca, lihat [Konsistensi baca](#).)

Anda dapat menggunakan `ProjectionExpression` untuk mengambil hanya atribut atau elemen tertentu, bukan seluruh item. `ProjectionExpression` dapat menentukan tingkat atas atau atribut bersarang menggunakan jalur dokumen. Untuk informasi selengkapnya, lihat [Ekspresi proyeksi](#).

Parameter dari metode `getItem` tidak membiarkan Anda menentukan konsistensi baca. Namun, Anda dapat membuat `GetItemSpec`, yang menyediakan akses penuh ke semua input ke operasi `GetItem` tingkat rendah. Contoh kode berikut membuat `GetItemSpec` dan menggunakan spesifikasi tersebut sebagai input untuk metode `getItem`.

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());
```

Untuk mencetak Item dalam format yang dapat dibaca manusia, gunakan metode `toJSONPretty`. Output dari contoh sebelumnya terlihat seperti berikut ini.

```
{
  "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 dan dokumen JSON

Di bagian [PutItem dan dokumen JSON](#), Anda menyimpan dokumen JSON dalam atribut Map yang disebut `VendorInfo`. Anda dapat menggunakan metode `getItem` untuk mengambil seluruh dokumen dalam format JSON. Selain itu, Anda dapat menggunakan notasi jalur dokumen untuk mengambil hanya beberapa elemen dalam dokumen. Contoh kode Java berikut mendemonstrasikan teknik ini.

```
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());
```

Output dari contoh sebelumnya terlihat seperti berikut ini.

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

Anda dapat menggunakan metode `toJSON` untuk mengonversi item apa pun (atau atributnya) ke string berformat JSON. Kode berikut mengambil beberapa atribut tingkat atas dan bertingkat serta mencetak hasilnya sebagai JSON.

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210)
    .withProjectionExpression("VendorInfo.V01, Title, Price");
```

```
Item item = table.getItem(spec);
System.out.println(item.toJSON());
```

Output-nya akan terlihat seperti berikut.

```
{"VendorInfo":{"V01":{"Name":"Acme Books","Offices":
["Seattle"]}},"Price":30,"Title":"Book 210 Title"}
```

Penulisan batch: Menempatkan dan menghapus beberapa item

Penulisan batch merujuk pada menempatkan dan menghapus beberapa item dalam satu batch.

Metode `batchWriteItem` memungkinkan Anda untuk memasukkan dan menghapus beberapa item dari satu atau lebih tabel dalam satu panggilan. Berikut adalah langkah-langkah untuk menempatkan atau menghapus beberapa item menggunakan AWS SDK for Java Document API.

1. Buat instans kelas `DynamoDB`.
2. Buat instans dari kelas `TableWriteItems` yang menjelaskan semua operasi put dan hapus untuk tabel. Jika Anda ingin menulis ke beberapa tabel dalam satu operasi penulisan batch, Anda harus membuat satu instans `TableWriteItems` per tabel.
3. Jalankan metode `batchWriteItem` dengan menyediakan objek `TableWriteItems` yang Anda buat pada langkah sebelumnya.
4. Proses responsnya. Anda harus memeriksa apakah ada item permintaan yang belum diproses yang dikembalikan sebagai respons. Hal ini dapat terjadi jika Anda mencapai kuota throughput yang disediakan atau kesalahan sementara lainnya. Selain itu, `DynamoDB` membatasi ukuran permintaan dan jumlah operasi yang dapat Anda tentukan dalam permintaan. Jika Anda melebihi batas ini, `DynamoDB` menolak permintaan tersebut. Untuk informasi selengkapnya, lihat [Layanan, akun, dan tabel kuota di Amazon DynamoDB](#).

Contoh kode Java berikut menunjukkan langkah sebelumnya. Contoh tersebut melakukan operasi `batchWriteItem` pada dua tabel: `Forum` dan `Thread`. Objek `TableWriteItems` yang sesuai menentukan tindakan berikut:

- Tempatkan satu item dalam tabel `Forum`.
- Tempatkan dan hapus satu item dalam tabel `Thread`.

Kode kemudian menjalankan `batchWriteItem` untuk melakukan operasi.

```
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
```

Untuk contoh pekerjaan, lihat [Contoh: Operasi batch write menggunakan AWS SDK for Java document API](#).

Batch get: Mendapatkan beberapa item

Metode `batchGetItem` memungkinkan Anda mengambil beberapa item dari satu atau lebih tabel. Untuk mengambil satu item, Anda dapat menggunakan metode `getItem`.

Ikuti langkah-langkah ini:

1. Buat instans kelas `DynamoDB`.
2. Buat instans dari kelas `TableKeysAndAttributes` yang menjelaskan daftar nilai kunci primer untuk mengambil dari tabel. Jika Anda ingin membaca dari beberapa tabel dalam satu operasi penulisan batch, Anda harus membuat satu instans `TableKeysAndAttributes` per tabel.
3. Jalankan metode `batchGetItem` dengan menyediakan objek `TableKeysAndAttributes` yang Anda buat pada langkah sebelumnya.

Contoh kode Java berikut menunjukkan langkah sebelumnya. Contoh tersebut mengambil dua item dari tabel `Forum` dan tiga item dari tabel `Thread`.

```
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);
    }
}
```

Menentukan parameter opsional

Bersamaan dengan parameter yang diperlukan, Anda juga dapat menentukan parameter opsional saat menggunakan `batchGetItem`. Misalnya, Anda dapat memberikan `ProjectionExpression` dengan setiap `TableKeysAndAttributes` yang Anda tentukan. Hal ini memungkinkan Anda menentukan atribut yang ingin Anda ambil dari tabel.

Contoh kode berikut mengambil dua item dari tabel Forum. Parameter `withProjectionExpression` menentukan bahwa hanya atribut `Threads` yang akan diambil.

Example

```
TableKeysAndAttributes forumTableKeysAndAttributes = new
    TableKeysAndAttributes("Forum")
        .withProjectionExpression("Threads");

forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(forumTableKeysAndAttributes);
```

Memperbarui Item

Metode `updateItem` dari objek `Table` dapat memperbarui nilai atribut yang ada, menambahkan atribut baru, atau menghapus atribut dari item yang ada.

Metode `updateItem` berperilaku sebagai berikut:

- Jika item tidak ada (tidak ada item dalam tabel dengan kunci primer yang ditentukan), `updateItem` menambahkan item baru ke tabel.

- Jika item ada, `updateItem` melakukan pembaruan seperti yang ditentukan oleh parameter `UpdateExpression`.

Note

Hal ini juga memungkinkan untuk "memperbarui" item menggunakan `putItem`. Misalnya, jika Anda memanggil `putItem` untuk menambahkan item ke tabel, tetapi sudah ada item dengan kunci primer yang ditentukan, `putItem` akan menggantikan seluruh item. Jika ada atribut dalam item yang ada yang tidak ditentukan dalam input, `putItem` akan menghapus atribut tersebut dari item.

Secara umum, sebaiknya gunakan `updateItem` setiap kali Anda ingin memodifikasi atribut item. Metode `updateItem` hanya memodifikasi atribut item yang Anda tentukan dalam input, dan atribut lainnya dalam item tidak berubah.

Ikuti langkah-langkah ini:

1. Buat instans kelas `Table` untuk mewakili tabel yang ingin Anda gunakan.
2. Panggil metode `updateTable` dari instans `Table`. Anda harus menentukan kunci primer dari item yang ingin Anda ambil, bersama dengan `UpdateExpression` yang menjelaskan atribut untuk memodifikasi dan cara memodifikasinya.

Contoh kode Java berikut menunjukkan tugas sebelumnya. Kode tersebut memperbarui item buku dalam tabel `ProductCatalog`. Contoh tersebut menambahkan penulis baru ke set `Authors`, dan menghapus atribut `ISBN` yang ada. Contoh tersebut juga mengurangi harga sebanyak satu.

Peta `ExpressionAttributeValue` digunakan dalam `UpdateExpression`. Placeholder `:val1` dan `:val2` diganti pada saat runtime dengan nilai aktual untuk `Authors` dan `Price`.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeName = new HashMap<String, String>();
expressionAttributeName.put("#A", "Authors");
expressionAttributeName.put("#P", "Price");
expressionAttributeName.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);
```

Menentukan parameter opsional

Bersamaan dengan parameter yang diperlukan, Anda juga dapat menentukan parameter opsional untuk metode `updateItem`, termasuk kondisi yang harus dipenuhi agar pembaruan bisa terjadi. Jika kondisi yang Anda tentukan tidak terpenuhi, AWS SDK for Java akan meluncurkan `ConditionalCheckFailedException`. Misalnya, contoh kode Java berikut memperbarui harga item buku secara kondisional menjadi 25. Ini menentukan pernyataan `ConditionExpression` bahwa harga harus diperbarui hanya jika harga saat ini adalah 20.

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);
```

Penghitung atom

Anda dapat menggunakan `updateItem` untuk mengimplementasi penghitung atom, di mana Anda menambahkan atau mengurangi nilai atribut yang ada tanpa mengganggu permintaan tulis lainnya. Untuk kenaikan penghitung atom, gunakan `UpdateExpression` dengan tindakan `set` untuk menambahkan nilai numerik ke atribut jenis `Number` yang ada.

Contoh berikut menunjukkan hal ini, menambahkan satu atribut `Quantity`. Hal ini juga menunjukkan penggunaan parameter `ExpressionAttributeNames` dalam `UpdateExpression`.

```
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);
```

Menghapus item

Metode `deleteItem` menghapus item dari tabel. Anda harus menyediakan kunci item primer yang ingin Anda hapus.

Ikuti langkah-langkah ini:

1. Buat instans dari klien `DynamoDB`.
2. Panggil metode `deleteItem` dengan menyediakan kunci item yang ingin Anda hapus.

Contoh Java berikut menunjukkan tugas ini.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);
```

```
Table table = dynamoDB.getTable("ProductCatalog");

DeleteItemOutcome outcome = table.deleteItem("Id", 101);
```

Menentukan parameter opsional

Anda dapat menentukan parameter opsional untuk `deleteItem`. Misalnya, contoh kode Java berikut menentukan `ConditionExpression`, yang menyatakan bahwa item buku dalam `ProductCatalog` hanya dapat dihapus jika buku ini tidak lagi dalam publikasi (atribut `InPublication` adalah salah).

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);
```

Contoh: Operasi CRUD menggunakan AWS SDK for Java document API

Contoh kode berikut menunjukkan operasi CRUD pada item Amazon DynamoDB. Contohnya membuat item, mengambilnya, melakukan berbagai pembaruan, dan akhirnya menghapus item tersebut.

Note

SDK untuk Java juga menyediakan model persistensi objek, memungkinkan Anda memetakan kelas sisi klien ke tabel DynamoDB. Pendekatan ini dapat mengurangi jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat [Java 1.x: DynamoDBMapper](#).

Note

Contoh kode ini mengasumsikan bahwa Anda telah memuat data ke DynamoDB untuk akun Anda dengan mengikuti petunjuk di bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihat [Contoh kode Java](#).

```
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 DocumentAPIItemCRUExample {

    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());
    }
}
}
```

Contoh: Operasi batch menggunakan AWS SDK for Java document API

Bagian ini memberikan contoh operasi batch write dan batch get di Amazon DynamoDB menggunakan AWS SDK for Java Document API.

Note

SDK untuk Java juga menyediakan model persistensi objek, memungkinkan Anda memetakan kelas sisi klien ke tabel DynamoDB. Pendekatan ini dapat mengurangi jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat [Java 1.x: DynamoDBMapper](#).

Topik

- [Contoh: Operasi batch write menggunakan AWS SDK for Java document API](#)
- [Contoh: Operasi batch get menggunakan AWS SDK for Java document API](#)

Contoh: Operasi batch write menggunakan AWS SDK for Java document API

Contoh kode Java berikut menggunakan metode `batchWriteItem` untuk melakukan operasi put dan delete berikut:

- Tempatkan satu item dalam tabel `Forum`.
- Tempatkan satu item dan hapus satu item dari tabel `Thread`.

Anda dapat menentukan sejumlah permintaan put dan delete terhadap satu atau beberapa tabel saat membuat permintaan batch write Anda. Namun, `batchWriteItem` membatasi ukuran permintaan penulisan batch dan jumlah operasi put dan delete dalam satu operasi penulisan batch. Jika permintaan Anda melebihi batas ini, permintaan Anda ditolak. Jika tabel Anda tidak memiliki cukup throughput yang disediakan untuk melayani permintaan ini, item permintaan yang belum diproses akan dikembalikan sebagai respons.

Contoh berikut memeriksa respons untuk melihat apakah ada item permintaan yang belum diproses. Jika ya, contoh tersebut akan mengulang kembali dan mengirim ulang permintaan `batchWriteItem` dengan item yang belum diproses dalam permintaan tersebut. Jika Anda mengikuti bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#), semestinya Anda telah membuat tabel `Forum` dan `Thread`. Anda juga dapat membuat tabel ini dan mengunggah data sampel secara terprogram. Untuk informasi selengkapnya, lihat [Membuat tabel contoh dan mengunggah data menggunakan AWS SDK for Java](#).

Untuk step-by-step petunjuk pengujian sampel berikut, lihat [Contoh kode Java](#).

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);
    }

}

}
```

Contoh: Operasi batch get menggunakan AWS SDK for Java document API

Contoh kode Java berikut menggunakan metode `batchGetItem` untuk mengambil beberapa item dari tabel `Forum` dan `Thread`. `BatchGetItemRequest` menentukan nama tabel dan daftar kunci untuk setiap item yang akan didapatkan. Contoh tersebut memproses respons dengan mencetak item yang diambil.

Note

Contoh kode ini mengasumsikan bahwa Anda telah memuat data ke DynamoDB untuk akun Anda dengan mengikuti petunjuk di bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihat [Contoh kode Java](#).

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());
}

}

}
```

Contoh: Penanganan atribut jenis biner menggunakan AWS SDK for Java document API

Contoh kode Java berikut menggambarkan penanganan atribut jenis biner. Contoh tersebut menambahkan item ke tabel Reply. Item termasuk atribut jenis biner (ExtendedMessage) yang menyimpan data terkompresi. Contoh tersebut kemudian mengambil item dan mencetak semua nilai atribut. Sebagai ilustrasi, contoh menggunakan kelas GZIPOutputStream untuk mengompresi aliran sampel dan menetapkannya ke atribut ExtendedMessage. Ketika atribut biner diambil, atribut didekompresi menggunakan kelas GZIPInputStream.

Note

SDK untuk Java juga menyediakan model persistensi objek, memungkinkan Anda memetakan kelas sisi klien ke tabel DynamoDB. Pendekatan ini dapat mengurangi jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat [Java 1.x: DynamoDBMapper](#).

Jika Anda mengikuti bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#), semestinya Anda telah membuat tabel Reply. Anda juga dapat membuat tabel ini secara terprogram. Untuk informasi selengkapnya, lihat [Membuat tabel contoh dan mengunggah data menggunakan AWS SDK for Java](#).

Untuk step-by-step petunjuk pengujian sampel berikut, lihat [Contoh kode Java](#).

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;
}
}
```

Bekerja dengan item: .NET

Anda dapat menggunakan API tingkat rendah AWS SDK for .NET untuk melakukan operasi buat, baca, perbarui, dan hapus (CRUD) pada item dalam tabel. Berikut ini adalah langkah-langkah umum yang Anda ikuti untuk melakukan operasi CRUD data menggunakan API tingkat rendah .NET:

1. Buat instans dari kelas `AmazonDynamoDBClient` (klien).
2. Berikan parameter khusus operasi yang diperlukan dalam objek permintaan yang sesuai.

Misalnya, gunakan permintaan `PutItemRequest` saat mengunggah item dan gunakan permintaan `GetItemRequest` saat mengambil item yang sudah ada.

Anda dapat menggunakan objek permintaan untuk menyediakan parameter wajib dan opsional.

3. Jalankan metode yang sesuai yang disediakan oleh klien dengan meneruskan objek permintaan yang Anda buat pada langkah sebelumnya.

Klien `AmazonDynamoDBClient` menyediakan metode `PutItem`, `GetItem`, `UpdateItem`, dan `DeleteItem` untuk operasi CRUD.

Topik

- [Menempatkan item](#)
- [Mendapatkan item](#)
- [Memperbarui Item](#)
- [Penghitung atom](#)
- [Menghapus item](#)
- [Penulisan batch: Menempatkan dan menghapus beberapa item](#)
- [Batch get: Mendapatkan beberapa item](#)
- [Contoh: Operasi CRUD menggunakan API tingkat rendah AWS SDK for .NET](#)
- [Contoh: Operasi batch menggunakan API tingkat rendah AWS SDK for .NET](#)
- [Contoh: Penanganan atribut jenis biner menggunakan API tingkat rendah AWS SDK for .NET](#)

Menempatkan item

Metode `PutItem` mengunggah item ke tabel. Jika item ada, item tersebut akan menggantikan keseluruhan item.

Note

Alih-alih mengganti seluruh item, jika Anda hanya ingin memperbarui atribut tertentu, Anda dapat menggunakan metode `UpdateItem`. Untuk informasi selengkapnya, lihat [Memperbarui Item](#).

Berikut langkah-langkah untuk mengunggah item menggunakan .NET SDK API tingkat rendah:

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Berikan parameter yang diperlukan dengan membuat instans kelas `PutItemRequest`.

Untuk menaruh item, Anda harus memberikan nama tabel dan itemnya.

3. Jalankan metode `PutItem` dengan menyediakan objek `PutItemRequest` yang Anda buat pada langkah sebelumnya.

Contoh #C berikut menunjukkan langkah-langkah sebelumnya. Contoh tersebut mengunggah item ke tabel `ProductCatalog`.

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);
```

Pada contoh sebelumnya, Anda mengunggah item buku yang memiliki atribut `Id`, `Title`, `ISBN`, dan `Authors`. Perhatikan bahwa `Id` adalah atribut jenis numerik, dan semua atribut lainnya adalah jenis string. Penulis adalah set `String`.

Menentukan parameter opsional

Anda juga dapat memberikan parameter opsional menggunakan objek `PutItemRequest` seperti yang ditunjukkan dalam contoh C# berikut. Contoh tersebut menentukan parameter opsional berikut:

- `ExpressionAttributeNames`, `ExpressionAttributeValues`, dan `ConditionExpression` menentukan bahwa barang tersebut hanya dapat diganti jika barang yang ada memiliki atribut `ISBN` dengan nilai tertentu.
- Parameter `ReturnValues` untuk meminta item lama dalam respons.

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);
```

Untuk informasi lebih lanjut, lihat [PutItem](#).

Mendapatkan item

Metode `GetItem` mengambil item.

Note

Untuk mengambil beberapa item, Anda dapat menggunakan metode `BatchGetItem`. Untuk informasi selengkapnya, lihat [Batch get: Mendapatkan beberapa item](#).

Berikut ini adalah langkah-langkah untuk mengambil item yang ada menggunakan API AWS SDK for .NET tingkat rendah.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Berikan parameter yang diperlukan dengan membuat instans kelas `GetItemRequest`.

Untuk mendapatkan item, Anda harus memberikan nama tabel dan kunci primer item tersebut.

3. Jalankan metode `GetItem` dengan menyediakan objek `GetItemRequest` yang Anda buat pada langkah sebelumnya.

Contoh #C berikut menunjukkan langkah-langkah sebelumnya. Contoh tersebut mengambil item dari tabel `ProductCatalog`.

```
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.
```

Menentukan parameter opsional

Anda juga dapat memberikan parameter opsional menggunakan objek `GetItemRequest`, seperti yang ditunjukkan dalam contoh C# berikut. Contoh tersebut menentukan parameter opsional berikut:

- Parameter `ProjectionExpression` untuk menentukan atribut yang akan diambil.
- Parameter `ConsistentRead` untuk melakukan bacaan sangat konsisten. Untuk mempelajari selengkapnya tentang konsistensi baca, lihat [Konsistensi baca](#).

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;
```

Untuk informasi lebih lanjut, lihat [GetItem](#).

Memperbarui Item

Metode `UpdateItem` memperbarui item yang ada jika ada. Anda dapat menggunakan operasi `UpdateItem` untuk memperbarui nilai atribut yang ada, menambahkan atribut baru, atau menghapus atribut dari koleksi yang ada. Jika item yang memiliki kunci primer yang ditentukan tidak ditemukan, item baru akan ditambahkan.

Operasi `UpdateItem` menggunakan panduan berikut:

- Jika item tidak ada, `UpdateItem` menambahkan item baru menggunakan kunci primer yang ditentukan dalam input.
- Jika item ada, `UpdateItem` menerapkan pembaruan sebagai berikut:
 - Menggantikan nilai atribut yang ada dengan nilai dalam pembaruan.
 - Jika atribut yang Anda berikan pada input tidak ada, atribut baru akan ditambahkan ke item tersebut.
 - Jika atribut input adalah null, atribut tersebut akan dihapus, jika ada.
 - Jika Anda menggunakan `ADD` untuk `Action`, Anda dapat menambahkan nilai ke set yang sudah ada (set string atau angka), atau menambahkan secara matematis (menggunakan angka positif atau mengurangi (menggunakan angka negatif) dari nilai atribut numerik yang ada).

Note

Operasi `PutItem` juga dapat melakukan pembaruan. Untuk informasi selengkapnya, lihat [Menempatkan item](#). Misalnya, jika Anda memanggil `PutItem` untuk mengunggah item dan kunci primer ada, operasi `PutItem` menggantikan seluruh item. Jika terdapat atribut dalam item yang ada yang tidak ditentukan dalam input, operasi `PutItem` akan menghapus atribut

tersebut Namun, UpdateItem hanya memperbarui atribut input yang ditentukan. Atribut lain yang ada dari item tersebut tidak akan berubah.

Berikut langkah-langkah untuk memperbarui item menggunakan .NET SDK API tingkat rendah:

1. Buat instans kelas AmazonDynamoDBClient.
2. Berikan parameter yang diperlukan dengan membuat instans kelas UpdateItemRequest.

Ini adalah objek permintaan tempat Anda menjelaskan semua pembaruan, seperti menambahkan atribut, memperbarui atribut yang ada, atau menghapus atribut. Untuk menghapus atribut yang ada, tentukan nama atribut dengan nilai null.

3. Jalankan metode UpdateItem dengan menyediakan objek UpdateItemRequest yang Anda buat pada langkah sebelumnya.

Contoh kode #C berikut menunjukkan langkah-langkah sebelumnya. Contoh tersebut memperbarui item buku dalam tabel ProductCatalog. Contoh tersebut menambahkan penulis baru ke koleksi Authors, dan menghapus atribut ISBN yang ada. Contoh tersebut juga mengurangi harga sebanyak satu.

```
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);

```

Menentukan parameter opsional

Anda juga dapat memberikan parameter opsional menggunakan objek `UpdateItemRequest`, seperti yang ditunjukkan dalam contoh C# berikut. Contoh tersebut menentukan parameter opsional berikut:

- `ExpressionAttributeValues` dan `ConditionExpression` untuk menentukan bahwa harga dapat diperbarui hanya jika harga yang ada adalah 20,00.
- Parameter `ReturnValues` untuk meminta item yang diperbarui dalam respons.

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);
```

Untuk informasi lebih lanjut, lihat [UpdateItem](#).

Penghitung atom

Anda dapat menggunakan `updateItem` untuk mengimplementasi penghitung atom, di mana Anda menambahkan atau mengurangi nilai atribut yang ada tanpa mengganggu permintaan tulis lainnya. Untuk memperbarui penghitung atom, gunakan `updateItem` dengan atribut jenis `Number` dalam parameter `UpdateExpression`, dan `ADD` sebagai `Action`.

Contoh berikut menunjukkan hal ini, menambahkan satu atribut `Quantity`.

```
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);
```

Menghapus item

Metode `DeleteItem` menghapus item dari tabel.

Berikut ini adalah langkah-langkah untuk menghapus item menggunakan .NET SDK API tingkat rendah.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Berikan parameter yang diperlukan dengan membuat instans kelas `DeleteItemRequest`.

Untuk menghapus item, diperlukan nama tabel dan kunci primer item.

3. Jalankan metode `DeleteItem` dengan menyediakan objek `DeleteItemRequest` yang Anda buat pada langkah sebelumnya.

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);
```

Menentukan parameter opsional

Anda juga dapat memberikan parameter opsional menggunakan objek `DeleteItemRequest` seperti yang ditunjukkan dalam contoh kode C# berikut. Contoh tersebut menentukan parameter opsional berikut:

- `ExpressionAttributeValues` dan `ConditionExpression` untuk menentukan bahwa item buku dapat dihapus hanya jika tidak lagi dalam publikasi (nilai `InPublication` atribut salah).
- Parameter `ReturnValues` untuk meminta item yang dihapus dalam respons.

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);
```

Untuk informasi lebih lanjut, lihat [DeleteItem](#).

Penulisan batch: Menempatkan dan menghapus beberapa item

Penulisan batch merujuk pada menempatkan dan menghapus beberapa item dalam satu batch. Metode `BatchWriteItem` memungkinkan Anda untuk memasukkan dan menghapus beberapa item dari satu atau lebih tabel dalam satu panggilan. Berikut ini adalah langkah-langkah untuk mengambil beberapa item menggunakan .NET SDK API tingkat rendah.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Jelaskan semua operasi tempatkan dan hapus dengan membuat instans dari kelas `BatchWriteItemRequest`.
3. Jalankan metode `BatchWriteItem` dengan menyediakan objek `BatchWriteItemRequest` yang Anda buat pada langkah sebelumnya.
4. Proses responsnya. Anda harus memeriksa apakah ada item permintaan yang belum diproses yang dikembalikan sebagai respons. Hal ini dapat terjadi jika Anda mencapai kuota throughput yang disediakan atau kesalahan sementara lainnya. Selain itu, DynamoDB membatasi ukuran permintaan dan jumlah operasi yang dapat Anda tentukan dalam permintaan. Jika Anda melebihi batas ini, DynamoDB menolak permintaan tersebut. Untuk informasi lebih lanjut, lihat [BatchWriteItem](#).

Contoh kode #C berikut menunjukkan langkah-langkah sebelumnya. Contoh ini membuat `BatchWriteItemRequest` untuk melakukan operasi tulis berikut:

- Tempatkan satu item dalam tabel Forum.
- Tempatkan dan hapus satu item dari tabel Thread.

Kode menjalankan `BatchWriteItem` untuk melakukan operasi batch.

```
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);
```

Untuk contoh pekerjaan, lihat [Contoh: Operasi batch menggunakan API tingkat rendah AWS SDK for .NET](#).

Batch get: Mendapatkan beberapa item

Metode `BatchGetItem` memungkinkan Anda mengambil beberapa item dari satu atau lebih tabel.

Note

Untuk mengambil satu item, Anda dapat menggunakan metode `GetItem`.

Berikut ini adalah langkah-langkah untuk mengambil beberapa item menggunakan API AWS SDK for .NET tingkat rendah.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Berikan parameter yang diperlukan dengan membuat instans kelas `BatchGetItemRequest`.

Untuk mengambil beberapa item, nama tabel dan daftar nilai kunci primer diperlukan.

3. Jalankan metode `BatchGetItem` dengan menyediakan objek `BatchGetItemRequest` yang Anda buat pada langkah sebelumnya.
4. Proses responsnya. Anda harus memeriksa apakah ada kunci yang belum diproses, yang dapat terjadi jika Anda mencapai kuota throughput yang disediakan atau kesalahan sementara lainnya.

Contoh kode #C berikut menunjukkan langkah-langkah sebelumnya. Contoh tersebut mengambil item dari dua tabel, `Forum` dan `Thread`. Permintaan menentukan dua item dalam `Forum` dan tiga item dalam tabel `Thread`. Respons tersebut mencakup item dari kedua tabel. Kode tersebut menunjukkan bagaimana Anda dapat memproses respons.

```
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);
}
```

Menentukan parameter opsional

Anda juga dapat memberikan parameter opsional menggunakan objek `BatchGetItemRequest` seperti yang ditunjukkan dalam contoh kode C# berikut. Contoh tersebut mengambil dua item dari tabel Forum. Contoh tersebut menentukan parameter opsional berikut:

- Parameter `ProjectionExpression` untuk menentukan atribut yang akan diambil.

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);
```

Untuk informasi lebih lanjut, lihat [BatchGetItem](#).

Contoh: Operasi CRUD menggunakan API tingkat rendah AWS SDK for .NET

Contoh kode C# berikut menunjukkan operasi CRUD pada item Amazon DynamoDB. Contoh tersebut menambahkan item ke tabel ProductCatalog, mengambilnya, melakukan berbagai pembaruan, dan akhirnya menghapus item. Jika Anda mengikuti langkah-langkah dalam [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#), Anda telah membuat tabel ProductCatalog. Anda juga dapat membuat tabel sampel ini secara terprogram. Untuk informasi selengkapnya, lihat [Membuat tabel contoh dan mengunggah data menggunakan AWS SDK for .NET](#).

Untuk step-by-step instruksi untuk menguji sampel berikut, lihat [Contoh kode .NET](#).

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 {
                B00L = 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("*****");
}
}
}
}

```

Contoh: Operasi batch menggunakan API tingkat rendah AWS SDK for .NET

Topik

- [Contoh: Operasi batch tulis menggunakan API tingkat rendah AWS SDK for .NET](#)
- [Contoh: Operasi batch get menggunakan API tingkat rendah AWS SDK for .NET](#)

Bagian ini memberikan contoh operasi batch, batch write dan batch get, yang mendukung Amazon DynamoDB.

Contoh: Operasi batch tulis menggunakan API tingkat rendah AWS SDK for .NET

Contoh kode C# berikut menggunakan metode `BatchWriteItem` untuk melakukan operasi put dan delete berikut:

- Tempatkan satu item dalam tabel Forum.
- Tempatkan satu item dan hapus satu item dari tabel Thread.

Anda dapat menentukan sejumlah permintaan put dan delete terhadap satu atau beberapa tabel saat membuat permintaan batch write Anda. Namun, `BatchWriteItem` DynamoDB membatasi ukuran permintaan penulisan batch dan jumlah operasi put dan delete dalam satu operasi penulisan batch. Untuk informasi lebih lanjut, lihat [BatchWriteItem](#). Jika permintaan Anda melebihi batas ini, permintaan Anda ditolak. Jika tabel Anda tidak memiliki cukup throughput yang disediakan untuk melayani permintaan ini, item permintaan yang belum diproses akan dikembalikan sebagai respons.

Contoh berikut memeriksa respons untuk melihat apakah ada item permintaan yang belum diproses. Jika ya, contoh tersebut akan mengulang kembali dan mengirim ulang permintaan `BatchWriteItem` dengan item yang belum diproses dalam permintaan tersebut. Jika Anda mengikuti langkah-langkah dalam [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#), Anda telah membuat tabel Forum dan Thread. Anda juga dapat membuat tabel sampel ini dan mengunggah data sampel secara terprogram. Untuk informasi selengkapnya, lihat [Membuat tabel contoh dan mengunggah data menggunakan AWS SDK for .NET](#).

Untuk step-by-step instruksi untuk menguji sampel berikut, lihat [Contoh kode .NET](#).

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);
}
}
```

```
}
```

Contoh: Operasi batch get menggunakan API tingkat rendah AWS SDK for .NET

Contoh kode C# berikut menggunakan metode `BatchGetItem` untuk mengambil beberapa item dari tabel `Forum` dan `Thread` dalam Amazon DynamoDB. `BatchGetItemRequest` menentukan nama tabel dan daftar kunci primer untuk setiap tabel. Contoh tersebut memproses respons dengan mencetak item yang diambil.

Jika Anda mengikuti langkah-langkah dalam [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#), Anda telah membuat tabel dengan data sampel. Anda juga dapat membuat tabel sampel ini dan mengunggah data sampel secara terprogram. Untuk informasi selengkapnya, lihat [Membuat tabel contoh dan mengunggah data menggunakan AWS SDK for .NET](#).

Untuk step-by-step instruksi untuk menguji sampel berikut, lihat [Contoh kode .NET](#).

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("*****");
    }
}
}

```

Contoh: Penanganan atribut jenis biner menggunakan API tingkat rendah AWS SDK for .NET

Contoh kode C# berikut menggambarkan penanganan atribut jenis biner. Contoh tersebut menambahkan item ke tabel Reply. Item termasuk atribut jenis biner (ExtendedMessage) yang

menyimpan data terkompresi. Contoh tersebut kemudian mengambil item dan mencetak semua nilai atribut. Sebagai ilustrasi, contoh menggunakan kelas `GZipStream` untuk mengompresi aliran sampel dan menetakannya ke atribut `ExtendedMessage`, serta mendekompresinya saat mencetak nilai atribut.

Jika Anda mengikuti langkah-langkah dalam [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#), Anda telah membuat tabel `Reply`. Anda juga dapat membuat tabel sampel ini secara terprogram. Untuk informasi selengkapnya, lihat [Membuat tabel contoh dan mengunggah data menggunakan AWS SDK for .NET](#).

Untuk step-by-step instruksi untuk menguji contoh berikut, lihat [Contoh kode .NET](#).

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();
    }
}
}
}

```

Koleksi item - cara memodelkan one-to-many hubungan di DynamoDB

Di DynamoDB, koleksi item adalah sekelompok item yang berbagi nilai kunci partisi yang sama, yang berarti item tersebut terkait. Koleksi item adalah mekanisme utama untuk memodelkan one-to-many hubungan di DynamoDB. Koleksi item hanya bisa ada pada tabel atau indeks yang dikonfigurasi untuk menggunakan [kunci primer komposit](#).

Note

Koleksi item bisa ada di tabel dasar atau indeks sekunder. Untuk informasi selengkapnya secara spesifik tentang bagaimana kumpulan item berinteraksi dengan indeks, lihat [Koleksi item dalam Indeks Sekunder Lokal](#).

Perhatikan tabel berikut yang menunjukkan tiga pengguna berbeda dan inventaris dalam game mereka:

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
		88A41A9A62B11CC8C120861928765A3EA41DEB9EAFE261D90F619473B89A2D4	Active	14275516966

Untuk beberapa item di setiap koleksi, kunci urutan adalah gabungan informasi yang digunakan untuk mengelompokkan data, misalnya `inventory::armor`, `inventory::weapon`, atau `info`. Setiap koleksi item dapat memiliki kombinasi yang berbeda dari atribut ini sebagai kunci urutan. Pengguna `account1234` memiliki item `inventory::weapons`, sementara pengguna `account1387` tidak memilikinya (karena mereka belum menemukan apa pun). Pengguna `account1138` hanya menggunakan dua item untuk kunci urutan mereka (karena mereka belum memiliki inventaris) sedangkan pengguna lain menggunakan tiga item.

DynamoDB memungkinkan Anda secara selektif mengambil item dari koleksi item ini untuk melakukan hal berikut:

- Mengambil semua item dari pengguna tertentu

- Mengambil hanya satu item dari pengguna tertentu
- Mengambil semua item dari jenis tertentu milik pengguna tertentu

Mempercepat kueri dengan mengatur data Anda dengan koleksi item

Dalam contoh ini, masing-masing item dalam tiga kumpulan item ini mewakili pemain dan model data yang telah kita pilih, berdasarkan pola akses game dan pemain. Data apa yang dibutuhkan game? Kapan game membutuhkannya? Seberapa sering game membutuhkannya? Berapa biaya untuk melakukannya dengan cara ini? Keputusan pemodelan data ini dibuat berdasarkan jawaban atas pertanyaan-pertanyaan ini.

Dalam game ini, ada halaman berbeda yang disajikan kepada pemain untuk inventaris senjata dan halaman lain untuk baju besi. Saat pemain membuka inventarisnya, senjata ditampilkan terlebih dahulu karena kami ingin halaman tersebut dimuat dengan sangat cepat, sedangkan halaman inventaris berikutnya dapat dimuat setelahnya. Karena masing-masing jenis item ini bisa berukuran cukup besar seiring pemain memperoleh lebih banyak item dalam game, kami memutuskan bahwa setiap halaman inventaris akan menjadi itemnya sendiri dalam koleksi item pemain di database.

Bagian berikut membahas lebih lanjut tentang bagaimana Anda dapat berinteraksi dengan koleksi item melalui operasi `Query`.

Topik

- [Operasi kueri di DynamoDB](#)

Operasi kueri di DynamoDB

Anda dapat menggunakan operasi API `Query` di Amazon DynamoDB untuk menemukan item berdasarkan nilai kunci primer.

Anda harus memberikan nama atribut kunci partisi dan satu nilai untuk atribut tersebut. `Query` mengembalikan semua item dengan nilai kunci partisi tersebut. Secara opsional, Anda dapat memberikan atribut kunci urutan dan menggunakan operator perbandingan untuk menyaring hasil pencarian.

Untuk informasi selengkapnya tentang cara menggunakan `Query`, seperti sintaks permintaan, parameter respons, dan contoh tambahan, lihat [Kueri](#) di Referensi API Amazon DynamoDB.

Topik

- [Ekspresi kondisi kunci untuk operasi Query](#)
- [Ekspresi filter untuk operasi Kueri](#)
- [Pemberian nomor halaman hasil kueri tabel](#)
- [Aspek lain dari bekerja dengan operasi Kueri](#)
- [Membuat kueri tabel dan indeks: Java](#)
- [Membuat kueri tabel dan indeks: .NET](#)

Ekspresi kondisi kunci untuk operasi Query

Untuk menentukan kriteria pencarian, Anda menggunakan ekspresi kondisi kunci—string yang menentukan item yang akan dibaca dari tabel atau indeks.

Anda harus menentukan nama dan nilai kunci partisi sebagai syarat kesetaraan. Anda tidak dapat menggunakan atribut bukan kunci dalam ekspresi kondisi kunci.

Anda juga dapat memberikan kondisi kedua untuk kunci urutan (jika ada). Kondisi kunci urutan harus menggunakan salah satu operator perbandingan berikut:

- $a = b$ — true jika atribut a sama dengan nilai b
- $a < b$ — true jika a kurang dari b
- $a \leq b$ — true jika a kurang dari atau sama dengan b
- $a > b$ — true jika a lebih besar dari b
- $a \geq b$ — true jika a lebih besar atau sama dengan b
- a BETWEEN b AND c — true jika a lebih besar atau sama dengan b , dan lebih kecil atau sama dengan c .

Fungsi berikut ini juga didukung:

- `begins_with (a, substr)`— true jika nilai atribut a dimulai dengan substring tertentu.

Contoh berikut AWS Command Line Interface (AWS CLI) menunjukkan penggunaan ekspresi kondisi kunci. Ekspresi ini menggunakan placeholder (seperti `:name` dan `:sub`) dan bukan nilai aktual.

Untuk informasi selengkapnya, lihat [Nama atribut ekspresi di DynamoDB](#) dan [Nilai atribut ekspresi](#).

Example

Ajukan kueri tabel Thread untuk ForumName (kunci partisi) tertentu. Semua item dengan nilai ForumName tersebut dibaca oleh kueri karena kunci urutan (Subject) tidak termasuk dalam KeyConditionExpression.

```
aws dynamodb query \  
  --table-name Thread \  
  --key-condition-expression "ForumName = :name" \  
  --expression-attribute-values '{":name":{"S":"Amazon DynamoDB"}}'
```

Example

Ajukan kueri tabel Thread untuk ForumName kunci partisi), tetapi kali ini hanya mengembalikan item dengan Subject(kunci urutan) tertentu.

```
aws dynamodb query \  
  --table-name Thread \  
  --key-condition-expression "ForumName = :name and Subject = :sub" \  
  --expression-attribute-values file://values.json
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{  
  ":name":{"S":"Amazon DynamoDB"},  
  ":sub":{"S":"DynamoDB Thread 1"}  
}
```

Example

Kueri tabel Reply untuk Id tertentu (kunci partisi), tetapi hanya kembalikan item yang ReplyDateTime (kunci urutan) dimulai dengan karakter tertentu.

```
aws dynamodb query \  
  --table-name Reply \  
  --key-condition-expression "Id = :id and begins_with(ReplyDateTime, :dt)" \  
  --expression-attribute-values file://values.json
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
```

```
":id":{"S":"Amazon DynamoDB#DynamoDB Thread 1"},
":dt":{"S":"2015-09"}
}
```

Anda dapat menggunakan nama atribut apa pun dalam ekspresi kondisi kunci, asalkan karakter pertama adalah a-z atau A-Z dan karakter lainnya (mulai dari karakter kedua, jika ada) adalah a-z, A-Z, or 0-9. Selain itu, nama atribut tidak boleh berupa kata khusus DynamoDB. (Untuk daftar lengkapnya, lihat [Disimpan kata-kata di DynamoDB](#).) Jika nama atribut tidak memenuhi persyaratan ini, Anda harus menentukan nama atribut ekspresi sebagai placeholder. Untuk informasi selengkapnya, lihat [Nama atribut ekspresi di DynamoDB](#).

Untuk item dengan nilai kunci partisi tertentu, DynamoDB menyimpan item ini berdekatan, dalam urutan berdasarkan nilai kunci urutan. Dalam operasi Query, DynamoDB mengambil item dalam urutan yang diurutkan, lalu memproses item menggunakan KeyConditionExpression dan FilterExpression apa pun yang mungkin ada. Baru setelah itu hasil Query dikirim kembali ke klien.

Operasi Query selalu mengembalikan kumpulan hasil. Jika tidak ditemukan item yang cocok, set hasil kosong.

Hasil Query selalu diurutkan berdasarkan nilai kunci urutan. Jika tipe daya kunci urutan adalah Number, hasilnya dikembalikan dalam urutan numerik. Jika tidak, hasilnya akan dikembalikan dalam urutan UTF-8 byte. Secara default, urutannya menaik. Untuk membalikkan urutan, atur parameter ScanIndexForward ke false.

Operasi Query tunggal dapat mengambil data maksimum 1 MB. Batas ini berlaku sebelum FilterExpression atau ProjectionExpression diterapkan pada hasil. Jika LastEvaluatedKey ada dalam respons dan bukan null, Anda harus memberi nomor halaman pada kumpulan hasil (lihat [Pemberian nomor halaman hasil kueri tabel](#)).

Ekspresi filter untuk operasi Kueri

Jika Anda perlu menyempurnakan hasil Query lebih lanjut, Anda dapat memberikan ekspresi filter secara opsional. Ekspresi filter menentukan item mana dalam hasil Query yang harus dikembalikan kepada Anda. Semua hasil lainnya dibuang.

Filter ekspresi diterapkan setelah Query selesai, namun sebelum hasilnya dikembalikan. Oleh karena itu, Query menggunakan jumlah kapasitas baca yang sama, terlepas dari apakah ada ekspresi filter.

Operasi Query dapat mengambil data maksimal 1 MB. Batasan ini berlaku sebelum ekspresi filter dievaluasi.

Filter ekspresi tidak boleh berisi atribut kunci partisi atau kunci urutan. Anda perlu menentukan atribut tersebut dalam ekspresi kondisi kunci, bukan ekspresi filter.

Sintaks untuk ekspresi filter mirip dengan ekspresi kondisi kunci. Ekspresi filter dapat menggunakan perbandingan, fungsi, dan operator logika yang sama sebagai ekspresi kondisi utama. Selain itu, ekspresi filter dapat menggunakan operator tak sama dengan \neq (\neq), operator OR, operator CONTAINS, operator IN, operator BEGINS_WITH, operator BETWEEN, operator EXISTS, dan operator SIZE. Lihat informasi yang lebih lengkap di [Ekspresi kondisi kunci untuk operasi Query](#) dan [Sintaks untuk ekspresi filter dan kondisi](#).

Example

AWS CLI Contoh berikut query Thread tabel untuk tertentu ForumName (kunci partisi) dan Subject (sort key). Dari item yang ditemukan, hanya rangkaian diskusi terpopuler yang dikembalikan—dengan kata lain, hanya rangkaian diskusi yang memiliki jumlah Views lebih dari tertentu.

```
aws dynamodb query \  
  --table-name Thread \  
  --key-condition-expression "ForumName = :fn and Subject = :sub" \  
  --filter-expression "#v >= :num" \  
  --expression-attribute-names '{"#v": "Views"}' \  
  --expression-attribute-values file://values.json
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{  
  ":fn":{"S":"Amazon DynamoDB"},  
  ":sub":{"S":"DynamoDB Thread 1"},  
  ":num":{"N":"3"}  
}
```

Perhatikan bahwa Views adalah kata yang dicadangkan di DynamoDB (lihat [Disimpan kata-kata di DynamoDB](#)), sehingga contoh ini menggunakan `#v` sebagai placeholder. Untuk informasi selengkapnya, lihat [Nama atribut ekspresi di DynamoDB](#).

Note

Ekspresi filter menghapus item dari set hasil Query. Jika memungkinkan, hindari penggunaan Query saat Anda ingin mengambil item dalam jumlah besar namun juga harus membuang sebagian besar item tersebut.

Pemberian nomor halaman hasil kueri tabel

DynamoDB memberi nomor halaman hasil dari operasi Query. Dengan pemberian nomor halaman, hasil Query dibagi menjadi "halaman" data berukuran 1 MB (atau kurang). Aplikasi bisa memproses halaman pertama hasil, lalu halaman kedua, dan seterusnya.

Query tunggal hanya mengembalikan set hasil yang sesuai dalam batas ukuran 1 MB. Untuk menentukan apakah terdapat lebih banyak hasil, dan untuk mengambilkan satu halaman sekaligus, aplikasi harus melakukan hal berikut:

1. Periksa hasil Query tingkat rendah:
 - Jika hasilnya berisi elemen `LastEvaluatedKey` dan elemen tersebut non-null, lanjutkan ke langkah 2.
 - Jika tidak ada `LastEvaluatedKey` pada hasilnya, tidak ada lagi item yang dapat diambil.
2. Buat permintaan Query baru, dengan parameter yang sama seperti yang sebelumnya. Namun, kali ini, ambil nilai `LastEvaluatedKey` dari langkah 1 dan gunakan sebagai parameter `ExclusiveStartKey` dalam permintaan Query baru.
3. Jalankan permintaan Query baru.
4. Lanjutkan ke langkah 1.

Dengan kata lain, `LastEvaluatedKey` dari respons Query harus digunakan sebagai `ExclusiveStartKey` untuk permintaan Query berikutnya. Jika tidak ada elemen `LastEvaluatedKey` dalam respons Query, maka Anda telah mengambil halaman hasil akhir. Jika `LastEvaluatedKey` tidak kosong, bukan berarti ada lebih banyak data di set hasil. Satu-satunya cara untuk mengetahui kapan Anda telah mencapai akhir kumpulan hasil adalah ketika `LastEvaluatedKey` kosong.

Anda dapat menggunakan AWS CLI untuk melihat perilaku ini. AWS CLI Mengirimkan Query permintaan tingkat rendah ke DynamoDB berulang kali, `LastEvaluatedKey` hingga tidak ada lagi dalam hasil. Perhatikan AWS CLI contoh berikut yang mengambil judul film dari tahun tertentu.

```
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
```

Biasanya, AWS CLI menangani pagination secara otomatis. Namun, dalam contoh ini, AWS CLI `--page-size` parameter membatasi jumlah item per halaman. Parameter `--debug` mencetak informasi tingkat rendah tentang permintaan dan respons.

Jika Anda menjalankan contoh ini, respons pertama dari DynamoDB terlihat serupa dengan yang berikut ini.

```
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}'
```

`LastEvaluatedKey` dalam responsnya menunjukkan bahwa tidak semua item telah diambil. AWS CLI kemudian mengeluarkan Query permintaan lain ke DynamoDB. Pola permintaan dan respons ini berlanjut hingga respons akhir.

```
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}'
```

Tidak adanya `LastEvaluatedKey` menunjukkan bahwa tidak ada lagi item yang dapat diambil.

Note

AWS SDK menangani respons DynamoDB tingkat rendah (termasuk ada atau tidak adanya `LastEvaluatedKey`) dan menyediakan berbagai abstraksi untuk hasil paginasi. Query Misalnya, SDK untuk antarmuka dokumen SDK for Java menyediakan dukungan `java.util.Iterator` sehingga Anda dapat mempelajari hasilnya satu per satu.

Untuk contoh kode dalam berbagai bahasa pemrograman, lihat [Panduan Memulai Amazon DynamoDB](#) dan dokumentasi AWS SDK untuk bahasa Anda.

Anda juga dapat mengurangi ukuran halaman dengan membatasi jumlah item dalam set hasil, dengan parameter `Limit` operasi `Query`.

Untuk informasi selengkapnya tentang pembuatan kueri dengan DynamoDB, lihat [Operasi kueri di DynamoDB](#).

Aspek lain dari bekerja dengan operasi Kueri

Membatasi jumlah item dalam set hasil

Dengan operasi `Query`, Anda dapat membatasi jumlah item yang dibaca. Untuk melakukannya, tetapkan parameter `Limit` ke jumlah maksimum item yang Anda inginkan.

Misalnya, anggaplah bahwa Anda `Query` tabel, dengan nilai `Limit` dari 6, dan tanpa ekspresi filter. Hasil `Query` berisi enam item pertama dari tabel yang cocok dengan ekspresi kondisi kunci dari permintaan.

Sekarang anggap bahwa Anda menambahkan ekspresi filter ke `Query`. Dalam kasus ini, DynamoDB membaca hingga enam item, lalu hanya mengembalikan item yang cocok dengan ekspresi filter. Hasil `Query` akhir berisi enam item atau lebih sedikit, meskipun lebih banyak item akan cocok dengan ekspresi filter jika DynamoDB terus membaca lebih banyak item.

Menghitung item dalam hasil

Selain item yang sesuai dengan kriteria Anda, respons `Query` berisi elemen berikut:

- `ScannedCount` — Jumlah item yang cocok dengan ekspresi kondisi kunci sebelum ekspresi filter (jika ada) diterapkan.
- `Count` — Jumlah item yang tersisa setelah ekspresi filter (jika ada) diterapkan.

Note

Jika Anda tidak menggunakan ekspresi filter, `ScannedCount` dan `Count` memiliki nilai yang sama.

Jika ukuran set hasil Query lebih besar dari 1 MB, ScannedCount dan Count mewakili hanya jumlah parsial dari total item. Anda harus melakukan beberapa operasi Query untuk mengambil semua hasil (lihat [Pemberian nomor halaman hasil kueri tabel](#)).

Setiap respons Query berisi ScannedCount dan Count untuk item yang diproses oleh permintaan Query khusus. Untuk mendapatkan total keseluruhan untuk semua permintaan Query, Anda dapat menyimpan penghitungan ScannedCount dan Count.

Unit kapasitas yang digunakan oleh kueri

Anda dapat melakukan Query tabel atau indeks sekunder apa pun, selama Anda memberikan nama atribut kunci partisi dan satu nilai untuk atribut tersebut. Query mengembalikan semua item dengan nilai kunci partisi tersebut. Secara opsional, Anda dapat memberikan atribut kunci urutan dan menggunakan operator perbandingan untuk menyaring hasil pencarian. Query Operasi API menggunakan unit kapasitas baca, sebagai berikut

Jika Anda Query...	DynamoDB menggunakan unit kapasitas baca dari...
Tabel	Kapasitas baca yang ditetapkan tabel.
Indeks sekunder global	Kapasitas baca yang ditetapkan indeks.
Indeks sekunder lokal	Kapasitas baca yang ditetapkan tabel dasar.

Secara default, operasi Query tidak menampilkan data tentang berapa banyak kapasitas baca yang digunakan. Namun, Anda dapat menentukan parameter ReturnConsumedCapacity dalam sebuah permintaan Query untuk mendapatkan informasi ini. Berikut ini adalah pengaturan yang valid untuk ReturnConsumedCapacity:

- NONE — Tidak ada penggunaan data kapasitas yang ditampilkan. (Ini menjadi opsi default.)
- TOTAL — Respons mencakup jumlah agregat unit kapasitas baca yang digunakan.
- INDEXES — Respons menunjukkan jumlah agregat unit kapasitas baca yang digunakan, beserta dengan kapasitas yang digunakan untuk setiap tabel dan indeks yang diakses.

DynamoDB menghitung jumlah unit kapasitas baca yang dikonsumsi berdasarkan jumlah item dan ukuran item tersebut, bukan pada jumlah data yang dikembalikan ke aplikasi. Dengan alasan ini,

jumlah unit kapasitas yang digunakan adalah sama, entah Anda meminta semua atribut (perilaku default) atau hanya beberapa dari atribut tersebut (menggunakan ekspresi proyeksi). Jumlahnya juga sama apakah Anda menggunakan ekspresi filter atau tidak. Query mengonsumsi unit kapasitas baca minimum untuk melakukan satu pembacaan yang sangat konsisten per detik, atau dua pembacaan yang konsisten per detik untuk item hingga 4 KB. Jika Anda perlu membaca item yang lebih besar dari 4 KB, DynamoDB memerlukan unit permintaan baca tambahan. Tabel kosong dan tabel yang sangat besar yang memiliki jumlah kunci partisi yang jarang mungkin melihat beberapa RCU tambahan yang dibebankan melebihi jumlah data yang ditanyakan. Ini mencakup biaya melayani Query permintaan, bahkan jika tidak ada data.

Konsistensi baca untuk kueri

Operasi Query melakukan bacaan akhir konsisten, secara default. Ini berarti bahwa hasil Query mungkin tidak mencerminkan perubahan karena operasi PutItem atau UpdateItem baru saja selesai. Untuk informasi selengkapnya, lihat [Konsistensi baca](#).

Jika Anda memerlukan bacaan sangat konsisten, atur parameter ConsistentRead ke true dalam permintaan Query.

Membuat kueri tabel dan indeks: Java

Operasi Query memungkinkan Anda membuat kueri tabel atau indeks sekunder di Amazon DynamoDB. Anda harus memberikan nilai kunci partisi dan syarat kesetaraan. Jika tabel atau indeks memiliki kunci urutan, Anda dapat memperbaiki hasil dengan memberikan nilai kunci urutan dan syarat.

Note

Ini AWS SDK for Java juga menyediakan model persistensi objek, memungkinkan Anda untuk memetakan kelas sisi klien Anda ke tabel DynamoDB. Pendekatan ini dapat mengurangi jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat [Java 1.x: DynamoDBMapper](#).

Berikut ini adalah langkah-langkah untuk mengambil item menggunakan AWS SDK for Java Document API.

1. Buat instans kelas `DynamoDB`.
2. Buat instans kelas `Table` untuk mewakili tabel yang ingin Anda gunakan.

3. Panggil metode `query` dari instans `Table`. Anda harus menentukan nilai kunci partisi dari item yang ingin Anda ambil, beserta dengan parameter kueri opsional.

Respons mencakup objek `ItemCollection` yang menyediakan semua item yang dikembalikan oleh kueri.

Contoh kode Java berikut menunjukkan tugas sebelumnya. Contoh tersebut menganggap bahwa Anda memiliki tabel `Reply` yang menyimpan balasan untuk thread forum. Untuk informasi selengkapnya, lihat [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

```
Reply ( Id, ReplyDateTime, ... )
```

Setiap thread forum memiliki ID unik dan dapat memiliki nol balasan atau lebih. Oleh karena itu, atribut `Id` dari tabel `Reply` terdiri dari nama forum dan subjek forum. `Id` (kunci partisi) dan `ReplyDateTime` (kunci urutan) menyusun kunci primer komposit untuk tabel.

Kueri berikut mengambil semua balasan untuk subjek thread tertentu. Kueri membutuhkan nama tabel dan nilai `Subject`.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
    .withRegion(Regions.US_WEST_2).build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("Reply");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Id = :v_id")
    .withValueMap(new ValueMap()
        .withString(":v_id", "Amazon DynamoDB#DynamoDB Thread 1"));

ItemCollection<QueryOutcome> items = table.query(spec);

Iterator<Item> iterator = items.iterator();
Item item = null;
while (iterator.hasNext()) {
    item = iterator.next();
    System.out.println(item.toJSONPretty());
}
```

Menentukan parameter opsional

Metode query mendukung beberapa parameter opsional. Misalnya, Anda dapat secara opsional mempersempit hasil dari kueri sebelumnya untuk mengembalikan balasan dalam dua minggu terakhir dengan menentukan kondisi. Kondisi ini disebut kondisi kunci urutan, karena DynamoDB mengevaluasi kondisi kueri yang Anda tentukan terhadap kunci urutan dari kunci primer. Anda dapat menentukan parameter opsional lainnya untuk hanya mengambil daftar atribut tertentu dari item dalam hasil kueri.

Contoh kode Java berikut mengambil balasan thread forum yang diposting dalam 15 hari terakhir. Contoh ini menentukan parameter opsional menggunakan yang berikut ini:

- `KeyConditionExpression` untuk mengambil balasan dari forum diskusi tertentu (kunci partisi) dan, dalam kumpulan item tersebut, balasan yang diposting dalam 15 hari terakhir (kunci urutan).
- `FilterExpression` untuk hanya mengembalikan balasan dari pengguna tertentu. Filter diterapkan setelah kueri diproses, namun sebelum hasilnya dikembalikan ke pengguna.
- `ValueMap` untuk menentukan nilai sebenarnya untuk placeholder `KeyConditionExpression`.
- Pengaturan `ConsistentRead` dengan nilai `true`, untuk meminta bacaan sangat konsisten.

Contoh ini menggunakan objek `QuerySpec` yang memberikan akses ke semua parameter input Query tingkat rendah.

Example

```
Table table = dynamoDB.getTable("Reply");

long twoWeeksAgoMilli = (new Date()).getTime() - (15L*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);

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Id = :v_id and ReplyDateTime > :v_reply_dt_tm")
    .withFilterExpression("PostedBy = :v_posted_by")
    .withValueMap(new ValueMap()
        .withString(":v_id", "Amazon DynamoDB#DynamoDB Thread 1")
        .withString(":v_reply_dt_tm", twoWeeksAgoStr)
        .withString(":v_posted_by", "User B"))
```

```
.withConsistentRead(true);

ItemCollection<QueryOutcome> items = table.query(spec);

Iterator<Item> iterator = items.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next().toJSONPretty());
}
```

Anda juga dapat membatasi jumlah item per halaman secara opsional dengan menggunakan metode `withMaxPageSize`. Saat Anda memanggil metode `query`, Anda mendapatkan `ItemCollection` yang berisi item yang dihasilkan. Anda kemudian dapat menelusuri hasilnya, memproses satu halaman sekaligus, hingga tidak ada lagi halaman yang tersisa.

Contoh kode Java berikut mengubah spesifikasi kueri yang ditampilkan sebelumnya. Kali ini, spesifikasi kueri menggunakan metode `withMaxPageSize`. Kelas `Page` menyediakan iterator yang memungkinkan kode memproses item di setiap halaman.

Example

```
spec.withMaxPageSize(10);

ItemCollection<QueryOutcome> items = table.query(spec);

// Process each page of results
int pageNum = 0;
for (Page<Item, QueryOutcome> page : items.pages()) {

    System.out.println("\nPage: " + ++pageNum);

    // Process each item on the current page
    Iterator<Item> item = page.iterator();
    while (item.hasNext()) {
        System.out.println(item.next().toJSONPretty());
    }
}
```

Contoh - kueri menggunakan Java

Tabel berikut menyimpan informasi tentang koleksi forum. Untuk informasi selengkapnya, lihat [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Note

SDK untuk Java juga menyediakan model persistensi objek, memungkinkan Anda memetakan kelas sisi klien ke tabel DynamoDB. Pendekatan ini dapat mengurangi jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat [Java 1.x: DynamoDBMapper](#).

Example

```
Forum ( Name, ... )
Thread ( ForumName, Subject, Message, LastPostedBy, LastPostDateTime, ... )
Reply ( Id, ReplyDateTime, Message, PostedBy, ... )
```

Dalam contoh kode Java ini, Anda menjalankan variasi dalam menemukan balasan untuk thread “DynamoDB Thread 1” di forum “DynamoDB”.

- Temukan balasan untuk thread.
- Temukan balasan untuk thread, menentukan batas jumlah item per halaman hasil. Jika jumlah item dalam set hasil melebihi ukuran halaman, Anda hanya mendapatkan halaman pertama dari hasil. Pola pengodean ini memastikan bahwa kode Anda memproses semua halaman dalam hasil kueri.
- Temukan balasan dalam 15 hari terakhir.
- Temukan balasan dalam rentang tanggal tertentu.

Dua kueri sebelumnya menunjukkan bagaimana Anda dapat menentukan syarat kunci urutan untuk mempersempit hasil kueri serta menggunakan parameter kueri opsional lainnya.

Note

Contoh kode ini mengasumsikan bahwa Anda telah memuat data ke DynamoDB untuk akun Anda dengan mengikuti petunjuk di bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihat [Contoh kode Java](#).

```
package com.amazonaws.codesamples.document;

import java.text.SimpleDateFormat;
import java.util.Date;
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.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.Page;
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;

public class DocumentAPIQuery {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "Reply";

    public static void main(String[] args) throws Exception {

        String forumName = "Amazon DynamoDB";
        String threadSubject = "DynamoDB Thread 1";

        findRepliesForAThread(forumName, threadSubject);
        findRepliesForAThreadSpecifyOptionalLimit(forumName, threadSubject);
        findRepliesInLast15DaysWithConfig(forumName, threadSubject);
        findRepliesPostedWithinTimePeriod(forumName, threadSubject);
        findRepliesUsingAFilterExpression(forumName, threadSubject);
    }

    private static void findRepliesForAThread(String forumName, String threadSubject) {

        Table table = dynamoDB.getTable(tableName);

        String replyId = forumName + "#" + threadSubject;

        QuerySpec spec = new QuerySpec().withKeyConditionExpression("Id = :v_id")
            .withValueMap(new ValueMap().withString(":v_id", replyId));
```

```
    ItemCollection<QueryOutcome> items = table.query(spec);

    System.out.println("\nfindRepliesForAThread results:");

    Iterator<Item> iterator = items.iterator();
    while (iterator.hasNext()) {
        System.out.println(iterator.next().toJSONPretty());
    }
}

private static void findRepliesForAThreadSpecifyOptionalLimit(String forumName,
String threadSubject) {

    Table table = dynamoDB.getTable(tableName);

    String replyId = forumName + "#" + threadSubject;

    QuerySpec spec = new QuerySpec().withKeyConditionExpression("Id = :v_id")
        .withValueMap(new ValueMap().withString(":v_id",
replyId)).withMaxPageSize(1);

    ItemCollection<QueryOutcome> items = table.query(spec);

    System.out.println("\nfindRepliesForAThreadSpecifyOptionalLimit results:");

    // Process each page of results
    int pageNum = 0;
    for (Page<Item, QueryOutcome> page : items.pages()) {

        System.out.println("\nPage: " + ++pageNum);

        // Process each item on the current page
        Iterator<Item> item = page.iterator();
        while (item.hasNext()) {
            System.out.println(item.next().toJSONPretty());
        }
    }
}

private static void findRepliesInLast15DaysWithConfig(String forumName, String
threadSubject) {
```

```
Table table = dynamoDB.getTable(tableName);

long twoWeeksAgoMilli = (new Date()).getTime() - (15L * 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);

String replyId = forumName + "#" + threadSubject;

QuerySpec spec = new QuerySpec().withProjectionExpression("Message,
ReplyDateTime, PostedBy")
    .withKeyConditionExpression("Id = :v_id and ReplyDateTime
<= :v_reply_dt_tm")
    .withValueMap(new ValueMap().withString(":v_id",
replyId).withString(":v_reply_dt_tm", twoWeeksAgoStr));

ItemCollection<QueryOutcome> items = table.query(spec);

System.out.println("\nfindRepliesInLast15DaysWithConfig results:");
Iterator<Item> iterator = items.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next().toJSONPretty());
}

}

private static void findRepliesPostedWithinTimePeriod(String forumName, String
threadSubject) {

    Table table = dynamoDB.getTable(tableName);

    long startDateMilli = (new Date()).getTime() - (15L * 24L * 60L * 60L * 1000L);
    long endDateMilli = (new Date()).getTime() - (5L * 24L * 60L * 60L * 1000L);
    java.text.SimpleDateFormat df = new java.text.SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");
    String startDate = df.format(startDateMilli);
    String endDate = df.format(endDateMilli);

    String replyId = forumName + "#" + threadSubject;

    QuerySpec spec = new QuerySpec().withProjectionExpression("Message,
ReplyDateTime, PostedBy")
```

```
        .withKeyConditionExpression("Id = :v_id and ReplyDateTime
between :v_start_dt and :v_end_dt")
        .withValueMap(new ValueMap().withString(":v_id",
replyId).withString(":v_start_dt", startDate)
        .withString(":v_end_dt", endDate));

    ItemCollection<QueryOutcome> items = table.query(spec);

    System.out.println("\nfindRepliesPostedWithinTimePeriod results:");
    Iterator<Item> iterator = items.iterator();
    while (iterator.hasNext()) {
        System.out.println(iterator.next().toJSONPretty());
    }
}

private static void findRepliesUsingAFilterExpression(String forumName, String
threadSubject) {

    Table table = dynamoDB.getTable(tableName);

    String replyId = forumName + "#" + threadSubject;

    QuerySpec spec = new QuerySpec().withProjectionExpression("Message,
ReplyDateTime, PostedBy")
        .withKeyConditionExpression("Id
= :v_id").withFilterExpression("PostedBy = :v_postedby")
        .withValueMap(new ValueMap().withString(":v_id",
replyId).withString(":v_postedby", "User B"));

    ItemCollection<QueryOutcome> items = table.query(spec);

    System.out.println("\nfindRepliesUsingAFilterExpression results:");
    Iterator<Item> iterator = items.iterator();
    while (iterator.hasNext()) {
        System.out.println(iterator.next().toJSONPretty());
    }
}
}
```


Membuat kueri tabel dan indeks: .NET

Operasi Query memungkinkan Anda membuat kueri tabel atau indeks sekunder di Amazon DynamoDB. Anda harus memberikan nilai kunci partisi dan syarat kesetaraan. Jika tabel atau indeks memiliki kunci urutan, Anda dapat memperbaiki hasil dengan memberikan nilai kunci urutan dan syarat.

Berikut ini adalah langkah-langkah untuk menanyakan tabel menggunakan AWS SDK for .NET API tingkat rendah.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Buat instans dari kelas `QueryRequest` dan berikan parameter operasi kueri.
3. Jalankan metode `Query` dan sediakan objek `QueryRequest` yang Anda buat pada langkah sebelumnya.

Respons mencakup objek `QueryResult` yang menyediakan semua item yang dikembalikan oleh kueri.

Contoh kode #C berikut mendemonstrasikan tugas sebelumnya. Kode tersebut menganggap bahwa Anda memiliki tabel `Reply` yang menyimpan balasan untuk thread forum. Untuk informasi selengkapnya, lihat [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Example

```
Reply Id, ReplyDateTime, ... )
```

Setiap thread forum memiliki ID unik dan dapat memiliki nol balasan atau lebih. Oleh karena itu, kunci primer terdiri dari baik `Id` (kunci partisi) dan `ReplyDateTime` (kunci urutan).

Kueri berikut mengambil semua balasan untuk subjek thread tertentu. Kueri membutuhkan nama tabel dan nilai `Subject`.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

var request = new QueryRequest
{
    TableName = "Reply",
    KeyConditionExpression = "Id = :v_Id",
```

```
ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
    {":v_Id", new AttributeValue { S = "Amazon DynamoDB#DynamoDB Thread 1" }}}
};

var response = client.Query(request);

foreach (Dictionary<string, AttributeValue> item in response.Items)
{
    // Process the result.
    PrintItem(item);
}
```

Menentukan parameter opsional

Metode `Query` mendukung beberapa parameter opsional. Misalnya, Anda dapat secara opsional mempersempit hasil kueri pada kueri sebelumnya untuk mengembalikan balasan dalam dua minggu terakhir dengan menentukan kondisi. Kondisi ini disebut kondisi kunci urutan, karena DynamoDB mengevaluasi kondisi kueri yang Anda tentukan terhadap kunci urutan dari kunci primer. Anda dapat menentukan parameter opsional lainnya untuk hanya mengambil daftar atribut tertentu dari item dalam hasil kueri. Untuk informasi selengkapnya, lihat [Kueri](#).

Contoh kode C# berikut mengambil balasan thread forum yang diposting dalam 15 hari terakhir. Contoh tersebut menentukan parameter opsional berikut:

- `KeyConditionExpression` untuk mengambil hanya balasan dalam 15 hari terakhir.
- Parameter `ProjectionExpression` untuk menentukan daftar atribut guna mengambil item dalam hasil kueri.
- Parameter `ConsistentRead` untuk melakukan bacaan sangat konsisten.

Example

```
DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
string twoWeeksAgoString = twoWeeksAgoDate.ToString(AWSSDKUtils.ISO8601DateFormat);

var request = new QueryRequest
{
    TableName = "Reply",
    KeyConditionExpression = "Id = :v_Id and ReplyDateTime > :v_twoWeeksAgo",
    ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
        {":v_Id", new AttributeValue { S = "Amazon DynamoDB#DynamoDB Thread 2" }},
```

```
        {":v_twoWeeksAgo", new AttributeValue { S = twoWeeksAgoString }}
    },
    ProjectionExpression = "Subject, ReplyDateTime, PostedBy",
    ConsistentRead = true
};

var response = client.Query(request);

foreach (Dictionary<string, AttributeValue> item in response.Items)
{
    // Process the result.
    PrintItem(item);
}
```

Selain itu, Anda juga dapat membatasi ukuran halaman, atau jumlah item per halaman, dengan menambahkan parameter `Limit` opsional. Setiap kali Anda menjalankan metode `Query`, Anda mendapatkan satu halaman hasil yang memiliki jumlah item tertentu. Untuk mengambil halaman berikutnya, jalankan kembali metode `Query` dengan memberikan nilai kunci primer item terakhir di halaman sebelumnya sehingga metode tersebut dapat mengembalikan kumpulan item berikutnya. Anda memberikan informasi ini dalam permintaan dengan menetapkan properti `ExclusiveStartKey`. Awalnya, properti ini bisa bernilai null. Untuk mengambil halaman berikutnya, Anda harus memperbarui nilai properti ini ke kunci primer item terakhir di halaman sebelumnya.

Contoh C# berikut menanyakan tabel `Reply`. Dalam permintaan, ini menentukan parameter opsional `Limit` dan `ExclusiveStartKey`. Perulangan `do/while` terus memindai satu halaman sekaligus hingga `LastEvaluatedKey` menampilkan nilai null.

Example

```
Dictionary<string, AttributeValue> lastKeyEvaluated = null;

do
{
    var request = new QueryRequest
    {
        TableName = "Reply",
        KeyConditionExpression = "Id = :v_Id",
        ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
            {":v_Id", new AttributeValue { S = "Amazon DynamoDB#DynamoDB Thread 2" }}
        },
```

```
// Optional parameters.
Limit = 1,
ExclusiveStartKey = lastKeyEvaluated
};

var response = client.Query(request);

// Process the query result.
foreach (Dictionary<string, AttributeValue> item in response.Items)
{
    PrintItem(item);
}

lastKeyEvaluated = response.LastEvaluatedKey;
} while (lastKeyEvaluated != null && lastKeyEvaluated.Count != 0);
```

Contoh - query menggunakan AWS SDK for .NET

Tabel berikut menyimpan informasi tentang koleksi forum. Untuk informasi selengkapnya, lihat [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Example

```
Forum ( Name, ... )
Thread ( ForumName, Subject, Message, LastPostedBy, LastPostDateTime, ... )
Reply ( Id, ReplyDateTime, Message, PostedBy, ... )
```

Dalam contoh ini, Anda menjalankan variasi “Temukan balasan untuk thread “DynamoDB Thread 1” di forum “DynamoDB”.

- Temukan balasan untuk thread.
- Temukan balasan untuk thread. Tentukan parameter kueri `Limit` untuk mengatur ukuran halaman.

Fungsi ini menggambarkan penggunaan pemberian nomor halaman untuk memproses hasil beberapa halaman. DynamoDB memiliki batas ukuran halaman dan jika hasil Anda melebihi ukuran halaman tersebut, Anda hanya mendapatkan halaman pertama dari hasil. Pola pengodean ini memastikan bahwa kode Anda memproses semua halaman dalam hasil kueri.

- Temukan balasan dalam 15 hari terakhir.
- Temukan balasan dalam rentang tanggal tertentu.

Dua kueri sebelumnya menunjukkan bagaimana Anda dapat menentukan kondisi kunci urutan untuk mempersempit hasil kueri dan menggunakan parameter kueri opsional lainnya.

Untuk step-by-step instruksi untuk menguji contoh berikut, lihat [Contoh kode .NET](#).

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.Util;

namespace com.amazonaws.codesamples
{
    class LowLevelQuery
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                // Query a specific forum and thread.
                string forumName = "Amazon DynamoDB";
                string threadSubject = "DynamoDB Thread 1";

                FindRepliesForAThread(forumName, threadSubject);
                FindRepliesForAThreadSpecifyOptionalLimit(forumName, threadSubject);
                FindRepliesInLast15DaysWithConfig(forumName, threadSubject);
                FindRepliesPostedWithinTimePeriod(forumName, threadSubject);

                Console.WriteLine("Example complete. To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message);
            Console.ReadLine(); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message);
            Console.ReadLine(); }
            catch (Exception e) { Console.WriteLine(e.Message); Console.ReadLine(); }
        }
    }
}
```

```
private static void FindRepliesPostedWithinTimePeriod(string forumName, string
threadSubject)
{
    Console.WriteLine("*** Executing FindRepliesPostedWithinTimePeriod() ***");
    string replyId = forumName + "#" + threadSubject;
    // You must provide date value based on your test data.
    DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(21);
    string start = startDate.ToString(AWSSDKUtils.ISO8601DateFormat);

    // You provide date value based on your test data.
    DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(5);
    string end = endDate.ToString(AWSSDKUtils.ISO8601DateFormat);

    var request = new QueryRequest
    {
        TableName = "Reply",
        ReturnConsumedCapacity = "TOTAL",
        KeyConditionExpression = "Id = :v_replyId and ReplyDateTime
between :v_start and :v_end",
        ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
            {":v_replyId", new AttributeValue {
                S = replyId
            }},
            {":v_start", new AttributeValue {
                S = start
            }},
            {":v_end", new AttributeValue {
                S = end
            }}
        }
    };

    var response = client.Query(request);

    Console.WriteLine("\nNo. of reads used (by query in
FindRepliesPostedWithinTimePeriod) {0}",
        response.ConsumedCapacity.CapacityUnits);
    foreach (Dictionary<string, AttributeValue> item
        in response.Items)
    {
        PrintItem(item);
    }
    Console.WriteLine("To continue, press Enter");
    Console.ReadLine();
}
```

```
    }

    private static void FindRepliesInLast15DaysWithConfig(string forumName, string
threadSubject)
    {
        Console.WriteLine("*** Executing FindRepliesInLast15DaysWithConfig() ***");
        string replyId = forumName + "#" + threadSubject;

        DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
        string twoWeeksAgoString =
            twoWeeksAgoDate.ToString(AWSSDKUtils.ISO8601DateFormat);

        var request = new QueryRequest
        {
            TableName = "Reply",
            ReturnConsumedCapacity = "TOTAL",
            KeyConditionExpression = "Id = :v_replyId and ReplyDateTime
> :v_interval",
            ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
                {":v_replyId", new AttributeValue {
                    S = replyId
                }},
                {":v_interval", new AttributeValue {
                    S = twoWeeksAgoString
                }},
            },

            // Optional parameter.
            ProjectionExpression = "Id, ReplyDateTime, PostedBy",
            // Optional parameter.
            ConsistentRead = true
        };

        var response = client.Query(request);

        Console.WriteLine("No. of reads used (by query in
FindRepliesInLast15DaysWithConfig) {0}",
            response.ConsumedCapacity.CapacityUnits);
        foreach (Dictionary<string, AttributeValue> item
            in response.Items)
        {
            PrintItem(item);
        }
        Console.WriteLine("To continue, press Enter");
    }
}
```

```
        Console.ReadLine();
    }

    private static void FindRepliesForAThreadSpecifyOptionalLimit(string forumName,
string threadSubject)
    {
        Console.WriteLine("*** Executing
FindRepliesForAThreadSpecifyOptionalLimit() ***");
        string replyId = forumName + "#" + threadSubject;

        Dictionary<string, AttributeValue> lastKeyEvaluated = null;
        do
        {
            var request = new QueryRequest
            {
                TableName = "Reply",
                ReturnConsumedCapacity = "TOTAL",
                KeyConditionExpression = "Id = :v_replyId",
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>
            {
                {":v_replyId", new AttributeValue {
                    S = replyId
                }}
            },
                Limit = 2, // The Reply table has only a few sample items. So the
page size is smaller.
                ExclusiveStartKey = lastKeyEvaluated
            };

            var response = client.Query(request);

            Console.WriteLine("No. of reads used (by query in
FindRepliesForAThreadSpecifyLimit) {0}\n",
                response.ConsumedCapacity.CapacityUnits);
            foreach (Dictionary<string, AttributeValue> item
                in response.Items)
            {
                PrintItem(item);
            }
            lastKeyEvaluated = response.LastEvaluatedKey;
        } while (lastKeyEvaluated != null && lastKeyEvaluated.Count != 0);

        Console.WriteLine("To continue, press Enter");
    }
}
```



```
        Console.ReadLine();
    }

    private static void FindRepliesForAThread(string forumName, string
threadSubject)
    {
        Console.WriteLine("*** Executing FindRepliesForAThread() ***");
        string replyId = forumName + "#" + threadSubject;

        var request = new QueryRequest
        {
            TableName = "Reply",
            ReturnConsumedCapacity = "TOTAL",
            KeyConditionExpression = "Id = :v_replyId",
            ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
                {":v_replyId", new AttributeValue {
                    S = replyId
                }}
            }
        };

        var response = client.Query(request);
        Console.WriteLine("No. of reads used (by query in FindRepliesForAThread)
{0}\n",
            response.ConsumedCapacity.CapacityUnits);
        foreach (Dictionary<string, AttributeValue> item in response.Items)
        {
            PrintItem(item);
        }
        Console.WriteLine("To continue, press Enter");
        Console.ReadLine();
    }

    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("*****");
}
}
```

Bekerja dengan pemindaian di DynamoDB

Operasi Scan di Amazon DynamoDB membaca setiap item dalam tabel atau indeks sekunder. Secara default, operasi Scan mengembalikan semua atribut data untuk setiap item dalam tabel atau indeks. Anda dapat menggunakan parameter `ProjectionExpression` sehingga Scan hanya mengembalikan beberapa atribut, bukan semuanya.

Scan selalu mengembalikan set hasil. Jika tidak ditemukan item yang cocok, set hasil kosong.

Permintaan Scan tunggal dapat mengambil data maksimum 1 MB. Atau, DynamoDB dapat menerapkan ekspresi filter pada data ini, mengerucutkan hasil sebelum dikembalikan ke pengguna.

Topik

- [Ekspresi filter untuk pemindaian](#)
- [Membatasi jumlah item dalam set hasil](#)
- [Memberi nomor halaman hasil](#)
- [Menghitung item dalam hasil](#)
- [Unit kapasitas dikonsumsi oleh pemindaian](#)
- [Konsistensi baca untuk kueri](#)
- [Pemindaian paralel](#)
- [Pemindaian tabel dan indeks: Java](#)
- [Memindai tabel dan indeks: NET](#)

Ekspresi filter untuk pemindaian

Jika Anda perlu menyempurnakan hasil Scan lebih lanjut, Anda dapat memberikan ekspresi filter secara opsional. Ekspresi filter menentukan item mana dalam hasil Scan yang harus dikembalikan kepada Anda. Semua hasil lainnya dibuang.

Filter ekspresi diterapkan setelah Scan selesai, tetapi sebelum hasilnya dikembalikan. Oleh karena itu, Scan menggunakan jumlah kapasitas baca yang sama, terlepas dari apakah ada ekspresi filter.

Operasi Scan dapat mengambil data maksimal 1 MB. Batasan ini berlaku sebelum ekspresi filter dievaluasi.

Dengan Scan, Anda dapat menentukan atribut apa pun dalam ekspresi filter, termasuk atribut kunci partisi dan kunci urutan.

Sintaks untuk ekspresi filter mirip dengan ekspresi kondisi kunci. Ekspresi filter dapat menggunakan perbandingan, fungsi, dan operator logika yang sama sebagai ekspresi kondisi utama. Lihat [Operator perbandingan dan referensi fungsi](#) untuk informasi selengkapnya tentang operator logika.

Example

Contoh berikut AWS Command Line Interface (AWS CLI) memindai Thread tabel dan hanya mengembalikan item yang terakhir diposting oleh pengguna tertentu.

```
aws dynamodb scan \  
  --table-name Thread \  
  --filter-expression "LastPostedBy = :name" \  
  --expression-attribute-values '{":name":{"S":"User A"}}'
```

Membatasi jumlah item dalam set hasil

Operasi Scan memungkinkan Anda membatasi jumlah item yang dikembalikan dalam hasil. Untuk melakukannya, tetapkan parameter `Limit` pada jumlah maksimum item yang Anda inginkan untuk dikembalikan operasi Scan, sebelum evaluasi ekspresi filter.

Misalnya, anggaplah bahwa Anda melakukan Scan pada tabel, dengan nilai `Limit` adalah 6, dan tanpa ekspresi filter. Hasil Scan berisi enam item pertama dari tabel.

Sekarang anggap bahwa Anda menambahkan ekspresi filter ke Scan. Dalam kasus ini, DynamoDB menerapkan ekspresi filter untuk enam item yang dikembalikan, membuang item yang tidak cocok. Hasil Scan akhir berisi enam item atau lebih sedikit, tergantung jumlah item yang disaring.

Memberi nomor halaman hasil

DynamoDB memberi nomor halaman hasil dari operasi Scan. Dengan pemberian nomor halaman, hasil Scan dibagi menjadi "halaman" data berukuran 1 MB (atau kurang). Aplikasi bisa memproses halaman pertama hasil, lalu halaman kedua, dan seterusnya.

Scan tunggal hanya mengembalikan set hasil yang sesuai dalam batas ukuran 1 MB.

Untuk menentukan apakah ada lebih banyak hasil dan mengambilnya satu halaman dalam satu waktu, aplikasi harus melakukan hal berikut:

1. Periksa hasil Scan tingkat rendah:
 - Jika hasilnya berisi elemen `LastEvaluatedKey`, lanjutkan ke langkah 2.
 - Jika tidak ada `LastEvaluatedKey` pada hasilnya, tidak ada lagi item yang dapat diambil.
2. Buat permintaan Scan baru, dengan parameter yang sama seperti yang sebelumnya. Namun, kali ini, ambil nilai `LastEvaluatedKey` dari langkah 1 dan gunakan sebagai parameter `ExclusiveStartKey` dalam permintaan Scan baru.
3. Jalankan permintaan Scan baru.
4. Lanjutkan ke langkah 1.

Dengan kata lain, `LastEvaluatedKey` dari respons Scan harus digunakan sebagai `ExclusiveStartKey` untuk permintaan Scan berikutnya. Jika tidak ada elemen `LastEvaluatedKey` dalam respons Scan, Anda telah mengambil halaman hasil akhir. (Tidak adanya `LastEvaluatedKey` adalah satu-satunya cara untuk mengetahui bahwa Anda telah mencapai akhir set hasil.)

Anda dapat menggunakan AWS CLI untuk melihat perilaku ini. AWS CLI Mengirim Scan permintaan tingkat rendah ke DynamoDB, berulang kali, `LastEvaluatedKey` sampai tidak lagi hadir dalam hasil. Perhatikan AWS CLI contoh berikut yang memindai seluruh `Movies` tabel tetapi hanya mengembalikan film dari genre tertentu.

```
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
```

Biasanya, AWS CLI menangani pagination secara otomatis. Namun, dalam contoh ini, AWS CLI `--page-size` parameter membatasi jumlah item per halaman. Parameter `--debug` mencetak informasi tingkat rendah tentang permintaan dan respons.

Note

Hasil pemberian nomor halaman Anda juga akan berbeda berdasarkan parameter input yang Anda lewati.

- Menggunakan `aws dynamodb scan --table-name Prices --max-items 1` mengembalikan `NextToken`
- Menggunakan `aws dynamodb scan --table-name Prices --limit 1` mengembalikan `LastEvaluatedKey`.

Ketahui juga bahwa menggunakan `--starting-token` khususnya membutuhkan nilai `NextToken`.

Jika Anda menjalankan contoh ini, respons pertama dari DynamoDB terlihat serupa dengan yang berikut ini.

```
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}'
```

`LastEvaluatedKey` dalam responsnya menunjukkan bahwa tidak semua item telah diambil. AWS CLI kemudian mengeluarkan Scan permintaan lain ke DynamoDB. Pola permintaan dan respons ini berlanjut hingga respons akhir.

```
2017-07-07 12:19:17,830 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"WarGames"}}], "ScannedCount":6}'
```

Tidak adanya `LastEvaluatedKey` menunjukkan bahwa tidak ada lagi item yang dapat diambil.

Note

AWS SDK menangani respons DynamoDB tingkat rendah (termasuk ada atau tidak adanya `LastEvaluatedKey`) dan menyediakan berbagai abstraksi untuk hasil paginasi. Scan Misalnya, SDK untuk antarmuka dokumen SDK for Java menyediakan dukungan `java.util.Iterator` sehingga Anda dapat mempelajari hasilnya satu per satu. Untuk contoh kode dalam berbagai bahasa pemrograman, lihat [Panduan Memulai Amazon DynamoDB](#) dan dokumentasi AWS SDK untuk bahasa Anda.

Menghitung item dalam hasil

Selain item yang sesuai dengan kriteria Anda, respons Scan berisi elemen berikut:

- `ScannedCount` — Jumlah item yang dievaluasi, sebelum `ScanFilter` diterapkan. Nilai `ScannedCount` yang tinggi dengan sedikit, atau tanpa hasil `Count`, menunjukkan operasi Scan. Jika Anda tidak menggunakan filter dalam permintaan, `ScannedCount` sama dengan `Count`.
- `Count` — Jumlah item yang tersisa setelah ekspresi filter (jika ada) diterapkan.

Note

Jika Anda tidak menggunakan ekspresi filter, `ScannedCount` dan `Count` memiliki nilai yang sama.

Jika ukuran set hasil Scan lebih besar dari 1 MB, `ScannedCount` dan `Count` mewakili hanya jumlah parsial dari total item. Anda harus melakukan beberapa operasi Scan untuk mengambil semua hasil (lihat [Memberi nomor halaman hasil](#)).

Setiap respons Scan berisi `ScannedCount` dan `Count` untuk item yang diproses oleh permintaan Scan khusus. Untuk mendapatkan total keseluruhan untuk semua permintaan Scan, Anda dapat menyimpan penghitungan `ScannedCount` dan `Count`.

Unit kapasitas dikonsumsi oleh pemindaian

Anda dapat melakukan Scan pada setiap tabel atau indeks sekunder. Operasi Scan mengonsumsi unit kapasitas baca, sebagai berikut.

Jika Anda Scan...	DynamoDB menggunakan unit kapasitas baca dari...
Tabel	Kapasitas baca yang ditetapkan tabel.
Indeks sekunder global	Kapasitas baca yang ditetapkan indeks.
Indeks sekunder lokal	Kapasitas baca yang ditetapkan tabel dasar.

Secara default, operasi Scan tidak menampilkan data tentang berapa banyak kapasitas baca yang digunakan. Namun, Anda dapat menentukan parameter `ReturnConsumedCapacity` dalam sebuah permintaan Scan untuk mendapatkan informasi ini. Berikut ini adalah pengaturan yang valid untuk `ReturnConsumedCapacity`:

- **NONE** — Tidak ada penggunaan data kapasitas yang ditampilkan. (Ini menjadi opsi default.)
- **TOTAL** — Respons mencakup jumlah agregat unit kapasitas baca yang digunakan.
- **INDEXES** — Respons menunjukkan jumlah agregat unit kapasitas baca yang digunakan, beserta dengan kapasitas yang digunakan untuk setiap tabel dan indeks yang diakses.

DynamoDB menghitung jumlah unit kapasitas baca yang dikonsumsi berdasarkan jumlah item dan ukuran item tersebut, bukan pada jumlah data yang dikembalikan ke aplikasi. Dengan alasan ini, jumlah unit kapasitas yang digunakan adalah sama, entah Anda meminta semua atribut (perilaku default) atau hanya beberapa dari atribut tersebut (menggunakan ekspresi proyeksi). Jumlahnya juga sama apakah Anda menggunakan ekspresi filter atau tidak. Scan mengonsumsi unit kapasitas baca minimum untuk melakukan satu pembacaan yang sangat konsisten per detik, atau dua pembacaan yang konsisten per detik untuk item hingga 4 KB. Jika Anda perlu membaca item yang lebih besar dari 4 KB, DynamoDB memerlukan unit permintaan baca tambahan. Tabel kosong dan tabel yang sangat besar yang memiliki jumlah kunci partisi yang jarang mungkin melihat beberapa RCU tambahan yang dibebankan melebihi jumlah data yang dipindai. Ini mencakup biaya melayani Scan permintaan, bahkan jika tidak ada data.

Konsistensi baca untuk kueri

Operasi Scan melakukan bacaan akhir konsisten, secara default. Ini berarti bahwa hasil Scan mungkin tidak mencerminkan perubahan karena operasi `PutItem` atau `UpdateItem` baru saja selesai. Untuk informasi selengkapnya, lihat [Konsistensi baca](#).

Jika Anda memerlukan bacaan sangat konsisten, pada saat Scan dimulai, atur parameter `ConsistentRead` menjadi `true` dalam permintaan Scan. Hal ini memastikan bahwa semua operasi tulis yang diselesaikan sebelum Scan dimulai termasuk dalam respons Scan.

Mengatur `ConsistentRead` ke `true` dapat berguna dalam skenario pencadangan atau replikasi tabel, dalam hubungannya dengan [DynamoDB Streams](#). Anda pertama kali menggunakan Scan dengan `ConsistentRead` yang diatur ke `true` untuk mendapatkan salinan data yang konsisten dalam tabel. Selama Scan, DynamoDB Streams mencatat aktivitas tulis tambahan yang terjadi pada tabel. Setelah Scan selesai, Anda dapat menerapkan aktivitas tulis dari stream ke table.

Note

Operasi Scan dengan `ConsistentRead` yang diatur ke `true` mengonsumsi dua kali lebih banyak unit kapasitas baca dibandingkan dengan meninggalkan `ConsistentRead` pada nilai defaultnya (`false`).

Pemindaian paralel

Secara default, operasi Scan memproses data secara berurutan. Amazon DynamoDB mengembalikan data ke aplikasi dalam kenaikan 1 MB, dan aplikasi melakukan operasi Scan tambahan untuk mengambil 1 MB data berikutnya.

Makin besar tabel atau indeks yang dipindai, makin banyak waktu yang diperlukan untuk menyelesaikan Scan. Selain itu, Scan berurutan mungkin tidak selalu dapat sepenuhnya menggunakan kapasitas throughput baca yang disediakan: Meskipun DynamoDB mendistribusikan data tabel besar ke beberapa partisi fisik, operasi Scan hanya dapat membaca satu partisi dalam satu waktu. Karena alasan ini, throughput Scan dibatasi oleh throughput maksimum dari satu partisi.

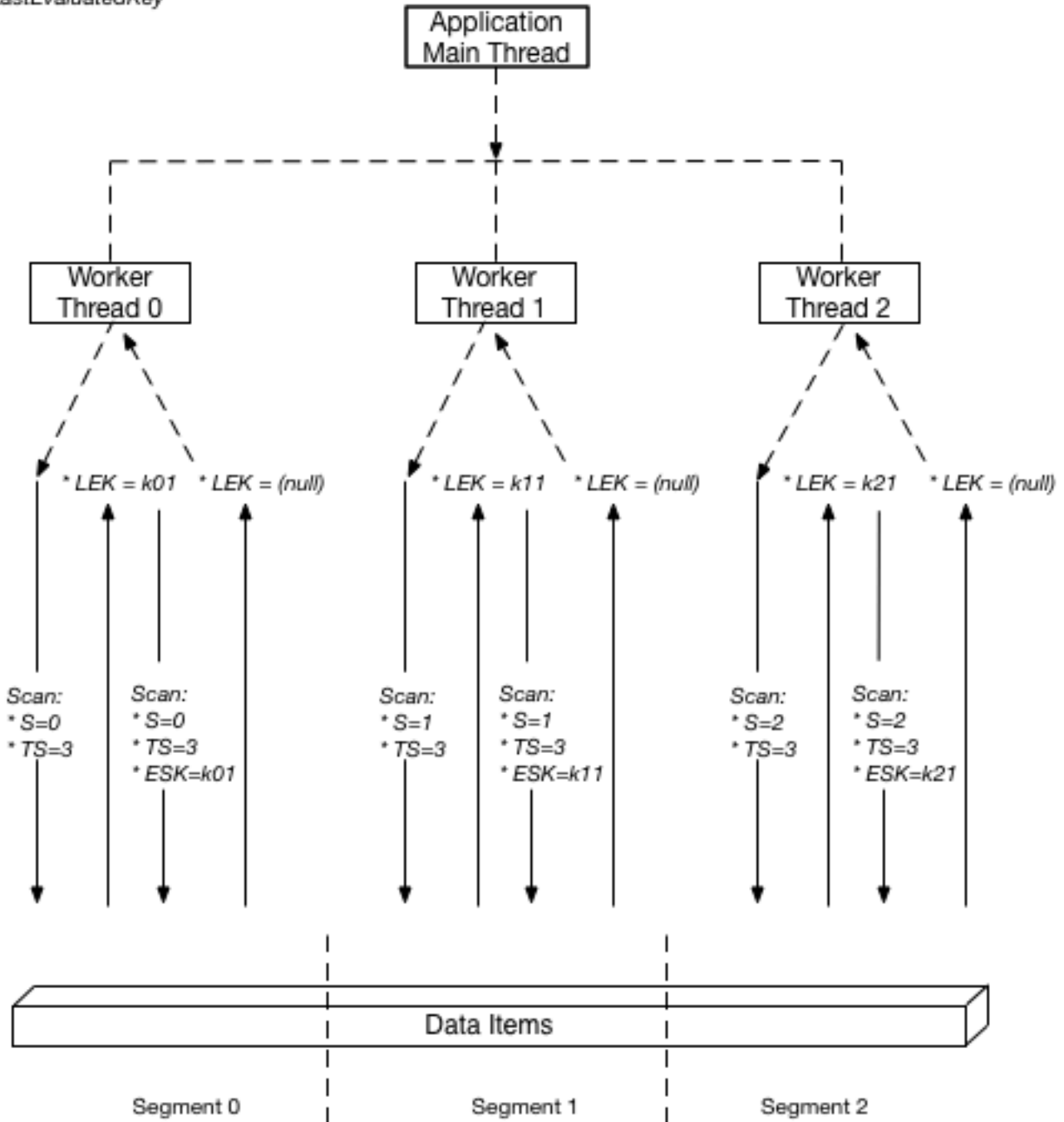
Untuk mengatasi masalah ini, operasi Scan dapat secara logis membagi tabel atau indeks sekunder menjadi beberapa segmen, dengan beberapa pekerja aplikasi memindai segmen tersebut secara paralel. Setiap pekerja dapat berupa thread (dalam bahasa pemrograman yang mendukung multithreading) atau proses sistem operasi. Untuk melakukan pemindaian paralel, setiap pekerja mengeluarkan permintaan Scan-nya sendiri dengan parameter berikut:

- `Segment` — Segmen yang akan dipindai oleh pekerja tertentu. Setiap pekerja harus menggunakan nilai yang berbeda untuk `Segment`.
- `TotalSegments` — Jumlah total segmen untuk pemindaian paralel. Nilai ini harus sama dengan jumlah pekerja yang akan digunakan aplikasi Anda.

Diagram berikut menunjukkan bagaimana aplikasi multithreaded melakukan Scan paralel dengan tiga derajat paralelisme.

S: Segment
TS: TotalSegments

ESK: ExclusiveStartKey
LEK: LastEvaluatedKey



Dalam diagram ini, aplikasi memunculkan tiga thread dan memberikan angka pada setiap thread. (Segmen berbasis nol, sehingga angka pertama selalu 0.) Setiap thread menerbitkan permintaan Scan, mengatur Segment ke angka yang ditunjuk dan mengatur TotalSegments ke 3. Setiap thread memindai segmen yang ditunjuk, mengambil data 1 MB pada satu waktu, dan mengembalikan data ke thread utama aplikasi.

Nilai untuk Segment dan TotalSegments diterapkan untuk permintaan Scan individual, dan Anda dapat menggunakan nilai yang berbeda setiap saat. Anda mungkin perlu bereksperimen dengan nilai-nilai ini, dan jumlah pekerja yang Anda gunakan, sampai aplikasi Anda mencapai performa terbaiknya.

Note

Pemindaian paralel dengan sejumlah besar pekerja dapat dengan mudah menggunakan semua throughput yang disediakan untuk tabel atau indeks yang sedang dipindai. Sebaiknya hindari pemindaian seperti itu jika tabel atau indeks juga mengalami aktivitas baca atau tulis yang berat dari aplikasi lain.

Untuk mengontrol jumlah data yang dikembalikan per permintaan, gunakan parameter Limit. Hal ini dapat membantu mencegah situasi di mana satu pekerja mengkonsumsi semua throughput yang ditetapkan, dengan mengorbankan semua pekerja lainnya.

Pemindaian tabel dan indeks: Java

Operasi Scan membaca semua item dalam tabel atau indeks di Amazon DynamoDB.

Berikut ini adalah langkah-langkah untuk memindai tabel menggunakan AWS SDK for Java Document API.

1. Buat instans kelas AmazonDynamoDB.
2. Buat instans dari kelas ScanRequest dan berikan parameter pemindaian.

Satu-satunya parameter yang diperlukan adalah nama tabel.

3. Jalankan metode scan dan sediakan objek ScanRequest yang Anda buat pada langkah sebelumnya.

Tabel Reply berikut menyimpan balasan untuk thread forum.

Example

```
Reply ( Id, ReplyDateTime, Message, PostedBy )
```

Tabel mengelola semua balasan untuk berbagai thread forum. Oleh karena itu, kunci primer terdiri dari baik Id (kunci partisi) dan ReplyDateTime (kunci urutan). Contoh kode Java berikut contoh memindai seluruh tabel. Instans ScanRequest menentukan nama tabel yang akan dipindai.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

ScanRequest scanRequest = new ScanRequest()
    .withTableName("Reply");

ScanResponse result = client.scan(scanRequest);
for (Map<String, AttributeValue> item : result.getItems()){
    printItem(item);
}
```

Menentukan parameter opsional

Metode scan mendukung beberapa parameter opsional. Misalnya, Anda dapat menggunakan ekspresi filter untuk menyaring hasil pemindaian. Dalam ekspresi filter, Anda dapat menentukan kondisi dan nama atribut dan nilai-nilai yang Anda inginkan untuk dievaluasi kondisi. Untuk informasi selengkapnya, lihat [Scan](#).

Contoh Java berikut memindai tabel ProductCatalog untuk menemukan item dengan harga kurang dari 0. Contoh tersebut menentukan parameter opsional berikut:

- Sebuah filter ekspresi untuk mengambil hanya item dengan harga kurang dari 0 (kondisi kesalahan).
- Daftar atribut untuk mengambil item dalam hasil kueri.

Example

```
Map<String, AttributeValue> expressionAttributeValues =
    new HashMap<String, AttributeValue>();
expressionAttributeValues.put(":val", new AttributeValue().withN("0"));
```

```
ScanRequest scanRequest = new ScanRequest()
    .withTableName("ProductCatalog")
    .withFilterExpression("Price < :val")
    .withProjectionExpression("Id")
    .withExpressionAttributeValues(expressionAttributeValues);

ScanResponse result = client.scan(scanRequest);
for (Map<String, AttributeValue> item : result.getItems()) {
    printItem(item);
}
```

Anda juga dapat secara opsional membatasi ukuran halaman, atau jumlah item per halaman, dengan menggunakan metode `withLimit` dari permintaan pemindaian. Setiap kali Anda menjalankan metode `scan`, Anda mendapatkan satu halaman hasil yang memiliki jumlah item tertentu. Untuk mengambil halaman berikutnya, jalankan kembali metode `scan` dengan memberikan nilai kunci primer item terakhir di halaman sebelumnya sehingga metode `scan` dapat mengembalikan kumpulan item berikutnya. Anda memberikan informasi ini dalam permintaan dengan menggunakan metode `withExclusiveStartKey`. Awalnya, parameter metode ini bisa null. Untuk mengambil halaman berikutnya, Anda harus memperbarui nilai properti ini ke kunci primer item terakhir di halaman sebelumnya.

Contoh kode Java berikut contoh memindai tabel `ProductCatalog`. Dalam permintaan, metode `withLimit` dan `withExclusiveStartKey` digunakan. Perulangan `do/while` terus memindai satu halaman sekaligus hingga metode `getLastEvaluatedKey` hasil menampilkan nilai null.

Example

```
Map<String, AttributeValue> lastKeyEvaluated = null;
do {
    ScanRequest scanRequest = new ScanRequest()
        .withTableName("ProductCatalog")
        .withLimit(10)
        .withExclusiveStartKey(lastKeyEvaluated);

    ScanResponse result = client.scan(scanRequest);
    for (Map<String, AttributeValue> item : result.getItems()){
        printItem(item);
    }
    lastKeyEvaluated = result.getLastEvaluatedKey();
} while (lastKeyEvaluated != null);
```

Contoh - pindai menggunakan Java

Contoh kode Java berikut memberikan sampel kerja yang memindai tabel ProductCatalog untuk menemukan item dengan harga kurang dari 100.

Note

SDK untuk Java juga menyediakan model persistensi objek, memungkinkan Anda memetakan kelas sisi klien ke tabel DynamoDB. Pendekatan ini dapat mengurangi jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat [Java 1.x: DynamoDBMapper](#).

Note

Contoh kode ini mengasumsikan bahwa Anda telah memuat data ke DynamoDB untuk akun Anda dengan mengikuti petunjuk di bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihat [Contoh kode Java](#).

```
package com.amazonaws.codesamples.document;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

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.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.ScanOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;

public class DocumentAPIScan {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);
```

```
static String tableName = "ProductCatalog";

public static void main(String[] args) throws Exception {

    findProductsForPriceLessThanOneHundred();
}

private static void findProductsForPriceLessThanOneHundred() {

    Table table = dynamoDB.getTable(tableName);

    Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
    expressionAttributeValues.put(":pr", 100);

    ItemCollection<ScanOutcome> items = table.scan("Price < :pr", //
FilterExpression
        "Id, Title, ProductCategory, Price", // ProjectionExpression
        null, // ExpressionAttributeNames - not used in this example
        expressionAttributeValues);

    System.out.println("Scan of " + tableName + " for items with a price less than
100.");
    Iterator<Item> iterator = items.iterator();
    while (iterator.hasNext()) {
        System.out.println(iterator.next().toJSONPretty());
    }
}
}
```

Contoh - pemindaian paralel menggunakan Java

Contoh kode Java berikut menunjukkan pemindaian paralel. Program ini menghapus dan menciptakan kembali tabel bernama `ParallelScanTest`, kemudian memuat tabel dengan data. Ketika beban data selesai, program memunculkan beberapa thread dan menerbitkan permintaan Scan paralel. Program ini mencetak statistik runtime untuk setiap permintaan paralel.

Note

SDK untuk Java juga menyediakan model persistensi objek, memungkinkan Anda memetakan kelas sisi klien ke tabel DynamoDB. Pendekatan ini dapat mengurangi

jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat [Java 1.x: DynamoDBMapper](#).

Note

Contoh kode ini mengasumsikan bahwa Anda telah memuat data ke DynamoDB untuk akun Anda dengan mengikuti petunjuk di bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihat [Contoh kode Java](#).

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

import com.amazonaws.AmazonServiceException;
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.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.ScanOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.ScanSpec;
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;

public class DocumentAPIParallelScan {

    // total number of sample items
```

```
static int scanItemCount = 300;

// number of items each scan request should return
static int scanItemLimit = 10;

// number of logical segments for parallel scan
static int parallelScanThreads = 16;

// table that will be used for scanning
static String parallelScanTestTableName = "ParallelScanTest";

static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
static DynamoDB dynamoDB = new DynamoDB(client);

public static void main(String[] args) throws Exception {
    try {

        // Clean up the table
        deleteTable(parallelScanTestTableName);
        createTable(parallelScanTestTableName, 10L, 5L, "Id", "N");

        // Upload sample data for scan
        uploadSampleProducts(parallelScanTestTableName, scanItemCount);

        // Scan the table using multiple threads
        parallelScan(parallelScanTestTableName, scanItemLimit,
parallelScanThreads);
    } catch (AmazonServiceException ase) {
        System.err.println(ase.getMessage());
    }
}

private static void parallelScan(String tableName, int itemLimit, int
numberOfThreads) {
    System.out.println(
        "Scanning " + tableName + " using " + numberOfThreads + " threads " +
itemLimit + " items at a time");
    ExecutorService executor = Executors.newFixedThreadPool(numberOfThreads);

    // Divide DynamoDB table into logical segments
    // Create one task for scanning each segment
    // Each thread will be scanning one segment
    int totalSegments = numberOfThreads;
    for (int segment = 0; segment < totalSegments; segment++) {
```



```
        // Runnable task that will only scan one segment
        ScanSegmentTask task = new ScanSegmentTask(tableName, itemLimit,
totalSegments, segment);

        // Execute the task
        executor.execute(task);
    }

    shutdownExecutorService(executor);
}

// Runnable task for scanning a single segment of a DynamoDB table
private static class ScanSegmentTask implements Runnable {

    // DynamoDB table to scan
    private String tableName;

    // number of items each scan request should return
    private int itemLimit;

    // Total number of segments
    // Equals to total number of threads scanning the table in parallel
    private int totalSegments;

    // Segment that will be scanned with by this task
    private int segment;

    public ScanSegmentTask(String tableName, int itemLimit, int totalSegments, int
segment) {
        this.tableName = tableName;
        this.itemLimit = itemLimit;
        this.totalSegments = totalSegments;
        this.segment = segment;
    }

    @Override
    public void run() {
        System.out.println("Scanning " + tableName + " segment " + segment + " out
of " + totalSegments
            + " segments " + itemLimit + " items at a time...");
        int totalScannedItemCount = 0;

        Table table = dynamoDB.getTable(tableName);
```

```
        try {
            ScanSpec spec = new
ScanSpec().withMaxResultSize(itemLimit).withTotalSegments(totalSegments)
                .withSegment(segment);

            ItemCollection<ScanOutcome> items = table.scan(spec);
            Iterator<Item> iterator = items.iterator();

            Item currentItem = null;
            while (iterator.hasNext()) {
                totalScannedItemCount++;
                currentItem = iterator.next();
                System.out.println(currentItem.toString());
            }

        } catch (Exception e) {
            System.err.println(e.getMessage());
        } finally {
            System.out.println("Scanned " + totalScannedItemCount + " items from
segment " + segment + " out of "
                + totalSegments + " of " + tableName);
        }
    }
}

private static void uploadSampleProducts(String tableName, int itemCount) {
    System.out.println("Adding " + itemCount + " sample items to " + tableName);
    for (int productIndex = 0; productIndex < itemCount; productIndex++) {
        uploadProduct(tableName, productIndex);
    }
}

private static void uploadProduct(String tableName, int productIndex) {

    Table table = dynamoDB.getTable(tableName);

    try {
        System.out.println("Processing record #" + productIndex);

        Item item = new Item().withPrimaryKey("Id", productIndex)
            .withString("Title", "Book " + productIndex + "
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);

    } catch (Exception e) {
        System.err.println("Failed to create item " + productIndex + " in " +
tableName);
        System.err.println(e.getMessage());
    }
}

private static void deleteTable(String tableName) {
    try {

        Table table = dynamoDB.getTable(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("Failed to delete table " + tableName);
        e.printStackTrace(System.err);
    }
}

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 {
        System.out.println("Creating table " + tableName);
```

```
        List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
        keySchema.add(new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH)); //
Partition

                // key

        List<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));
        }

        Table table = dynamoDB.createTable(tableName, keySchema,
attributeDefinitions, new ProvisionedThroughput()

.withReadCapacityUnits(readCapacityUnits).withWriteCapacityUnits(writeCapacityUnits));
        System.out.println("Waiting for " + tableName + " to be created...this may
take a while...");
        table.waitForActive();

    } catch (Exception e) {
        System.err.println("Failed to create table " + tableName);
        e.printStackTrace(System.err);
    }
}

private static void shutDownExecutorService(ExecutorService executor) {
    executor.shutdown();
    try {
        if (!executor.awaitTermination(10, TimeUnit.SECONDS)) {
            executor.shutdownNow();
        }
    } catch (InterruptedException e) {
```

```
        executor.shutdownNow();

        // Preserve interrupt status
        Thread.currentThread().interrupt();
    }
}
```

Memindai tabel dan indeks: NET

Operasi Scan membaca semua item dalam tabel atau indeks di Amazon DynamoDB.

Berikut ini adalah langkah-langkah untuk memindai tabel menggunakan API AWS SDK for .NET tingkat rendah:

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Buat instans dari kelas `ScanRequest` dan berikan parameter operasi kueri.

Satu-satunya parameter yang diperlukan adalah nama tabel.

3. Jalankan metode `Scan` dan sediakan objek `ScanRequest` yang Anda buat pada langkah sebelumnya.

Tabel `Reply` berikut menyimpan balasan untuk thread forum.

Example

```
>Reply ( Id, ReplyDateTime, Message, PostedBy )
```

Tabel mengelola semua balasan untuk berbagai thread forum. Oleh karena itu, kunci primer terdiri dari baik `Id` (kunci partisi) dan `ReplyDateTime` (kunci urutan). Contoh kode C# berikut contoh memindai seluruh tabel. Instans `ScanRequest` menentukan nama tabel yang akan dipindai.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
```

```
var request = new ScanRequest
{
    TableName = "Reply",
};

var response = client.Scan(request);
var result = response.ScanResult;

foreach (Dictionary<string, AttributeValue> item in response.ScanResult.Items)
{
    // Process the result.
    PrintItem(item);
}
```

Menentukan parameter opsional

Metode Scan mendukung beberapa parameter opsional. Misalnya, Anda dapat menggunakan filter pemindaian untuk memfilter hasil pemindaian. Dalam filter pemindaian, Anda dapat menentukan kondisi dan nama atribut yang ingin Anda evaluasi kondisinya. Untuk informasi selengkapnya, lihat [Scan](#).

Kode C# berikut memindai tabel ProductCatalog untuk menemukan item dengan harga kurang dari 0. Contoh tersebut menentukan parameter opsional berikut:

- Parameter `FilterExpression` untuk mengambil hanya item dengan harga kurang dari 0 (kondisi kesalahan).
- Parameter `ProjectionExpression` untuk menentukan atribut untuk mengambil item dalam hasil kueri.

Contoh C# berikut memindai tabel ProductCatalog untuk menemukan semua item dengan harga kurang dari 0.

Example

```
var forumScanRequest = new ScanRequest
{
    TableName = "ProductCatalog",
    // Optional parameters.
    ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
        {":val", new AttributeValue { N = "0" }}
    }
}
```

```
},  
FilterExpression = "Price < :val",  
ProjectionExpression = "Id"  
};
```

Selain itu, Anda juga dapat membatasi ukuran halaman atau jumlah item per halaman, dengan menambahkan parameter `Limit` opsional. Setiap kali Anda menjalankan metode `Scan`, Anda mendapatkan satu halaman hasil yang memiliki jumlah item tertentu. Untuk mengambil halaman berikutnya, jalankan kembali metode `Scan` dengan memberikan nilai kunci primer item terakhir di halaman sebelumnya sehingga metode `Scan` dapat mengembalikan kumpulan item berikutnya. Anda memberikan informasi ini dalam permintaan dengan menetapkan properti `ExclusiveStartKey`. Awalnya, properti ini bisa bernilai `null`. Untuk mengambil halaman berikutnya, Anda harus memperbarui nilai properti ini ke kunci primer item terakhir di halaman sebelumnya.

Contoh kode C# berikut memindai tabel `ProductCatalog`. Dalam permintaan, ini menentukan parameter opsional `Limit` dan `ExclusiveStartKey`. Perulangan `do/while` terus memindai satu halaman sekaligus hingga `LastEvaluatedKey` menampilkan nilai `null`.

Example

```
Dictionary<string, AttributeValue> lastKeyEvaluated = null;  
do  
{  
    var request = new ScanRequest  
    {  
        TableName = "ProductCatalog",  
        Limit = 10,  
        ExclusiveStartKey = lastKeyEvaluated  
    };  
  
    var response = client.Scan(request);  
  
    foreach (Dictionary<string, AttributeValue> item  
        in response.Items)  
    {  
        PrintItem(item);  
    }  
    lastKeyEvaluated = response.LastEvaluatedKey;  
} while (lastKeyEvaluated != null && lastKeyEvaluated.Count != 0);
```

Contoh - pemindaian menggunakan .NET

Kode C# berikut memberikan sampel kerja yang memindai tabel ProductCatalog untuk menemukan item dengan harga kurang dari 0.

Untuk step-by-step petunjuk pengujian sampel berikut, lihat [Contoh kode .NET](#).

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelScan
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                FindProductsForPriceLessThanZero();

                Console.WriteLine("Example complete. To continue, press Enter");
                Console.ReadLine();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
        }

        private static void FindProductsForPriceLessThanZero()
        {
            Dictionary<string, AttributeValue> lastKeyEvaluated = null;
            do
            {
                var request = new ScanRequest
                {
                    TableName = "ProductCatalog",
```



```
        Limit = 2,
        ExclusiveStartKey = lastKeyEvaluated,
        ExpressionAttributeValues = new Dictionary<string, AttributeValue>
    {
        {":val", new AttributeValue {
            N = "0"
        }}
    },
    FilterExpression = "Price < :val",

    ProjectionExpression = "Id, Title, Price"
};

var response = client.Scan(request);

foreach (Dictionary<string, AttributeValue> item
    in response.Items)
{
    Console.WriteLine("\nScanThreadTableUsePaging - printing.....");
    PrintItem(item);
}
lastKeyEvaluated = response.LastEvaluatedKey;
} while (lastKeyEvaluated != null && lastKeyEvaluated.Count != 0);

Console.WriteLine("To continue, press Enter");
Console.ReadLine();
}

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("*****");  
}  
}
```

Contoh - pemindaian paralel menggunakan .NET

Contoh kode C# berikut menunjukkan pemindaian paralel. Program ini menghapus dan membuat ulang tabel bernama ProductCatalog, lalu memuat tabel tersebut dengan data. Ketika beban data selesai, program memunculkan beberapa thread dan menerbitkan permintaan Scan paralel. Akhirnya, program ini mencetak ringkasan statistik runtime.

Untuk step-by-step petunjuk pengujian sampel berikut, lihat [Contoh kode .NET](#).

```
using System;  
using System.Collections.Generic;  
using System.Threading;  
using System.Threading.Tasks;  
using Amazon.DynamoDBv2;  
using Amazon.DynamoDBv2.Model;  
using Amazon.Runtime;  
  
namespace com.amazonaws.codesamples  
{  
    class LowLevelParallelScan  
    {  
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();  
        private static string tableName = "ProductCatalog";  
        private static int exampleItemCount = 100;  
        private static int scanItemLimit = 10;  
        private static int totalSegments = 5;  
  
        static void Main(string[] args)  
        {  
            try  
            {  
                DeleteExampleTable();  
                CreateExampleTable();  
                UploadExampleData();  
                ParallelScanExampleTable();  
            }  
        }  
    }  
}
```

```
    }
    catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
    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 ParallelScanExampleTable()
{
    Console.WriteLine("\n*** Creating {0} Parallel Scan Tasks to scan {1}",
totalSegments, tableName);
    Task[] tasks = new Task[totalSegments];
    for (int segment = 0; segment < totalSegments; segment++)
    {
        int tmpSegment = segment;
        Task task = Task.Factory.StartNew(() =>
            {
                ScanSegment(totalSegments, tmpSegment);
            });

        tasks[segment] = task;
    }

    Console.WriteLine("All scan tasks are created, waiting for them to
complete.");
    Task.WaitAll(tasks);

    Console.WriteLine("All scan tasks are completed.");
}

private static void ScanSegment(int totalSegments, int segment)
{
    Console.WriteLine("*** Starting to Scan Segment {0} of {1} out of {2} total
segments ***", segment, tableName, totalSegments);
    Dictionary<string, AttributeValue> lastEvaluatedKey = null;
    int totalScannedItemCount = 0;
    int totalScanRequestCount = 0;
    do
    {
        var request = new ScanRequest
        {
            TableName = tableName,
```

```
        Limit = scanItemLimit,
        ExclusiveStartKey = lastEvaluatedKey,
        Segment = segment,
        TotalSegments = totalSegments
    };

    var response = client.Scan(request);
    lastEvaluatedKey = response.LastEvaluatedKey;
    totalScanRequestCount++;
    totalScannedItemCount += response.ScannedCount;
    foreach (var item in response.Items)
    {
        Console.WriteLine("Segment: {0}, Scanned Item with Title: {1}",
segment, item["Title"].S);
    }
    } while (lastEvaluatedKey.Count != 0);

    Console.WriteLine("*** Completed Scan Segment {0} of {1}.
TotalScanRequestCount: {2}, TotalScannedItemCount: {3} ***", segment, tableName,
totalScanRequestCount, totalScannedItemCount);
}

private static void UploadExampleData()
{
    Console.WriteLine("\n*** Uploading {0} Example Items to {1} Table***",
exampleItemCount, tableName);
    Console.Write("Uploading Items: ");
    for (int itemIndex = 0; itemIndex < exampleItemCount; itemIndex++)
    {
        Console.Write("{0}, ", itemIndex);
        CreateItem(itemIndex.ToString());
    }
    Console.WriteLine();
}

private static void CreateItem(string itemIndex)
{
    var request = new PutItemRequest
    {
        TableName = tableName,
        Item = new Dictionary<string, AttributeValue>()
    {
        { "Id", new AttributeValue {
            N = itemIndex
```

```
        }},
        { "Title", new AttributeValue {
            S = "Book " + itemIndex + " 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 CreateExampleTable()
{
    Console.WriteLine("\n*** Creating {0} Table ***", tableName);
    var request = new CreateTableRequest
    {
        AttributeDefinitions = new List<AttributeDefinition>()
        {
            new AttributeDefinition
            {
                AttributeName = "Id",
                AttributeType = "N"
            }
        },
        KeySchema = new List<KeySchemaElement>
        {
            new KeySchemaElement
            {
                AttributeName = "Id",
                KeyType = "HASH" //Partition key
            }
        }
    }
}
```

```
    },
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = 5,
        WriteCapacityUnits = 6
    },
    TableName = tableName
};

var response = client.CreateTable(request);

var result = response;
var tableDescription = result.TableDescription;
Console.WriteLine("{1}: {0} \t ReadsPerSec: {2} \t WritesPerSec: {3}",
    tableDescription.TableStatus,
    tableDescription.TableName,
    tableDescription.ProvisionedThroughput.ReadCapacityUnits,
    tableDescription.ProvisionedThroughput.WriteCapacityUnits);

string status = tableDescription.TableStatus;
Console.WriteLine(tableName + " - " + status);

WaitUntilTableReady(tableName);
}

private static void DeleteExampleTable()
{
    try
    {
        Console.WriteLine("\n*** Deleting {0} Table ***", tableName);
        var request = new DeleteTableRequest
        {
            TableName = tableName
        };

        var response = client.DeleteTable(request);
        var result = response;
        Console.WriteLine("{0} is being deleted...", tableName);
        WaitUntilTableDeleted(tableName);
    }
    catch (ResourceNotFoundException)
    {
        Console.WriteLine("{0} Table delete failed: Table does not exist",
            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 WaitUntilTableDeleted(string tableName)  
{  
    string status = null;  
    // Let us wait until table is deleted. 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)
    {
        Console.WriteLine("Table name: {0} is not found. It is deleted",
            tableName);
        return;
    }
} while (status == "DELETING");
}
}
```

PartiQL - Bahasa kueri yang kompatibel dengan SQL untuk Amazon DynamoDB

Amazon DynamoDB mendukung [PartiQL](#), bahasa kueri yang kompatibel dengan SQL untuk memilih, menyisipkan, memperbarui, dan menghapus data di Amazon DynamoDB. Dengan PartiQL, Anda dapat dengan mudah berinteraksi menggunakan tabel DynamoDB dan menjalankan kueri ad hoc menggunakan AWS Management Console, NoSQL Workbench, AWS Command Line Interface, dan DynamoDB API untuk PartiQL.

Operasi PartiQL memberikan ketersediaan, latensi, dan performa yang sama seperti operasi bidang data DynamoDB lainnya.

Bagian berikut menjelaskan implementasi DynamoDB PartiQL.

Topik

- [Apa yang dimaksud dengan PartiQL?](#)
- [PartiQL di Amazon DynamoDB](#)
- [Mulai Menggunakan PartiQL untuk DynamoDB](#)
- [Tipe data PartiQL untuk DynamoDB](#)
- [Pernyataan PartiQL untuk DynamoDB](#)
- [Gunakan fungsi PartiQL dengan amazon DynamoDB](#)
- [PartiQL aritmatika, perbandingan, dan operator logis untuk DynamoDB](#)

- [Melakukan transaksi dengan PartiQL untuk DynamoDB](#)
- [Menjalankan operasi batch dengan PartiQL untuk DynamoDB](#)
- [Kebijakan keamanan IAM dengan PartiQL untuk DynamoDB](#)

Apa yang dimaksud dengan PartiQL?

PartiQL menyediakan akses kueri yang kompatibel dengan SQL di beberapa penyimpanan data yang berisi data terstruktur, data semi terstruktur, dan data bersarang. Layanan ini banyak digunakan dalam Amazon dan sekarang tersedia sebagai bagian dari berbagai layanan AWS, termasuk DynamoDB.

Untuk spesifikasi PartiQL dan tutorial tentang bahasa kueri inti, lihat [Dokumentasi PartiQL](#).

Note

- Amazon DynamoDB mendukung subset dari bahasa kueri [PartiQL](#).
- Amazon DynamoDB tidak mendukung format data [Amazon ion](#) atau literal Amazon Ion.

PartiQL di Amazon DynamoDB

Untuk menjalankan kueri PartiQL di DynamoDB, Anda dapat menggunakan:

- Konsol DynamoDB
- NoSQL Workbench
- AWS Command Line Interface (AWS CLI)
- DynamoDB API

Untuk informasi tentang cara menggunakan metode ini guna mengakses DynamoDB, lihat [Mengakses DynamoDB](#).

Mulai Menggunakan PartiQL untuk DynamoDB

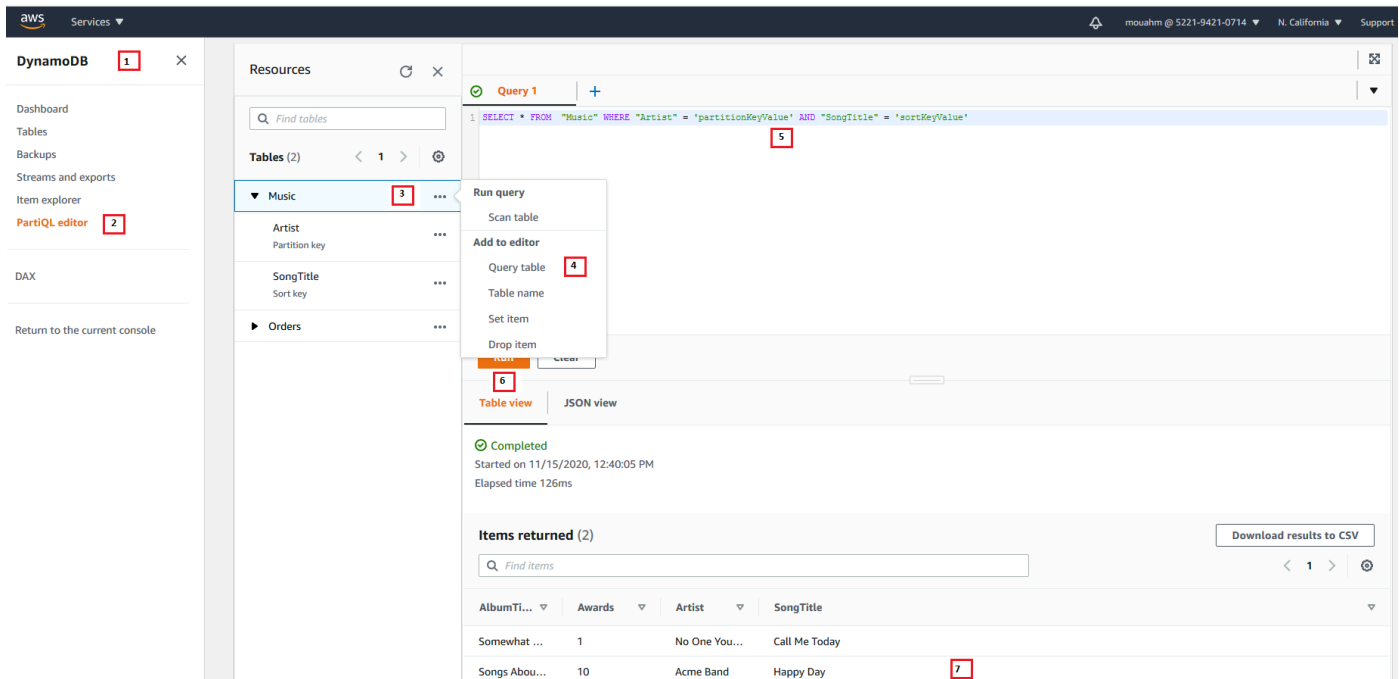
Bagian ini menjelaskan cara menggunakan PartiQL untuk DynamoDB dari konsol Amazon DynamoDB, AWS Command Line Interface (AWS CLI), dan DynamoDB API.

Dalam contoh berikut, tabel DynamoDB yang didefinisikan dalam [Mulai Menggunakan DynamoDB](#) tutorial adalah prasyarat.

Untuk informasi tentang menggunakan konsol DynamoDB, AWS Command Line Interface, atau DynamoDB API untuk mengakses DynamoDB, lihat [Mengakses DynamoDB](#).

Keunduhan dan gunakan [NoSQL Workbench](#) build [PartiQL untuk DynamoDB](#) pernyataan memilih Operasi PartiQL di sudut kanan atas NoSQL for DynamoDB [Pembangun Operasi](#).

Console



The screenshot shows the AWS DynamoDB console interface. On the left, the navigation pane has 'PartiQL editor' highlighted with a red box labeled '2'. The main area shows the 'Music' table selected with a red box labeled '3'. A context menu is open over the table, with 'Query table' selected, highlighted with a red box labeled '4'. The query editor shows a PartiQL query with a red box labeled '5' around the table name. Below the query, the execution status is 'Completed' with a red box labeled '6' around the 'Run query' button. The results section shows 'Items returned (2)' with a red box labeled '7' around the table of results.

AlbumTI...	Awards	Artist	SongTitle
Somewhat ...	1	No One You...	Call Me Today
Songs Abou...	10	Acme Band	Happy Day

Note

PartiQL untuk DynamoDB hanya tersedia di konsol DynamoDB baru. Untuk menggunakan konsol DynamoDB baru, pilih Coba Pratinjau konsol baru dalam panel navigasi di sisi kiri konsol.

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi pada sisi kiri konsol, pilih editor PartiQL.
3. Pilih tabel Musik.

4. Pilih Tabel kueri. Tindakan ini membuat kueri yang tidak akan menghasilkan pemindaian tabel penuh.
5. Ganti `partitionKeyValue` dengan nilai string `Acme Band`. Ganti `sortKeyValue` dengan nilai string `Happy Day`.
6. Pilih tombol Jalankan.
7. Anda dapat melihat hasil kueri dengan memilih tombol Tampilan tabel atau Tampilan JSON.

NoSQL workbench

PartiQL statement
 PartiQL transaction
 PartiQL batch

1

Statement

```
1 SELECT *
2 FROM Music
3 WHERE Artist=? and SongTitle=?
```

2

Optional request parameters 3.a

Enable strongly consistent reads i

Parameters i

Attribute type	Attribute value 3.c	
<input type="text" value="String"/>	<input type="text" value="Acme Band"/>	<small>🗑️</small>
Attribute type	Attribute value	
<input type="text" value="String"/>	<input type="text" value="PartiQL Rocks"/>	<small>🗑️</small>

+ Add new parameter 3.b

5
4
6

Clear form
Run
Generate code
Save operation

▲ Hide operation

1. Pilih Pernyataan PartiQL.
2. Masukkan [Pernyataan SELECT](#) PartiQL berikut

```
SELECT *
```

```
FROM Music
WHERE Artist=? and SongTitle=?
```

3. Untuk menentukan nilai parameter Artist dan SongTitle:
 - a. Pilih Parameter permintaan opsional.
 - b. Pilih Tambahkan parameter baru.
 - c. Pilih jenis atribut string dan nilai Acme Band.
 - d. Ulangi langkah b dan c, lalu pilih jenis string dan nilai PartiQL Rocks.
4. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih bahasa yang Anda inginkan dari tab yang ditampilkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

5. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.

AWS CLI

1. Buat item dalam tabel Music menggunakan pernyataan INSERT PartiQL.

```
aws dynamodb execute-statement --statement "INSERT INTO Music \
    VALUE \
    {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

2. Ambil item dari tabel Musik menggunakan pernyataan SELECT PartiQL.

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

3. Perbarui item dalam tabel Music menggunakan pernyataan UPDATE PartiQL.

```
aws dynamodb execute-statement --statement "UPDATE Music \
    SET AwardsWon=1 \
    SET AwardDetail={'Grammys':[2020,
    2018]} \
    WHERE Artist='Acme Band' AND
    SongTitle='PartiQL Rocks'"
```

Tambahkan nilai daftar untuk item dalam tabel Music.

```
aws dynamodb execute-statement --statement "UPDATE Music \
                                         SET AwardDetail.Grammys
=list_append(AwardDetail.Grammys,[2016]) \
                                         WHERE Artist='Acme Band' AND
SongTitle='PartiQL Rocks'"
```

Hapus nilai daftar untuk item dalam tabel Music.

```
aws dynamodb execute-statement --statement "UPDATE Music \
                                         REMOVE AwardDetail.Grammys[2] \
                                         WHERE Artist='Acme Band' AND
SongTitle='PartiQL Rocks'"
```

Tambahkan anggota peta baru untuk item dalam tabel Music.

```
aws dynamodb execute-statement --statement "UPDATE Music \
                                         SET AwardDetail.BillBoard=[2020] \
                                         WHERE Artist='Acme Band' AND
SongTitle='PartiQL Rocks'"
```

Tambahkan atribut kumpulan string baru untuk item dalam tabel Music.

```
aws dynamodb execute-statement --statement "UPDATE Music \
                                         SET BandMembers =<<'member1',
'member2'>> \
                                         WHERE Artist='Acme Band' AND
SongTitle='PartiQL Rocks'"
```

Perbarui atribut kumpulan string baru untuk item dalam tabel Music.

```
aws dynamodb execute-statement --statement "UPDATE Music \
                                         SET BandMembers
=set_add(BandMembers, <<'newmember'>>) \
                                         WHERE Artist='Acme Band' AND
SongTitle='PartiQL Rocks'"
```

4. Hapus item dari tabel Music menggunakan pernyataan DELETE PartiQL.

```
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 com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.ConditionalCheckFailedException;
import com.amazonaws.services.dynamodbv2.model.ExecuteStatementRequest;
import com.amazonaws.services.dynamodbv2.model.ExecuteStatementResult;
import com.amazonaws.services.dynamodbv2.model.InternalServerErrorException;
import
    com.amazonaws.services.dynamodbv2.model.ItemCollectionSizeLimitExceededException;
import
    com.amazonaws.services.dynamodbv2.model.ProvisionedThroughputExceededException;
import com.amazonaws.services.dynamodbv2.model.RequestLimitExceededException;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import com.amazonaws.services.dynamodbv2.model.TransactionConflictException;

public class DynamoDBPartiQLGettingStarted {

    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.getMessage());
    } 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.getMessage());
    } 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());
    }
}
}

```

Tipe data PartiQL untuk DynamoDB

Tabel berikut mencantumkan jenis data yang dapat Anda gunakan dengan PartiQL untuk DynamoDB.

Tipe data DynamoDB	Representasi PartiQL	Catatan
Boolean	TRUE FALSE	Tidak peka terhadap huruf besar-kecil.
Binary	N/A	Hanya didukung melalui kode.
List	[value1, value2,...]	Tidak ada batasan pada tipe data yang dapat disimpan dalam elemen daftar, dan elemen dalam elemen daftar tidak harus bertipe sama.
Map	{ 'name' : value }	Tidak ada batasan pada tipe data yang dapat disimpan dalam elemen peta, dan elemen dalam peta tidak harus bertipe sama.
Null	NULL	Tidak peka terhadap huruf besar-kecil.
Number	1, 1.0, 1e0	Angka bisa positif, negatif, atau nol. Angka dapat memiliki hingga 38 digit presisi.

Tipe data DynamoDB	Representasi PartiQL	Catatan
Number Set	<<number1, number2>>	Elemen dalam sejumlah set harus dari bertipe Angka.
String Set	<<'string1', 'string2'>>	Elemen dalam suatu set string harus bertipe String.
String	'nilai string'	Tanda kutip tunggal harus digunakan untuk menentukan nilai String.

Contoh-contoh

Pernyataan berikut menunjukkan cara memasukkan tipe data berikut: String, Number, Map, List, Number Set, dan String Set.

```
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'>>
}
```

Pernyataan berikut menunjukkan cara menyisipkan elemen baru ke dalam tipe Map, List, Number Set, and String Set serta mengubah nilai tipe Number.

```
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'
```

Pernyataan berikut menunjukkan cara menghapus elemen dari tipe Map, List, Number Set, and String Set serta mengubah nilai tipe Number.

```
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'
```

Untuk informasi selengkapnya, lihat [tipe data DynamoDB](#).

Pernyataan PartiQL untuk DynamoDB

Amazon DynamoDB mendukung pernyataan PartiQL berikut.

Note

DynamoDB tidak mendukung semua pernyataan PartiQL. Referensi ini menyediakan sintaks dasar dan penggunaan contoh pernyataan PartiQL yang Anda jalankan secara manual menggunakan AWS CLI atau API.

Bahasa manipulasi data (DML) adalah kumpulan pernyataan PartiQL yang Anda gunakan untuk mengelola data dalam tabel DynamoDB. Anda menggunakan pernyataan DML untuk menambah, mengubah, atau menghapus data dalam sebuah tabel.

Berikut DML dan kueri bahasa pernyataan yang didukung:

- [Pernyataan pemilihan PartiQL untuk DynamoDB](#)
- [Pernyataan pembaruan PartiQL untuk DynamoDB](#)
- [Pernyataan sisipkan PartiQL untuk DynamoDB](#)
- [Pernyataan hapus PartiQL untuk DynamoDB](#)

[Melakukan transaksi dengan PartiQL untuk DynamoDB](#) and [Menjalankan operasi batch dengan PartiQL untuk DynamoDB](#) juga didukung oleh PartiQL untuk DynamoDB.

Pernyataan pemilihan PartiQL untuk DynamoDB

Gunakan pernyataan SELECT untuk mengambil data dari tabel di Amazon DynamoDB.

Menggunakan SELECT pernyataan dapat menghasilkan pemindaian tabel lengkap jika kondisi kesetaraan atau IN dengan kunci partisi tidak disediakan dalam klausa WHERE. Operasi pemindaian

memeriksa setiap item untuk nilai yang diminta dan dapat menggunakan throughput yang disediakan untuk indeks atau tabel besar dalam satu operasi.

Jika ingin menghindari pemindaian tabel secara lengkap di PartiQL, Anda dapat:

- Menulis pernyataan SELECT Anda agar tidak memindai tabel secara lengkap dengan memastikan [ketentuan klausul WHERE](#) Anda dikonfigurasi dengan semestinya.
- Menonaktifkan pemindaian tabel lengkap menggunakan kebijakan IAM yang ditentukan di [Contoh: Izinkan pernyataan pilih dan tolak pernyataan pemindaian tabel secara penuh di PartiQL untuk DynamoDB](#), dalam panduan developer DynamoDB.

Untuk informasi selengkapnya, lihat [Praktik terbaik untuk Mengkueri dan memindai data](#), dalam panduan developer DynamoDB.

Topik

- [Sintaksis](#)
- [Parameter-parameter](#)
- [Contoh-contoh](#)

Sintaksis

```
SELECT expression [, ...]  
FROM table[.index]  
[ WHERE condition ] [ [ORDER BY key [DESC|ASC] , ...]
```

Parameter-parameter

ekspresi

(Diperlukan) Proyeksi yang terbentuk dari wildcard * atau daftar proyeksi dari satu nama atribut atau lebih atau jalur dokumen dari set hasil. Ekspresi dapat terdiri dari panggilan ke [Gunakan fungsi PartiQL dengan amazon DynamoDB](#) atau bidang yang diubah oleh [PartiQL aritmatika, perbandingan, dan operator logis untuk DynamoDB](#).

tabel

(Diperlukan) Nama tabel untuk kueri.

indeks

(Opsional) Nama indeks untuk kueri.

Note

Anda harus menambahkan tanda kutip ganda ke nama tabel dan nama indeks saat mengkueri indeks.

```
SELECT *  
FROM "TableName"."IndexName"
```

ketentuan

(Opsional) Kriteria seleksi untuk kueri.

Important

Untuk memastikan bahwa pernyataan SELECT tidak menghasilkan pemindaian tabel secara lengkap, kondisi klausul WHERE harus menentukan kunci partisi. Gunakan kesetaraan atau operator IN.

Misalnya, jika Anda memiliki tabel `Orders` dengan kunci partisi `OrderID` dan atribut non-kunci lainnya, termasuk `Address`, pernyataan berikut tidak akan menghasilkan pemindaian tabel secara lengkap:

```
SELECT *  
FROM "Orders"  
WHERE OrderID = 100
```

```
SELECT *  
FROM "Orders"  
WHERE OrderID = 100 and Address='some address'
```

```
SELECT *  
FROM "Orders"  
WHERE OrderID = 100 or pk = 200
```

```
SELECT *  
FROM "Orders"
```

```
WHERE OrderID IN [100, 300, 234]
```

Namun, pernyataan SELECT berikut akan menghasilkan pemindaian tabel secara lengkap:

```
SELECT *
FROM "Orders"
WHERE OrderID > 1

SELECT *
FROM "Orders"
WHERE Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 OR Address='some address'
```

kunci

(Opsional) Kunci hash atau kunci urutan untuk digunakan dalam mengurutkan hasil yang dikembalikan. Urutan defaultnya adalah naik (ASC) tentukan DESC jika Anda ingin hasil dikembalikan dalam urutan turun.

Note

Jika Anda menghapus klausul WHERE, semua item dalam tabel akan diambil.

Contoh-contoh

Kueri berikut mengembalikan satu item, jika ada, dari tabel `Orders` dengan menentukan kunci partisi, `OrderID`, dan menggunakan operator kesetaraan.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1
```

Kueri berikut mengembalikan semua item dalam tabel `Orders` yang memiliki kunci partisi spesifik, `OrderID`, nilai menggunakan operator `OR`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1 OR OrderID = 2
```

Kueri berikut mengembalikan semua item dalam tabel `Orders` yang memiliki kunci partisi spesifik, `OrderID`, nilai menggunakan operator `IN`. Hasil yang dikembalikan dalam urutan turun, berdasarkan nilai atribut kunci `OrderID`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID IN [1, 2, 3] ORDER BY OrderID DESC
```

Kueri berikut menunjukkan pemindaian tabel secara lengkap yang mengembalikan semua item dari tabel `Orders` yang memiliki `Total` lebih dari 500, dengan `Total` adalah atribut non-kunci.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total > 500
```

Kueri berikut menunjukkan pemindaian tabel secara lengkap yang mengembalikan semua item dari tabel `Orders` di dalam rentang urutan `Total` tertentu, menggunakan operator `IN` dan atribut non-kunci `Total`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total IN [500, 600]
```

Kueri berikut menunjukkan pemindaian tabel secara lengkap yang mengembalikan semua item dari tabel `Orders` di dalam rentang urutan `Total` tertentu, menggunakan operator `BETWEEN` dan atribut non-kunci `Total`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total BETWEEN 500 AND 600
```


Kueri berikut mengembalikan tanggal pertama perangkat firestick digunakan untuk menonton dengan menentukan kunci partisi `CustomerID` dan kunci urutan `MovieID` dalam kondisi klausul `WHERE` dan menggunakan jalur dokumen dalam klausul `SELECT`.

```
SELECT Devices.FireStick.DateWatched[0]
FROM WatchList
WHERE CustomerID= 'C1' AND MovieID= 'M1'
```

Kueri berikut menunjukkan pemindaian tabel secara penuh yang mengembalikan daftar item tempat perangkat firestick pertama kali digunakan setelah 24/12/19 menggunakan jalur dokumen dalam kondisi klausul `WHERE`.

```
SELECT Devices
FROM WatchList
WHERE Devices.FireStick.DateWatched[0] >= '12/24/19'
```

Pernyataan pembaruan PartiQL untuk DynamoDB

Gunakan pernyataan `UPDATE` untuk mengubah nilai dari satu atribut atau lebih dalam item dalam tabel Amazon DynamoDB.

Note

Anda hanya dapat memperbarui satu item pada satu waktu; Anda tidak dapat mengeluarkan pernyataan DynamoDB PartiQL tunggal yang memperbarui beberapa item. Untuk informasi tentang memperbarui beberapa item, lihat [Melakukan transaksi dengan PartiQL untuk DynamoDB](#) atau [Menjalankan operasi batch dengan PartiQL untuk DynamoDB](#).

Topik

- [Sintaksis](#)
- [Parameter-parameter](#)
- [Nilai yang dikembalikan](#)
- [Contoh-contoh](#)

Sintaksis

```
UPDATE table
```

```
[SET | REMOVE] path [= data] [...]  
WHERE condition [RETURNING returnvalues]  
<returnvalues> ::= [ALL OLD | MODIFIED OLD | ALL NEW | MODIFIED NEW] *
```

Parameter-parameter

tabel

(Diperlukan) Tabel yang berisi data yang akan diubah.

path

(Diperlukan) Nama atribut atau jalur dokumen yang akan dibuat atau diubah.

data

(Diperlukan) Nilai atribut atau hasil operasi.

Operasi yang didukung untuk digunakan bersama SET:

- LIST_APPEND: menambahkan nilai ke jenis daftar.
- SET_ADD: menambahkan nilai ke set angka atau string.
- SET_DELETE: menghapus nilai dari set angka atau string.

ketentuan

(Diperlukan) Kriteria pemilihan untuk item yang akan diubah. Syarat ini harus diselesaikan untuk satu nilai kunci primer.

returnvalues

(Opsional) Gunakan `returnvalues` jika Anda ingin mendapatkan atribut item seperti yang muncul sebelum atau setelah diperbarui. Nilai yang valid adalah:

- ALL OLD *- Mengembalikan semua atribut item, saat atribut item tersebut muncul sebelum operasi pembaruan.
- MODIFIED OLD *- Mengembalikan hanya atribut yang diperbarui, saat atribut item tersebut muncul sebelum operasi pembaruan.
- ALL NEW *- Mengembalikan semua atribut item, seperti yang muncul setelah operasi pembaruan.
- MODIFIED NEW *- Hanya mengembalikan atribut yang diperbarui, seperti yang muncul setelah operasi `UpdateItem`.

Nilai yang dikembalikan

Pernyataan ini tidak mengembalikan nilai kecuali parameter `returnValues` ditentukan.

Note

Jika klausul `WHERE` pernyataan `UPDATE` tidak dinilai `true` untuk item apa pun dalam tabel DynamoDB, `ConditionalCheckFailedException` akan dikembalikan.

Contoh-contoh

Memperbarui nilai atribut dalam item yang sudah ada. Jika atribut tersebut tidak ada, atribut tersebut akan dibuat.

Kueri berikut memperbarui item dalam tabel "Music" dengan menambahkan atribut dari nomor jenis (`AwardsWon`) dan atribut dari jenis peta (`AwardDetail`).

```
UPDATE "Music"  
SET AwardsWon=1  
SET AwardDetail={'Grammys':[2020, 2018]}  
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Anda dapat menambahkan `RETURNING ALL OLD *` untuk mengembalikan atribut seperti yang muncul sebelum operasi `Update`.

```
UPDATE "Music"  
SET AwardsWon=1  
SET AwardDetail={'Grammys':[2020, 2018]}  
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'  
RETURNING ALL OLD *
```

Ini akan menghasilkan hal berikut:

```
{  
  "Items": [  
    {  
      "Artist": {  
        "S": "Acme Band"  
      },  
      "SongTitle": {  
        "S": "PartiQL Rocks"  
      }  
    }  
  ]  
}
```

```

    }
  }
]
}

```

Anda dapat menambahkan RETURNING ALL NEW * untuk mengembalikan atribut seperti yang muncul setelah operasi Update e.

```

UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL NEW *

```

Ini akan menghasilkan hal berikut:

```

{
  "Items": [
    {
      "AwardDetail": {
        "M": {
          "Grammys": {
            "L": [
              {
                "N": "2020"
              },
              {
                "N": "2018"
              }
            ]
          }
        }
      },
      "AwardsWon": {
        "N": "1"
      }
    }
  ]
}

```

Kueri berikut memperbarui item dalam tabel "Music" dengan menambahkan ke daftar AwardDetail.Grammys.

```
UPDATE "Music"  
SET AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])  
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Kueri berikut memperbarui item dalam tabel "Music" dengan menghapus dari daftar AwardDetail.Grammys.

```
UPDATE "Music"  
REMOVE AwardDetail.Grammys[2]  
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Kueri berikut memperbarui item dalam tabel "Music" dengan menambahkan Billboard ke peta AwardDetail.

```
UPDATE "Music"  
SET AwardDetail.BillBoard=[2020]  
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Kueri berikut memperbarui item dalam tabel "Music" dengan menambahkan atribut set string BandMembers.

```
UPDATE "Music"  
SET BandMembers =<<'member1', 'member2'>>  
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Kueri berikut memperbarui item dalam tabel "Music" dengan menambahkan newbandmember ke set string BandMembers.

```
UPDATE "Music"  
SET BandMembers =set_add(BandMembers, <<'newbandmember'>>  
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Pernyataan hapus PartiQL untuk DynamoDB

Gunakan pernyataan DELETE untuk menghapus item yang sudah ada dari tabel Amazon DynamoDB Anda.

Note

Anda hanya dapat menghapus satu item dalam satu waktu. Anda tidak dapat mengeluarkan pernyataan DynamoDB PartiQL tunggal yang menghapus beberapa item. Untuk informasi tentang menghapus beberapa item, lihat [Melakukan transaksi dengan PartiQL untuk DynamoDB](#) atau [Menjalankan operasi batch dengan PartiQL untuk DynamoDB](#).

Topik

- [Sintaksis](#)
- [Parameter-parameter](#)
- [Nilai yang dikembalikan](#)
- [Contoh-contoh](#)

Sintaksis

```
DELETE FROM table  
WHERE condition [RETURNING returnvalues]  
<returnvalues> ::= ALL OLD *
```

Parameter-parameter

table

(Diperlukan) Tabel DynamoDB yang berisi item yang akan dihapus.

ketentuan

(Diperlukan) Kriteria seleksi untuk item yang akan dihapus; syarat ini harus diselesaikan untuk nilai kunci primer tunggal.

returnvalues

(Opsional) Gunakan `returnvalues` jika Anda ingin mendapatkan atribut item seperti yang muncul sebelum atribut item tersebut dihapus. Nilai yang valid adalah:

- ALL OLD *- Konten dari item lama akan dikembalikan.

Nilai yang dikembalikan

Pernyataan ini tidak mengembalikan nilai kecuali parameter `returnvalues` ditentukan.

Note

Jika tabel DynamoDB tidak memiliki item dengan kunci primer yang sama seperti yang item yang DELETE-nya dikeluarkan, SUCCESS dikembalikan dengan 0 item dihapus. Jika tabel memiliki item dengan kunci primer yang sama, tetapi syarat dalam klausul WHERE pernyataan dari DELETE bernilai false, `ConditionalCheckFailedException` akan dikembalikan.

Contoh-contoh

Kueri berikut akan menghapus item dalam tabel "Music".

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
```

Anda dapat menambahkan parameter `RETURNING ALL OLD *` untuk mengembalikan data yang telah dihapus.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'  
RETURNING ALL OLD *
```

Pernyataan Delete sekarang mengembalikan yang berikut:

```
{  
  "Items": [  
    {  
      "Artist": {  
        "S": "Acme Band"  
      },  
      "SongTitle": {  
        "S": "PartiQL Rocks"  
      }  
    }  
  ]  
}
```

Pernyataan sisipkan PartiQL untuk DynamoDB

Gunakan pernyataan INSERT untuk menambahkan item ke tabel di Amazon DynamoDB.

Note

Anda hanya dapat menyisipkan satu item pada satu waktu; Anda tidak dapat mengeluarkan pernyataan DynamoDB PartiQL tunggal yang menyisipkan beberapa item. Untuk informasi tentang cara menyisipkan beberapa item, lihat [Melakukan transaksi dengan PartiQL untuk DynamoDB](#) atau [Menjalankan operasi batch dengan PartiQL untuk DynamoDB](#).

Topik

- [Sintaks](#)
- [Parameter](#)
- [Nilai yang dikembalikan](#)
- [Contoh-contoh](#)

Sintaks

Sisipkan satu item.

```
INSERT INTO table VALUE item
```

Parameter

table

(Diperlukan) Tabel tempat Anda ingin menyisipkan data. Tabel harus sudah ada.

item

(Diperlukan) Item DynamoDB yang valid diwakili sebagai [tupel PartiQL](#). Anda harus menentukan hanya satu item dan setiap nama atribut dalam item adalah peka huruf besar-kecil dan dapat dilambangkan dengan tanda petik tunggal (' . . . ') di PartiQL.

Nilai string juga dilambangkan dengan tanda petik tunggal (' . . . ') di PartiQL.

Nilai yang dikembalikan

Pernyataan ini tidak mengembalikan nilai apa pun.

Note

Jika tabel DynamoDB sudah memiliki item dengan kunci primer yang sama sebagai kunci primer dari item yang sedang disisipkan, `DuplicateItemException` dikembalikan.

Contoh-contoh

```
INSERT INTO "Music" value {'Artist' : 'Acme Band', 'SongTitle' : 'PartiQL Rocks'}
```

Gunakan fungsi PartiQL dengan amazon DynamoDB

PartiQL di Amazon DynamoDB mendukung varian bawaan berikut dari fungsi standar SQL.

Note

Fungsi SQL yang tidak termasuk dalam daftar ini saat ini tidak didukung di DynamoDB.

Fungsi agregat

- [Menggunakan fungsi SIZE dengan PartiQL untuk Amazon DynamoDB](#)

Fungsi kondisional

- [Menggunakan fungsi EXISTS dengan PartiQL untuk DynamoDB](#)
- [Menggunakan fungsi ATTRIBUTE_TYPE dengan PartiQL untuk DynamoDB](#)
- [Menggunakan fungsi BEGINH dengan PartiQL untuk DynamoDB](#)
- [Menggunakan fungsi CONTAINS dengan PartiQL untuk DynamoDB](#)
- [Menggunakan fungsi MISSING dengan PartiQL untuk DynamoDB](#)

Menggunakan fungsi EXISTS dengan PartiQL untuk DynamoDB

Anda dapat menggunakan EXISTS untuk melakukan fungsi yang sama seperti `ConditionCheck` tidak di [TransactWriteItems](#) API. Fungsi EXISTS hanya dapat digunakan dalam transaksi.

Dengan nilai, mengembalikan TRUE jika nilai adalah pengumpulan non-kosong. Sebaliknya, mengembalikan FALSE.

Note

Fungsi ini hanya dapat digunakan dalam operasi transaksional.

Sintaksis

```
EXISTS ( statement )
```

Pendapat

pernyataan

(Diperlukan) Pernyataan SELECT di mana fungsi melakukan evaluasi.

Note

Pernyataan SELECT harus menentukan kunci utama penuh dan satu kondisi lainnya.

Jenis pengembalian

bool

Contoh

```
EXISTS(  
  SELECT * FROM "Music"  
  WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks')
```

Menggunakan fungsi BEGINH dengan PartiQL untuk DynamoDB

Mengembalikan TRUE jika atribut yang ditentukan dimulai dengan substring tertentu.

Sintaksis

```
begins_with(path, value )
```

Pendapat

jalur

(Diperlukan) Nama atribut atau jalur dokumen yang akan digunakan.

nilai

(Diperlukan) String untuk mencari.

Jenis pengembalian

bool

Contoh

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND begins_with("Address", '7834 24th')
```

Menggunakan fungsi MISSING dengan PartiQL untuk DynamoDB

Mengembalikan TRUE jika item tidak berisi atribut yang ditentukan. Hanya operator kesetaraan dan ketidaksetaraan dapat digunakan dengan fungsi ini.

Sintaksis

```
attributename IS | IS NOT MISSING
```

Pendapat

attributename

(Diperlukan) Nama atribut untuk mencari.

Jenis pengembalian

bool

Contoh

```
SELECT * FROM Music WHERE "Awards" is MISSING
```

Menggunakan fungsi ATTRIBUTE_TYPE dengan PartiQL untuk DynamoDB

Mengembalikan TRUE jika atribut pada jalur yang ditentukan adalah tipe data tertentu.

Sintaksis

```
attribute_type( attributename, type )
```

Pendapat

attributename

(Diperlukan) Nama atribut yang digunakan.

tipe

(Diperlukan) Tipe atribut untuk memeriksa. Untuk daftar nilai yang valid, lihat [attribute_type](#) DynamoDB.

Jenis pengembalian

bool

Contoh

```
SELECT * FROM "Music" WHERE attribute_type("Artist", 'S')
```

Menggunakan fungsi CONTAINS dengan PartiQL untuk DynamoDB

Mengembalikan TRUE jika atribut yang ditentukan oleh jalur tersebut adalah salah satu dari hal berikut:

- String yang berisi substring tertentu.

- Kumpulan yang berisi elemen tertentu dalam kumpulan tersebut.

Untuk informasi selengkapnya, lihat fungsi [berisi](#) DynamoDB.

Sintaksis

```
contains( path, substring )
```

Pendapat

jalur

(Diperlukan) Nama atribut atau jalur dokumen yang akan digunakan.

substring

(Diperlukan) Substring atribut atau anggota yang ditetapkan untuk memeriksa. Untuk informasi selengkapnya, lihat fungsi [berisi](#) DynamoDB.

Jenis pengembalian

bool

Contoh

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND contains("Address", 'Kirkland')
```

Menggunakan fungsi SIZE dengan PartiQL untuk Amazon DynamoDB

Mengembalikan angka yang mewakili ukuran atribut dalam byte. Berikut ini adalah jenis data yang valid untuk digunakan dengan ukuran. Untuk informasi selengkapnya, lihat fungsi [ukuran](#) DynamoDB.

Sintaksis

```
size( path )
```

Pendapat

jalur

(Diperlukan) Nama atribut atau jalur dokumen.

Untuk jenis yang didukung, lihat fungsi [ukuran](#) DynamoDB.

Jenis pengembalian

int

Contoh

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND size("Image") >300
```

PartiQL aritmatika, perbandingan, dan operator logis untuk DynamoDB

PartiQL di Amazon DynamoDB mendukung [Operator standar SQL](#) berikut.

Note

Setiap operator SQL yang tidak termasuk dalam daftar ini saat ini tidak didukung di DynamoDB.

Operator aritmatika

Operator	Deskripsi
+	Tambahkan
-	Kurangi

Operator perbandingan

Operator	Deskripsi
=	Sama dengan
<>	Tidak Sama dengan
!=	Tidak Sama dengan
>	Lebih besar dari

Operator	Deskripsi
<	Kurang dari
>=	Lebih besar dari atau sama dengan
<=	Kurang dari atau sama dengan

Operator logika

Operator	Deskripsi
AND	TRUE jika semua kondisi dipisahkan oleh AND adalah TRUE
BETWEEN	TRUE jika operand berada dalam rentang perbandingan
IN	TRUE jika operand sama dengan salah satu dari daftar ekspresi (di max 50 nilai atribut atau di max 100 non-key nilai atribut)
IS	TRUE jika operand adalah diberikan, tipe data PartiQL, termasuk NULL atau MISSING
NOT	Membalikkan nilai ekspresi Boolean yang diberikan
OR	TRUE jika semua syarat dipisahkan oleh OR adalah TRUE

Melakukan transaksi dengan PartiQL untuk DynamoDB

Bagian ini menjelaskan cara menggunakan transaksi dengan PartiQL untuk DynamoDB. Transaksi PartiQL dibatasi hingga 100 total pernyataan (tindakan).

Untuk informasi lebih lanjut tentang transaksi DynamoDB, lihat [Mengelola alur kerja kompleks dengan transaksi DynamoDB](#).

Note

Seluruh transaksi harus terdiri dari pernyataan baca atau pernyataan tulis. Anda tidak dapat menggabungkan keduanya dalam satu transaksi. Fungsi EXISTS merupakan pengecualian. Anda dapat menggunakannya untuk memeriksa kondisi atribut tertentu dari item dengan cara yang mirip ConditionCheck dengan operasi [TransactWriteItemsAPI](#).

Topik

- [Sintaksis](#)
- [Parameter-parameter](#)
- [Nilai yang dikembalikan](#)
- [Contoh-contoh](#)

Sintaksis

```
[
  {
    "Statement": " statement ",
    "Parameters": [
      {
        " parametertype " : " parametervalue "
      }, ... ]
    } , ...
]
```

Parameter-parameter

pernyataan

(Diperlukan) Pernyataan yang didukung PartiQL untuk DynamoDB.

Note

Seluruh transaksi harus terdiri dari pernyataan baca atau pernyataan tulis. Anda tidak dapat menggabungkan keduanya dalam satu transaksi.

parametertype

(Opsional) Tipe DynamoDB, jika parameter digunakan saat menentukan pernyataan PartiQL.

parametervalue

(Opsional) Nilai parameter jika parameter digunakan saat menentukan pernyataan PartiQL.

Nilai yang dikembalikan

Pernyataan ini tidak mengembalikan nilai apa pun untuk operasi Tulis (INSERT, UPDATE, atau DELETE). Namun, ia mengembalikan nilai yang berbeda untuk operasi Baca (SELECT) berdasarkan kondisi yang ditentukan dalam klausul WHERE.

Note

Jika salah satu operasi INSERT, UPDATE, atau DELETE singleton mengembalikan kesalahan, transaksi dibatalkan dengan pengecualian `TransactionCanceledException`, dan kode alasan pembatalan termasuk kesalahan dari operasi singleton individual.

Contoh-contoh

Contoh berikut menjalankan beberapa pernyataan sebagai sebuah transaksi.

AWS CLI

1. Simpan kode JSON berikut ke file bernama `partiql.json`.

```
[
  {
    "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. Jalankan perintah berikut di prompt perintah.

```
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.getMessage());
    } 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.getMessage());
    } 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.getMessage());
    } catch (Exception e) {
        handleCommonErrors(e);
    }
}
```

```
    }  
  }  
  
  private static void handleCommonErrors(Exception exception) {  
    try {  
      throw exception;  
    } catch (InternalServerErrorException ise) {  
      System.out.println("Internal Server Error, generally safe to retry with  
exponential back-off. Error: " + ise.getMessage());  
    } catch (RequestLimitExceededException rlee) {  
      System.out.println("Throughput exceeds the current throughput limit for  
your account, increase account level throughput before " +  
      "retrying. Error: " + rlee.getMessage());  
    } 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.getMessage());  
    } catch (ResourceNotFoundException rnfe) {  
      System.out.println("One of the tables was not found, verify table exists  
before retrying. Error: " + rnfe.getMessage());  
    } 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.getMessage());  
    } 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());  
    }  
  }  
}
```

Contoh berikut menunjukkan nilai pengembalian yang berbeda ketika DynamoDB membaca item dengan kondisi berbeda yang ditentukan dalam klausul WHERE.

AWS CLI

1. Simpan kode JSON berikut ke file bernama `partiql.json`.

```
[
  // 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. perintah berikut di prompt perintah.

```
aws dynamodb execute-transaction --transact-statements file://partiql.json
```

3. Respons berikut dikembalikan:

```
{
  "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
    {}
]
}
```

Menjalankan operasi batch dengan PartiQL untuk DynamoDB

Bagian ini menjelaskan cara menggunakan pernyataan batch dengan PartiQL untuk DynamoDB.

Note

- Keseluruhan batch harus terdiri dari pernyataan baca atau pernyataan tulis; Anda tidak dapat mencampur keduanya dalam satu batch.
- `BatchExecuteStatement` dan `BatchWriteItem` dapat melakukan tidak lebih dari 25 pernyataan per batch.

Topik

- [Sintaksis](#)
- [Parameter-parameter](#)
- [Contoh-contoh](#)

Sintaksis

```
[
  {
    "Statement": " statement ",
    "Parameters": [
      {
        " parametertype " : " parametervalue "
      }, ... ]
    } , ...
]
```

Parameter-parameter

pernyataan

(Diperlukan) Pernyataan yang didukung PartiQL untuk DynamoDB.

Note

- Keseluruhan batch harus terdiri dari pernyataan baca atau pernyataan tulis; Anda tidak dapat mencampur keduanya dalam satu batch.
- `BatchExecuteStatement` dan `BatchWriteItem` dapat melakukan tidak lebih dari 25 pernyataan per batch.

parametertype

(Opsional) Tipe DynamoDB, jika parameter digunakan saat menentukan pernyataan PartiQL.

parametervalue

(Opsional) Nilai parameter jika parameter digunakan saat menentukan pernyataan PartiQL.

Contoh-contoh

AWS CLI

1. Simpan json berikut ke file bernama `partiql.json`

```
[
  {
    "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. Jalankan perintah berikut di prompt perintah.

```
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 ise) {
        System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + ise.getMessage());
    } catch (RequestLimitExceededException rlee) {
        System.out.println("Throughput exceeds the current throughput limit for
your account, increase account level throughput before " +
            "retrying. Error: " + rlee.getMessage());
    } 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.getMessage());
    }
}
```

```
    } catch (ResourceNotFoundException rnfe) {
        System.out.println("One of the tables was not found, verify table exists
before retrying. Error: " + rnfe.getMessage());
    } 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.getMessage());
    } 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());
    }
}
}
```

Kebijakan keamanan IAM dengan PartiQL untuk DynamoDB

Izin berikut diperlukan:

- Untuk membaca item menggunakan PartiQL untuk DynamoDB, Anda harus memiliki izin `dynamodb:PartiQLSelect` pada tabel atau indeks.
- Untuk menyisipkan item menggunakan PartiQL untuk DynamoDB, Anda harus memiliki izin `dynamodb:PartiQLInsert` pada tabel atau indeks.
- Untuk memperbaiki item menggunakan PartiQL untuk DynamoDB, Anda harus memiliki izin `dynamodb:PartiQLUpdate` pada tabel atau indeks.
- Untuk menghapus item menggunakan PartiQL untuk DynamoDB, Anda harus memiliki izin `dynamodb:PartiQLDelete` pada tabel atau indeks.

Contoh: Izinkan semua pernyataan PartiQL untuk DynamoDB (Pilih/Sisipkan/Perbarui/Hapus) pada tabel

Kebijakan IAM berikut memberikan izin untuk menjalankan semua pernyataan PartiQL untuk DynamoDB pada tabel.

```
{
  "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"
      ]
    }
  ]
}
```

Contoh: Izinkan pernyataan pilih PartiQL untuk DynamoDB pada tabel

Kebijakan IAM berikut memberikan izin untuk menjalankan pernyataan select pada tabel tertentu.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb: PartiQLSelect"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
      ]
    }
  ]
}
```

Contoh: Izinkan pernyataan sisipkan PartiQL untuk DynamoDB pada indeks

Kebijakan IAM berikut memberikan izin untuk menjalankan pernyataan insert pada indeks tertentu.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PartiQLInsert"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/Music/index/index1"
      ]
    }
  ]
}
```

Contoh: Izinkan pernyataan transaksional PartiQL untuk DynamoDB pada tabel

Kebijakan IAM berikut memberikan izin hanya untuk menjalankan pernyataan transaksional pada tabel tertentu.

```
{
  "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"
          ]
        }
      }
    }
  ]
}
```

```

    }
  }
}
]
}

```

Contoh: Izinkan pernyataan transaksional baca dan tulis non-transaksional PartiQL untuk DynamoDB serta blokir pernyataan transaksional baca dan tulis transaksional baca dan tulis transaksional PartiQL pada tabel.

Kebijakan IAM berikut memberikan izin guna menjalankan operasi baca dan tulis non-transaksional PartiQL untuk DynamoDB sambil memblokir operasi baca dan tulis transaksional PartiQL untuk DynamoDB.

```

{
  "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"
    ]
  }
]
}
```

Contoh: Izinkan pernyataan pilih dan tolak pernyataan pemindaian tabel secara penuh di PartiQL untuk DynamoDB

Kebijakan IAM berikut memberikan izin untuk menjalankan pernyataan `select` pada tabel tertentu sambil memblokir pernyataan `select` yang menghasilkan pemindaian tabel secara penuh.

```
{
  "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"
      ]
    }
  ]
}
```

}

Meningkatkan akses data dengan indeks sekunder

Topik

- [Menggunakan Indeks Sekunder Global di DynamoDB](#)
- [Indeks Sekunder Lokal](#)

Amazon DynamoDB menyediakan akses cepat ke item-item dalam sebuah tabel dengan menentukan nilai-nilai kunci primer. Namun, banyak aplikasi mungkin mendapatkan keuntungan dengan memiliki satu atau lebih kunci sekunder (atau alternatif) yang tersedia, untuk memungkinkan akses yang efisien ke data dengan atribut selain kunci primer. Untuk mengatasi hal ini, Anda dapat membuat satu atau lebih indeks sekunder pada tabel dan mengeluarkan Query atau Scan permintaan terhadap indeks ini.

Sebuah Indeks sekunder adalah struktur data yang berisi subset atribut dari tabel, bersama dengan kunci alternatif untuk mendukung operasi Query. Anda dapat mengambil data dari indeks menggunakan sebuah Query, dengan cara yang sama seperti saat Anda menggunakan Query dengan tabel. Sebuah tabel dapat memiliki beberapa indeks sekunder, yang memberikan aplikasi Anda akses ke banyak pola kueri yang berbeda.

Note

Anda juga dapat Scan sebuah indeks, dengan cara yang sama seperti Anda Scan sebuah tabel.

Setiap indeks sekunder dikaitkan hanya dengan satu tabel, dari mana indeks memperoleh datanya. Ini disebut tabel dasar untuk indeks. Saat Anda membuat indeks, Anda menentukan kunci alternatif untuk indeks (kunci partisi dan kunci pengurutan). Anda juga menentukan atribut yang ingin Anda diproyeksikan, atau disalin, dari tabel dasar ke dalam indeks. DynamoDB menyalin atribut ini ke dalam indeks, bersama dengan atribut kunci primer dari tabel dasar. Anda kemudian dapat melakukan kueri atau memindai indeks sama seperti Anda melakukan kueri atau memindai tabel.

Setiap indeks sekunder secara otomatis dikelola oleh DynamoDB. Ketika Anda menambahkan, memodifikasi, atau menghapus item dalam tabel dasar, setiap indeks pada tabel tersebut juga diperbarui untuk mencerminkan perubahan ini.

DynamoDB mendukung dua jenis indeks sekunder:

- [Indeks sekunder global](#) — Indeks dengan kunci partisi dan kunci pengurutan yang mungkin berbeda dari yang ada di tabel dasar. Indeks sekunder global dianggap "global" karena kueri pada indeks dapat menjangkau semua data di tabel dasar, di semua partisi. Indeks sekunder global disimpan dalam ruang partisinya sendiri jauh dari tabel dasar dan diskalakan secara terpisah dari tabel dasar.
- [Indeks sekunder lokal](#) — Indeks yang memiliki kunci partisi yang sama sebagai tabel dasar, tetapi dengan kunci pengurutan yang berbeda. Indeks sekunder lokal adalah "lokal" dalam arti bahwa setiap partisi indeks sekunder lokal dicakup ke partisi tabel dasar yang memiliki nilai kunci partisi yang sama.

Untuk perbandingan indeks sekunder global dan indeks sekunder lokal, lihat video ini.

[Membuat pilihan yang tepat antara GSI dan LSI](#)

Anda harus mempertimbangkan persyaratan aplikasi Anda saat menentukan jenis indeks yang akan digunakan. Tabel berikut menunjukkan perbedaan utama antara indeks sekunder global dan indeks sekunder lokal.

Karakteristik	Indeks sekunder global	Indeks sekunder lokal
Skema Kunci	Kunci primer dari indeks sekunder global dapat berupa sederhana (kunci partisi) atau gabungan (kunci partisi dan kunci pengurutan).	Kunci primer dari indeks sekunder lokal harus gabungan (kunci partisi dan kunci pengurutan).
Atribut Kunci	Kunci partisi indeks dan kunci pengurutan (jika ada) dapat berupa atribut tabel dasar apa pun dari tipe string, angka, atau biner.	Kunci partisi indeks adalah atribut yang sama dengan kunci partisi dari tabel dasar. Kunci pengurutan dapat berupa atribut tabel dasar apa pun dari tipe string, angka, atau biner.
Pembatasan Ukuran Per Nilai Kunci Partisi	Tidak ada batasan ukuran untuk indeks sekunder global.	Untuk setiap nilai kunci partisi, ukuran total semua item yang

Karakteristik	Indeks sekunder global	Indeks sekunder lokal
		diindeks harus 10 GB atau kurang.
Operasi Indeks Online	Indeks sekunder global dapat dibuat pada saat yang sama saat Anda membuat tabel. Anda juga dapat menambahkan indeks sekunder global baru ke tabel yang ada, atau menghapus indeks sekunder global yang ada. Untuk informasi selengkapnya, lihat Mengelola Indeks Sekunder Global .	Indeks sekunder lokal dibuat pada saat yang sama saat Anda membuat tabel. Anda tidak dapat menambahkan indeks sekunder lokal ke tabel yang sudah ada, Anda juga tidak dapat menghapus indeks sekunder lokal yang saat ini ada.
Query dan Partisi	Indeks sekunder global memungkinkan Anda melakukan kueri di seluruh tabel, di semua partisi.	Indeks sekunder lokal memungkinkan Anda melakukan kueri pada satu partisi, seperti yang ditentukan oleh nilai kunci partisi dalam kueri.
Baca Konsistensi	Kueri pada indeks sekunder global hanya mendukung konsistensi akhir.	Saat Anda melakukan kueri indeks sekunder lokal, Anda dapat memilih konsistensi akhir atau konsistensi kuat.

Karakteristik	Indeks sekunder global	Indeks sekunder lokal
Konsumsi Throughput yang Disediakan	Setiap indeks sekunder global memiliki pengaturan throughput yang disediakan sendiri untuk aktivitas baca dan tulis. Kueri atau pemindaian pada indeks sekunder global menggunakan unit kapasitas dari indeks, bukan dari tabel dasar. Hal yang sama berlaku untuk pembaruan indeks sekunder global karena penulisan tabel. Indeks sekunder global yang terkait dengan tabel global mengkonsumsi unit kapasitas tulis.	Kueri atau pemindaian pada indeks sekunder lokal menggunakan unit kapasitas baca dari tabel dasar. Ketika Anda menulis ke tabel indeks sekunder lokal juga diperbarui, dan pembaruan ini mengkonsumsi unit kapasitas tulis dari tabel dasar. Indeks sekunder lokal yang terkait dengan tabel global mengkonsumsi unit kapasitas tulis yang direplikasi.
Atribut yang Diproyeksikan	Dengan kueri atau pemindaian indeks sekunder global, Anda hanya dapat meminta atribut yang diproyeksikan ke dalam indeks. DynamoDB tidak mengambil atribut apapun dari tabel.	Jika Anda membuat kueri atau memindai indeks sekunder lokal, Anda dapat meminta atribut yang tidak diproyeksikan ke dalam indeks. DynamoDB secara otomatis mengambil atribut tersebut dari tabel.

Jika Anda ingin membuat lebih dari satu tabel dengan indeks sekunder, Anda harus melakukannya secara berurutan. Misalnya, Anda akan membuat tabel pertama dan menunggu agar itu menjadi ACTIVE, membuat tabel berikutnya dan menunggu agar itu menjadi ACTIVE, dan sebagainya. Jika Anda mencoba membuat lebih dari satu tabel secara bersamaan dengan indeks sekunder, DynamoDB mengembalikan `LimitExceededException`.

Setiap indeks sekunder menggunakan [kelas tabel](#) dan [mode kapasitas](#) yang sama dengan tabel dasar yang terkait dengannya. Untuk setiap indeks sekunder, Anda harus menentukan hal berikut:

- Jenis indeks yang akan dibuat – baik indeks sekunder global atau indeks sekunder lokal.
- Nama untuk indeks. Aturan penamaan untuk indeks sama dengan aturan untuk tabel, seperti yang tercantum dalam [Layanan, akun, dan tabel kuota di Amazon DynamoDB](#). Nama harus unik untuk tabel dasar yang terkait dengannya, tetapi Anda bisa menggunakan nama yang sama untuk indeks yang terkait dengan tabel dasar yang berbeda.
- Skema kunci untuk indeks. Setiap atribut dalam skema kunci indeks harus merupakan atribut tingkat atas dari tipe `String`, `Number`, atau `Binary`. Tipe data lain, termasuk dokumen dan set, tidak diperbolehkan. Persyaratan lain untuk skema kunci bergantung pada jenis indeks:
 - Untuk indeks sekunder global, kunci partisi dapat berupa atribut skalar apa pun dari tabel dasar. Kunci pengurutan adalah opsional, dan juga dapat berupa atribut skalar apa pun dari tabel dasar.
 - Untuk indeks sekunder lokal, kunci partisi harus sama dengan kunci partisi tabel dasar, dan kunci pengurutan harus berupa atribut tabel dasar bukan kunci.
- Atribut tambahan, jika ada, untuk diproyeksikan dari tabel dasar ke dalam indeks. Atribut ini merupakan tambahan dari atribut kunci tabel, yang secara otomatis diproyeksikan ke dalam setiap indeks. Anda dapat memproyeksikan atribut dari tipe data apa pun, termasuk skalar, dokumen, dan set.
- Pengaturan throughput yang disediakan untuk indeks, jika perlu:
 - Untuk indeks sekunder global, Anda harus menentukan pengaturan unit kapasitas baca dan tulis. Pengaturan throughput yang disediakan ini tidak tergantung pada pengaturan tabel dasar.
 - Untuk indeks sekunder lokal, Anda tidak perlu menentukan pengaturan unit kapasitas baca dan tulis. Setiap operasi baca dan tulis pada indeks sekunder lokal diambil dari pengaturan throughput yang disediakan dari tabel dasarnya.

Untuk fleksibilitas kueri maksimum, Anda dapat membuat hingga 20 indeks sekunder global (kuota default) dan hingga 5 indeks sekunder lokal per tabel.

Kuota indeks sekunder global per tabel adalah lima untuk Wilayah AWS berikut:

- AWSGovCloud(AS-timur)
- AWSGovCloud(AS-Barat)
- Europe (Stockholm)

Untuk mendapatkan daftar rinci indeks sekunder pada tabel, gunakan operasi `DescribeTable`. `DescribeTable` mengembalikan nama, ukuran penyimpanan, dan jumlah item untuk setiap indeks

sekunder pada tabel. Nilai-nilai ini tidak diperbarui secara real time, tetapi diperbarui kira-kira setiap enam jam.

Anda dapat mengakses data dalam indeks sekunder menggunakan operasi Query atau Scan. Anda harus menentukan nama tabel dasar dan nama indeks yang ingin Anda gunakan, atribut yang akan dikembalikan dalam hasil, dan ekspresi syarat atau filter apa pun yang ingin Anda terapkan. DynamoDB dapat mengembalikan hasil dalam urutan naik atau turun.

Saat Anda menghapus tabel, semua indeks yang terkait dengan tabel tersebut juga akan dihapus.

Untuk praktik terbaik, lihat [Praktik terbaik untuk menggunakan indeks sekunder di DynamoDB](#).

Menggunakan Indeks Sekunder Global di DynamoDB

Beberapa aplikasi mungkin perlu melakukan banyak jenis kueri, menggunakan berbagai atribut berbeda sebagai kriteria kueri. Untuk mendukung persyaratan ini, Anda dapat membuat satu atau beberapa indeks sekunder global dan mengeluarkan permintaan Query terhadap indeks ini di Amazon DynamoDB.

Topik

- [Skenario: Menggunakan Indeks Sekunder Global](#)
- [Proyeksi atribut](#)
- [Membaca data dari Indeks Sekunder Global Global Global Global Global Global Global Global Global Global Global](#)
- [Sinkronisasi antara tabel dan Indeks Sinkronisasi antara tabel dan Indeks Sekunder Global Global Global Global Global](#)
- [Kelas kelas dengan Indeks Sekunder Global Global Global Global Global Global Global Global Global](#)
- [Pertimbangan pertimbangan Throughput yang disediakan untuk Indeks Throughput yang Disediakan untuk Indeks Throughput untuk Indeks](#)
- [Pertimbangan penyimpanan untuk Indeks Sekunder Global Global Global Global Global Global Global Global Global Global Global](#)
- [Mengelola Indeks Sekunder Global](#)
- [Menggunakan Indeks Sekunder Global: Java](#)
- [Menggunakan Indeks Sekunder Global: .NET](#)
- [Bekerja dengan Indeks Sekunder Global:AWS CLI](#)

Skenario: Menggunakan Indeks Sekunder Global

Sebagai ilustrasi, pertimbangkan tabel bernama `GameScores` yang melacak pengguna dan skor untuk aplikasi game seluler. Setiap item dalam `GameScores` diidentifikasi dengan kunci partisi (`UserId`) dan kunci pengurutan (`GameTitle`). Diagram berikut menunjukkan bagaimana item dalam tabel akan diatur. (Tidak semua atribut ditampilkan.)

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	...
...

Sekarang anggaplah Anda ingin menulis aplikasi papan peringkat untuk menampilkan skor teratas untuk setiap game. Kueri yang menentukan atribut kunci (`UserId` dan `GameTitle`) akan sangat efisien. Namun, jika aplikasi perlu mengambil data dari `GameScores` berdasarkan `GameTitle` saja, aplikasi tersebut perlu menggunakan operasi `Scan`. Karena lebih banyak item ditambahkan ke tabel, pemindaian semua data akan menjadi lambat dan tidak efisien. Hal ini membuat sulit untuk menjawab pertanyaan-pertanyaan seperti berikut:

- Berapa skor tertinggi yang pernah tercatat untuk game `Meteor Blasters`?
- Pengguna mana yang memiliki skor tertinggi untuk `Galaxy Invaders`?
- Berapa rasio kemenangan tertinggi vs kerugian?

Untuk mempercepat kueri pada atribut non-kunci, Anda dapat membuat indeks sekunder global. Indeks sekunder global berisi pilihan atribut dari tabel dasar, tetapi atribut tersebut diatur oleh kunci primer yang berbeda dari tabel. Kunci indeks tidak perlu memiliki atribut kunci apa pun dari tabel. Bahkan tidak perlu memiliki skema kunci yang sama dengan tabel.

Misalnya, Anda dapat membuat indeks sekunder global bernama `GameTitleIndex`, dengan kunci partisi `GameTitle` dan kunci pengurutan `TopScore`. Atribut kunci primer tabel dasar selalu diproyeksikan ke dalam indeks, sehingga atribut `UserId` juga ada. Diagram berikut menunjukkan seperti apa indeks `GameTitleIndex` itu.

GameTitleIndex

<i>GameTitle</i>	<i>TopScore</i>	<i>UserId</i>
"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"
...

Sekarang Anda dapat meminta `GameTitleIndex` dan dengan mudah mendapatkan skor untuk `Meteor Blasters`. Hasilnya diurutkan berdasarkan nilai kunci pengurutan, `TopScore`. Jika Anda mengatur parameter `ScanIndexForward` ke `false`, hasilnya dikembalikan dalam urutan menurun, sehingga skor tertinggi dikembalikan terlebih dahulu.

Setiap indeks sekunder global harus memiliki kunci partisi, dan dapat memiliki kunci pengurutan opsional. Skema kunci indeks dapat berbeda dari skema tabel dasar. Anda dapat memiliki tabel dengan kunci primer sederhana (kunci partisi), dan membuat indeks sekunder global dengan kunci

primer komposit (kunci partisi dan kunci pengurutan)—atau sebaliknya. Atribut kunci indeks dapat terdiri dari atribut `String`, `Number`, atau `Binary` tingkat atas dari tabel dasar. Jenis skalar, jenis dokumen, dan jenis kumpulan lainnya tidak diperbolehkan.

Anda dapat memproyeksikan atribut tabel dasar lainnya ke dalam indeks jika Anda ingin. Saat Anda membuat kueri indeks, DynamoDB dapat mengambil atribut yang diproyeksikan ini secara efisien. Namun, kueri indeks sekunder global tidak dapat mengambil atribut dari tabel dasar. Misalnya, jika Anda membuat kueri `GameTitleIndex` seperti yang ditunjukkan pada diagram sebelumnya, kueri tidak dapat mengakses atribut non-kunci selain `TopScore` (meskipun atribut kunci `GameTitle` dan `UserId` akan diproyeksikan secara otomatis).

Dalam tabel DynamoDB, setiap nilai kunci harus unik. Namun, nilai kunci dalam indeks sekunder global tidak perlu unik. Sebagai ilustrasi, anggaplah sebuah game bernama `Comet Quest` sangat sulit, dengan banyak pengguna baru mencoba tetapi gagal mendapatkan skor di atas nol. Berikut beberapa data yang dapat mewakili hal tersebut.

UserId	GameTitle	TopScore
123	Comet Quest	0
201	Comet Quest	0
301	Comet Quest	0

Saat data ini ditambahkan ke tabel `GameScores`, DynamoDB menyebarkannya ke `GameTitleIndex`. Jika kita kemudian mengkueri indeks menggunakan `Comet Quest` untuk `GameTitle` dan `0` untuk `TopScore`, data berikut dikembalikan.

<i>GameTitle</i>	<i>TopScore</i>	<i>UserId</i>
"Comet Quest"	0	"123"
"Comet Quest"	0	"201"
"Comet Quest"	0	"301"

Hanya item dengan nilai kunci tertentu yang muncul dalam respons. Dalam kumpulan data itu, item-item tersebut tidak berada dalam urutan tertentu.

Indeks sekunder global hanya melacak item data di mana atribut utamanya benar-benar ada. Misalnya, Anda menambahkan item baru lainnya ke tabel `GameScores`, tetapi hanya menyediakan atribut kunci primer yang diperlukan.

Userld	GameTitle
400	Comet Quest

Karena Anda tidak menentukan atribut `TopScore`, DynamoDB tidak akan menyebarkan item ini ke `GameTitleIndex`. Jadi, jika Anda mengkueri `GameScores` untuk semua item `Comet Quest`, Anda akan mendapatkan empat item berikut.

Userld	GameTitle	TopScore
"123"	"Comet Quest"	0
"201"	"Comet Quest"	0
"301"	"Comet Quest"	0
"400"	"Comet Quest"	

Kueri serupa pada `GameTitleIndex` masih akan mengembalikan tiga item, bukan empat. Ini karena item dengan `TopScore` yang tidak ada tidak disebarkan ke indeks.

<i>GameTitle</i>	<i>TopScore</i>	<i>Userld</i>
"Comet Quest"	0	"123"
"Comet Quest"	0	"201"
"Comet Quest"	0	"301"

Proyeksi atribut

Proyeksi adalah kumpulan atribut yang disalin dari tabel ke indeks sekunder. Kunci partisi dan kunci pengurutan tabel selalu diproyeksikan ke dalam indeks; Anda dapat memproyeksikan atribut lain untuk mendukung persyaratan kueri aplikasi Anda. Saat Anda membuat kueri indeks, Amazon DynamoDB dapat mengakses atribut apa pun dalam proyeksi seolah-olah atribut tersebut ada dalam tabelnya sendiri.

Saat Anda membuat indeks sekunder, Anda perlu menentukan atribut yang akan diproyeksikan ke dalam indeks. DynamoDB menyediakan tiga opsi berbeda untuk ini:

- **KEYS_ONLY** – Setiap item dalam indeks hanya terdiri dari kunci partisi tabel dan nilai kunci pengurutan, ditambah nilai kunci indeks. Opsi **KEYS_ONLY** menghasilkan indeks sekunder sekecil mungkin.
- **INCLUDE** – Selain atribut yang dijelaskan dalam **KEYS_ONLY**, indeks sekunder akan menyertakan atribut non-kunci lainnya yang Anda tentukan.
- **ALL** – Indeks sekunder mencakup semua atribut dari tabel sumber. Karena semua data tabel diduplikasi dalam indeks, proyeksi **ALL** menghasilkan indeks sekunder terbesar yang mungkin.

Pada diagram sebelumnya, `GameTitleIndex` hanya memiliki satu atribut yang diproyeksikan: `UserId`. Jadi, meskipun aplikasi dapat secara efisien menentukan `UserId` dari pencetak skor terbanyak untuk setiap game menggunakan `GameTitle` dan `TopScore` dalam kueri, aplikasi tidak dapat secara efisien menentukan rasio kemenangan vs. kekalahan tertinggi untuk pencetak skor terbanyak. Untuk melakukannya, aplikasi harus melakukan kueri tambahan di tabel dasar untuk mengambil kemenangan dan kekalahan untuk masing-masing pencetak skor terbanyak. Cara yang lebih efisien untuk mendukung kueri pada data ini adalah dengan memproyeksikan atribut-atribut ini dari tabel dasar ke dalam indeks sekunder global, seperti yang ditunjukkan dalam diagram ini.

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
...

Karena atribut non-kunci `Wins` dan `Losses` diproyeksikan ke dalam indeks, aplikasi dapat menentukan rasio menang vs. kalah untuk game apa pun, atau untuk kombinasi game dan ID pengguna apa pun.

Saat Anda memilih atribut untuk diproyeksikan ke dalam indeks sekunder global, Anda harus mempertimbangkan tradeoff antara biaya throughput yang disediakan dan biaya penyimpanan:

- Jika Anda hanya perlu mengakses beberapa atribut dengan latensi serendah mungkin, pertimbangkan untuk hanya memproyeksikan atribut tersebut ke dalam indeks sekunder global. Semakin kecil indeks, semakin sedikit biaya untuk menyimpannya, dan semakin sedikit biaya tulis Anda.
- Jika aplikasi Anda sering mengakses beberapa atribut non-kunci, Anda harus mempertimbangkan untuk memproyeksikan atribut tersebut ke dalam indeks sekunder global. Biaya penyimpanan tambahan untuk indeks sekunder global mengimbangi biaya sering melakukan pemindaian tabel.
- Jika Anda perlu sering mengakses sebagian besar atribut non-kunci, Anda dapat memproyeksikan atribut ini—atau bahkan seluruh tabel dasar—ke dalam indeks sekunder global. Ini memberi Anda

fleksibilitas maksimum. Namun, biaya penyimpanan Anda akan meningkat, atau bahkan berlipat ganda.

- Jika aplikasi Anda jarang membuat kueri tabel, tetapi harus melakukan banyak penulisan atau pembaruan terhadap data dalam tabel, pertimbangkan untuk memproyeksikan KEYS_ONLY. Indeks sekunder global akan berukuran minimal, tetapi akan tetap tersedia bila diperlukan untuk aktivitas kueri.

Membaca data dari Indeks Sekunder Global Global Global Global Global Global Global Global Global Global

Anda dapat mengambil item dari indeks sekunder global menggunakan operasi Query dan Scan. Operasi GetItem dan BatchGetItem tidak dapat digunakan pada indeks sekunder global.

Mengkueri Indeks Sekunder Global

Anda dapat menggunakan operasi Query untuk mengakses satu atau beberapa item dalam indeks sekunder global. Kueri harus menentukan nama tabel dasar dan nama indeks yang ingin Anda gunakan, atribut yang akan dikembalikan dalam hasil kueri, dan syarat kueri apa pun yang ingin Anda terapkan. DynamoDB dapat mengembalikan hasil dalam urutan naik atau turun.

Pertimbangkan data berikut yang dikembalikan dari Query yang meminta data game untuk aplikasi papan peringkat.

```
{
  "TableName": "GameScores",
  "IndexName": "GameTitleIndex",
  "KeyConditionExpression": "GameTitle = :v_title",
  "ExpressionAttributeValues": {
    ":v_title": {"S": "Meteor Blasters"}
  },
  "ProjectionExpression": "UserId, TopScore",
  "ScanIndexForward": false
}
```

Dalam kueri ini:

- DynamoDB mengakses GameTitleIndex, menggunakan kunci GameTitlepartisi untuk menemukan item indeks untuk Meteor Blasters. Semua item indeks dengan kunci ini disimpan berdekatan satu sama lain untuk pengambilan cepat.

- Dalam game ini, DynamoDB menggunakan indeks untuk mengakses semua ID pengguna dan skor tertinggi untuk game ini.
- Hasilnya dikembalikan, diurutkan dalam urutan menurun karena parameter `ScanIndexForward` diatur ke salah.

Memindai Indeks Sekunder Global

Anda dapat menggunakan operasi `Scan` untuk mengambil semua data dari indeks sekunder global. Anda harus memberikan nama tabel dasar dan nama indeks dalam permintaan. Dengan `Scan`, DynamoDB membaca semua data dalam indeks dan mengembalikannya ke aplikasi. Anda juga dapat meminta agar hanya sebagian data yang dikembalikan, dan data yang tersisa harus dibuang. Untuk melakukan ini, gunakan parameter `FilterExpression` dari operasi `Scan`. Untuk informasi selengkapnya, lihat [Ekspresi filter untuk pemindaian](#).

Sinkronisasi antara tabel dan Indeks Sinkronisasi antara tabel dan Indeks Sekunder Global Global Global Global Global

DynamoDB secara otomatis menyinkronkan setiap indeks sekunder global dengan tabel dasarnya. Saat aplikasi menulis atau menghapus item dalam tabel, indeks sekunder global apa pun di tabel tersebut diperbarui secara asinkron, menggunakan model yang akhirnya konsisten. Aplikasi tidak pernah menulis langsung ke file indeks. Namun, penting bagi Anda untuk memahami implikasi dari cara DynamoDB mempertahankan indeks ini.

Indeks sekunder global mewarisi mode kapasitas baca/tulis dari tabel dasar. Untuk informasi selengkapnya, lihat [Pertimbangan saat mengubah Mode Kapasitas baca/tulis](#).

Saat Anda membuat indeks sekunder global, Anda menentukan satu atau beberapa atribut kunci indeks dan tipe datanya. Ini berarti bahwa setiap kali Anda menulis item ke tabel dasar, tipe data untuk atribut tersebut harus cocok dengan tipe data skema kunci indeks. Dalam kasus `GameTitleIndex`, kunci partisi `GameTitle` dalam indeks didefinisikan sebagai tipe data `String`. Kunci pengurutan `TopScore` dalam indeks adalah tipe `Number`. Jika Anda mencoba menambahkan item ke tabel `GameScores` dan menentukan tipe data yang berbeda untuk `GameTitle` atau `TopScore`, DynamoDB mengembalikan `ValidationException` karena tipe data tidak cocok.

Saat Anda meletakkan atau menghapus item dalam tabel, indeks sekunder global pada tabel tersebut diperbarui secara konsisten. Perubahan pada data tabel disebarkan ke indeks sekunder global dalam sepersekian detik, dalam kondisi normal. Namun, dalam beberapa skenario kegagalan yang tidak mungkin, penundaan propagasi yang lebih lama mungkin terjadi. Karena itu, aplikasi Anda perlu

mengantisipasi dan menangani situasi di mana kueri pada indeks sekunder global mengembalikan hasil yang tidak mutakhir.

Jika Anda menulis item ke tabel, Anda tidak perlu menentukan atribut untuk kunci pengurutan indeks sekunder global apa pun. Menggunakan `GameTitleIndex` sebagai contoh, Anda tidak perlu menentukan nilai atribut `TopScore` untuk menulis item baru ke tabel `GameScores`. Dalam hal ini, DynamoDB tidak menulis data apa pun ke indeks untuk item khusus ini.

Tabel dengan banyak indeks sekunder global menimbulkan biaya lebih tinggi untuk aktivitas tulis daripada tabel dengan indeks lebih sedikit. Untuk informasi selengkapnya, lihat [Pertimbangan pertimbangan Throughput yang disediakan untuk Indeks Throughput yang Disediakan untuk Indeks Throughput untuk Indeks](#).

Kelas kelas dengan Indeks Sekunder Global Global Global Global Global Global Global Global Global

Indeks sekunder global akan selalu menggunakan kelas tabel yang sama dengan tabel dasarnya. Setiap kali indeks sekunder global baru ditambahkan untuk tabel, indeks baru akan menggunakan kelas tabel yang sama dengan tabel dasarnya. Ketika kelas tabel tabel diperbarui, semua indeks sekunder global terkait diperbarui juga.

Pertimbangan pertimbangan Throughput yang disediakan untuk Indeks Throughput yang Disediakan untuk Indeks Throughput untuk Indeks

Saat Anda membuat indeks sekunder global pada tabel mode yang disediakan, Anda harus menentukan unit kapasitas baca dan tulis untuk beban kerja yang diharapkan pada indeks tersebut. Pengaturan throughput yang disediakan dari indeks sekunder global terpisah dari pengaturan tabel dasarnya. Operasi `Query` pada indeks sekunder global menggunakan unit kapasitas baca dari indeks, bukan tabel dasar. Saat Anda meletakkan, memperbarui, atau menghapus item dalam tabel, indeks sekunder global pada tabel tersebut juga diperbarui. Pembaruan indeks ini menggunakan unit kapasitas tulis dari indeks, bukan dari tabel dasar.

Misalnya, jika Anda `Query` indeks sekunder global dan melebihi kapasitas baca yang disediakan, permintaan Anda akan dibatasi. Jika Anda melakukan aktivitas tulis berat pada tabel, tetapi indeks sekunder global pada tabel tersebut memiliki kapasitas tulis yang tidak mencukupi, aktivitas tulis pada tabel akan dibatasi.

⚠ Important

Untuk menghindari potensi throttling, kapasitas tulis yang disediakan untuk indeks sekunder global harus sama atau lebih besar dari kapasitas tulis tabel dasar karena pembaruan baru menulis ke tabel dasar dan indeks sekunder global.

Untuk melihat pengaturan throughput yang disediakan untuk indeks sekunder global, gunakan operasi `DescribeTable`. Informasi terperinci tentang semua indeks sekunder global tabel dikembalikan.

Unit kapasitas yang disediakan

Indeks sekunder global mendukung bacaan akhir konsisten, yang masing-masing menggunakan setengah dari unit kapasitas baca. Ini berarti bahwa satu kueri indeks sekunder global dapat mengambil hingga $2 \times 4 \text{ KB} = 8 \text{ KB}$ per unit kapasitas baca.

Untuk kueri indeks sekunder global, DynamoDB menghitung aktivitas baca yang disediakan dengan cara yang sama seperti kueri terhadap tabel. Satu-satunya perbedaan adalah bahwa perhitungan didasarkan pada ukuran entri indeks, bukan ukuran item dalam tabel dasar. Jumlah unit kapasitas baca adalah jumlah dari semua ukuran atribut yang diproyeksikan di semua item yang dikembalikan. Hasilnya kemudian dibulatkan hingga batas 4 KB berikutnya. Untuk informasi selengkapnya tentang cara DynamoDB menghitung penggunaan throughput yang disediakan, lihat [Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB](#).

Ukuran maksimum hasil yang dikembalikan oleh operasi `Query` adalah 1 MB. Ini termasuk ukuran semua nama dan nilai atribut di semua item yang dikembalikan.

Misalnya, pertimbangkan indeks sekunder global di mana setiap item berisi 2.000 byte data. Sekarang anggaplah Anda `Query` indeks ini dan `KeyConditionExpression` kueri cocok dengan delapan item. Ukuran total item yang cocok adalah $2.000 \text{ byte} \times 8 \text{ item} = 16.000 \text{ byte}$. Hasil ini kemudian dibulatkan ke atas hingga batas 4 KB terdekat. Karena kueri indeks sekunder global pada akhirnya konsisten, biaya totalnya adalah $0,5 \times (16 \text{ KB} / 4 \text{ KB})$, atau 2 unit kapasitas baca.

Unit kapasitas yang disediakan

Ketika item dalam tabel ditambahkan, diperbarui, atau dihapus, dan indeks sekunder global terpengaruh oleh ini, indeks sekunder global menggunakan unit kapasitas tulis yang disediakan untuk operasi. Total biaya throughput yang disediakan untuk penulisan terdiri dari jumlah unit kapasitas tulis yang digunakan dengan menulis ke tabel dasar dan yang dikonsumsi dengan memperbarui indeks

sekunder global. Jika penulisan ke tabel tidak memerlukan pembaruan indeks sekunder global, tidak ada kapasitas tulis yang digunakan dari indeks.

Agar penulisan tabel berhasil, pengaturan throughput yang disediakan untuk tabel dan semua indeks sekunder globalnya harus memiliki kapasitas tulis yang cukup untuk mengakomodasi penulisan. Jika tidak, penulisan ke tabel akan dibatasi.

Biaya menulis item ke indeks sekunder global tergantung pada beberapa faktor:

- Jika Anda menulis item baru ke tabel yang menentukan atribut yang diindeks, atau Anda memperbarui item yang ada untuk menentukan atribut terindeks yang sebelumnya tidak ditentukan, satu operasi tulis diperlukan untuk memasukkan item ke dalam indeks.
- Jika pembaruan ke tabel mengubah nilai atribut kunci yang diindeks (dari A ke B), dua penulisan diperlukan, satu untuk menghapus item sebelumnya dari indeks dan penulisan lainnya untuk memasukkan item baru ke dalam indeks.
- Jika item ada dalam indeks, tetapi penulisan ke tabel menyebabkan atribut yang diindeks dihapus, satu penulisan diperlukan untuk menghapus proyeksi item lama dari indeks.
- Jika item tidak ada dalam indeks sebelum atau setelah item diperbarui, tidak ada biaya penulisan tambahan untuk indeks.
- Jika pembaruan ke tabel hanya mengubah nilai atribut yang diproyeksikan dalam skema kunci indeks, tetapi tidak mengubah nilai atribut kunci yang diindeks, satu penulisan diperlukan untuk memperbarui nilai atribut yang diproyeksikan ke dalam indeks.

Semua faktor ini mengasumsikan bahwa ukuran setiap item dalam indeks kurang dari atau sama dengan ukuran item 1 KB untuk menghitung unit kapasitas tulis. Entri indeks yang lebih besar memerlukan unit kapasitas tulis tambahan. Anda dapat meminimalkan biaya penulisan dengan mempertimbangkan atribut mana yang perlu dikembalikan oleh kueri Anda dan hanya memproyeksikan atribut tersebut ke dalam indeks.

Pertimbangan penyimpanan untuk Indeks Sekunder Global Global Global Global Global Global Global Global Global Global

Saat aplikasi menulis item ke tabel, DynamoDB secara otomatis menyalin subset atribut yang benar ke indeks sekunder global tempat atribut tersebut akan muncul. Akun AWS Anda dikenakan biaya untuk penyimpanan item di tabel dasar dan juga untuk penyimpanan atribut di indeks sekunder global apa pun di tabel itu.

Jumlah ruang yang digunakan oleh item indeks adalah jumlah dari berikut ini:

- Ukuran dalam byte dari kunci primer tabel dasar (kunci partisi dan kunci pengurutan)
- Ukuran dalam byte atribut kunci indeks
- Ukuran dalam byte dari atribut yang diproyeksikan (jika ada)
- 100 byte dari overhead per item indeks

Untuk memperkirakan persyaratan penyimpanan untuk indeks sekunder global, Anda bisa memperkirakan ukuran rata-rata item dalam indeks dan kemudian mengalikannya dengan jumlah item dalam tabel dasar yang memiliki atribut kunci indeks sekunder global.

Jika tabel berisi item di mana atribut tertentu tidak ditentukan, tetapi atribut tersebut didefinisikan sebagai kunci partisi indeks atau kunci pengurutan, DynamoDB tidak menulis data apa pun untuk item tersebut ke indeks.

Mengelola Indeks Sekunder Global

Bagian ini menjelaskan cara membuat, memodifikasi, dan menghapus indeks sekunder global di Amazon DynamoDB.

Topik

- [Membuat Tabel dengan Indeks Sekunder Global](#)
- [Menggambarkan Indeks Sekunder Global pada Tabel](#)
- [Menambahkan Indeks Sekunder Global ke Tabel yang Ada](#)
- [Menghapus Indeks Sekunder Global](#)
- [Memodifikasi Indeks Sekunder Global selama Pembuatan](#)
- [Mendeteksi dan mengoreksi pelanggaran kunci indeks](#)

Membuat Tabel dengan Indeks Sekunder Global

Untuk membuat tabel dengan satu atau beberapa indeks sekunder global, gunakan operasi `CreateTable` dengan parameter `GlobalSecondaryIndexes`. Untuk fleksibilitas kueri maksimum, Anda dapat membuat hingga 20 indeks sekunder global (kuota default) per tabel.

Anda harus menentukan satu atribut untuk bertindak sebagai kunci partisi indeks. Anda dapat secara opsional menentukan atribut lain untuk kunci pengurutan indeks. Atribut kunci ini tidak perlu sama dengan atribut kunci dalam tabel. Misalnya, di `GameScore` tabel (lihat [Menggunakan Indeks Sekunder Global di DynamoDB](#)), `tidakTopScore` juga `tidakTopScoreDateTime` adalah atribut kunci. Anda

dapat membuat indeks sekunder global dengan kunci partisi `TopScore` dan kunci pengurutan `TopScoreDateTime`. Anda dapat menggunakan indeks semacam itu untuk menentukan apakah ada korelasi antara skor tinggi dan waktu game dimainkan.

Setiap atribut kunci indeks harus berupa skalar bertipe `String`, `Number`, atau `Binary`. (Ini tidak bisa berupa dokumen atau kumpulan.) Anda dapat memproyeksikan atribut tipe data apa pun ke dalam indeks sekunder global. Ini termasuk skalar, dokumen, dan kumpulan. Untuk daftar lengkap tipe data, lihat [Tipe Data](#).

Jika menggunakan mode yang disediakan, Anda harus menyediakan pengaturan `ProvisionedThroughput` untuk indeks, yang terdiri dari `ReadCapacityUnits` dan `WriteCapacityUnits`. Pengaturan throughput yang disediakan ini terpisah dari pengaturan tabel, tetapi berperilaku dengan cara yang serupa. Untuk informasi selengkapnya, lihat [Pertimbangan pertimbangan Throughput yang disediakan untuk Indeks Throughput yang Disediakan untuk Indeks Throughput untuk Indeks](#).

Indeks sekunder global mewarisi mode kapasitas baca/tulis dari tabel dasar. Untuk informasi selengkapnya, lihat [Pertimbangan saat mengubah Mode Kapasitas baca/tulis](#).

Note

Operasi isi ulang dan operasi tulis yang sedang berlangsung berbagi throughput penulisan dalam indeks sekunder global. Saat membuat GSI baru, penting untuk memeriksa apakah pilihan kunci partisi Anda menghasilkan distribusi data atau lalu lintas yang tidak merata atau menyempit di seluruh nilai kunci partisi indeks baru. Jika ini terjadi, Anda bisa melihat pengurutan dan menulis operasi yang terjadi pada saat yang sama dan throttling menulis ke tabel dasar. Layanan ini mengambil langkah-langkah untuk meminimalkan potensi skenario ini, tetapi tidak memiliki wawasan tentang bentuk data pelanggan sehubungan dengan kunci partisi indeks, proyeksi yang dipilih, atau sparseness kunci utama indeks.

Jika Anda menduga bahwa indeks sekunder global baru Anda mungkin memiliki data atau distribusi lalu lintas yang sempit atau miring di seluruh nilai kunci partisi, pertimbangkan hal berikut sebelum menambahkan indeks baru ke tabel penting secara operasional.

- Mungkin paling aman untuk menambahkan indeks pada saat aplikasi Anda mendorong jumlah lalu lintas paling sedikit.
- Pertimbangkan untuk mengaktifkan CloudWatch Wawasan Kontributor pada tabel dasar dan indeks Anda. Ini akan memberi Anda wawasan berharga tentang distribusi lalu lintas Anda.

- Untuk tabel dan indeks dasar mode kapasitas yang disediakan, atur kapasitas tulis yang disediakan dari indeks baru Anda ke setidaknya dua kali lipat dari tabel dasar Anda. Tonton `WriteThrottleEvents`, `ThrottledRequests`, `OnlineIndexPercentageProgress`, dan `OnlineIndexPercentageProgress` CloudWatch metrik sepanjang proses. Sesuaikan kapasitas tulis yang disediakan sesuai kebutuhan untuk menyelesaikan isi ulang dalam waktu yang wajar tanpa efek throttling yang signifikan pada operasi Anda yang sedang berlangsung.
- Bersiaplah untuk membatalkan pembuatan indeks jika Anda mengalami dampak operasional karena pembatasan penulisan, dan meningkatkan kapasitas tulis yang disediakan di GSI baru Anda tidak menyelesaikannya.

Menggambarkan Indeks Sekunder Global pada Tabel

Untuk melihat status semua indeks sekunder global pada tabel, gunakan operasi `DescribeTable`. Bagian `GlobalSecondaryIndexes` dari respons menunjukkan semua indeks pada tabel, bersama dengan status saat ini dari masing-masing (`IndexStatus`).

`IndexStatus` untuk indeks sekunder global akan menjadi salah satu dari berikut ini:

- **CREATING** — Indeks sedang dibuat, dan belum tersedia untuk digunakan.
- **ACTIVE** — Indeks siap digunakan, dan aplikasi dapat melakukan operasi `Query` pada indeks.
- **UPDATING** — Pengaturan throughput yang disediakan dari indeks sedang diubah.
- **DELETING** — Indeks saat ini sedang dihapus, dan tidak dapat lagi digunakan.

Ketika DynamoDB selesai membangun indeks sekunder global, status indeks berubah dari **CREATING** menjadi **ACTIVE**.

Menambahkan Indeks Sekunder Global ke Tabel yang Ada

Untuk menambahkan indeks sekunder global ke tabel yang ada, gunakan operasi `UpdateTable` dengan parameter `GlobalSecondaryIndexUpdates`. Anda harus memberikan yang berikut ini:

- Nama indeks. Nama harus unik di antara semua indeks pada tabel.
- Skema kunci indeks. Anda harus menentukan satu atribut untuk kunci partisi indeks; Anda dapat secara opsional menentukan atribut lain untuk kunci pengurutan indeks. Atribut kunci ini tidak perlu sama dengan atribut kunci dalam tabel. Tipe data untuk setiap atribut skema harus skalar: `String`, `Number`, atau `Binary`.

- Atribut yang akan diproyeksikan dari tabel ke dalam indeks:
 - KEYS_ONLY — Setiap item dalam indeks hanya terdiri dari kunci partisi tabel dan nilai kunci pengurutan, ditambah nilai kunci indeks.
 - INCLUDE — Selain atribut yang dijelaskan di KEYS_ONLY, indeks sekunder menyertakan atribut non-kunci lainnya yang Anda tentukan.
 - ALL — Indeks mencakup semua atribut dari tabel sumber.
- Setelan throughput yang disediakan untuk indeks, yang terdiri dari `ReadCapacityUnits` dan `WriteCapacityUnits`. Pengaturan throughput yang disediakan ini terpisah dari pengaturan tabel.

Anda hanya dapat membuat satu indeks sekunder global per operasi `UpdateTable`.

Fase Pembuatan Indeks

Saat Anda menambahkan indeks sekunder global baru ke tabel yang sudah ada, tabel akan terus tersedia saat indeks sedang dibuat. Namun, indeks baru tidak tersedia untuk operasi Kueri hingga statusnya berubah dari `CREATING` menjadi `ACTIVE`.

Note

Pembuatan indeks sekunder global tidak menggunakan `Application Auto Scaling`. Meningkatkan `MINKapasitas` `Application Auto Scaling` tidak akan mengurangi waktu pembuatan indeks sekunder global.

Di balik layar, DynamoDB membangun indeks dalam dua fase:

Alokasi Sumber Daya

DynamoDB mengalokasikan sumber daya komputasi dan penyimpanan yang diperlukan untuk membangun indeks.

Selama fase alokasi sumber daya, atribut `IndexStatus` adalah `CREATING` dan atribut `Backfilling` adalah `salah`. Gunakan operasi `DescribeTable` untuk mengambil status tabel dan semua indeks sekundernya.

Saat indeks berada dalam fase alokasi sumber daya, Anda tidak dapat menghapus indeks atau menghapus tabel induknya. Anda juga tidak dapat mengubah throughput indeks atau tabel yang

disediakan. Anda tidak dapat menambah atau menghapus indeks lain pada tabel. Namun, Anda dapat mengubah throughput yang disediakan dari indeks lain ini.

Backfilling

Untuk setiap item dalam tabel, DynamoDB menentukan kumpulan atribut mana yang akan ditulis ke indeks berdasarkan proyeksinya (KEYS_ONLY, INCLUDE, atau ALL). Kemudian menulis atribut ini ke indeks. Selama fase backfilling, DynamoDB melacak item yang ditambahkan, dihapus, atau diperbarui dalam tabel. Atribut dari item ini juga ditambahkan, dihapus, atau diperbarui dalam indeks yang sesuai.

Selama fase backfilling, atribut `IndexStatus` diatur ke `CREATING`, dan atribut `Backfilling` adalah benar. Gunakan operasi `DescribeTable` untuk mengambil status tabel dan semua indeks sekundernya.

Saat indeks sedang backfilling, Anda tidak dapat menghapus tabel induknya. Namun, Anda masih dapat menghapus indeks atau mengubah throughput tabel yang disediakan dan indeks sekunder globalnya.

Note

Selama fase backfilling, beberapa penulisan item yang melanggar mungkin berhasil sementara yang lain ditolak. Setelah backfilling, semua penulisan ke item yang melanggar skema kunci indeks baru akan ditolak. Sebaiknya Anda menjalankan alat Detektor Pelanggaran setelah fase backfilling selesai untuk mendeteksi dan mengatasi pelanggaran utama yang mungkin terjadi. Untuk informasi selengkapnya, lihat [Mendeteksi dan mengoreksi pelanggaran kunci indeks](#).

Sementara alokasi sumber daya dan fase backfilling sedang berlangsung, indeks berada dalam status `CREATING`. Selama waktu ini, DynamoDB melakukan operasi baca pada tabel. Anda tidak dikenakan biaya untuk operasi baca dari tabel dasar untuk mengisi indeks sekunder global. Namun, Anda dikenakan biaya untuk operasi tulis untuk mengisi indeks sekunder global yang baru dibuat.

Saat pembangunan indeks selesai, statusnya berubah menjadi `ACTIVE`. Anda tidak dapat Query atau Scan indeks sampai indeks itu `ACTIVE`.

Note

Dalam beberapa kasus, DynamoDB tidak dapat menulis data dari tabel ke indeks karena pelanggaran kunci indeks. Hal ini dapat terjadi jika:

- Tipe data nilai atribut tidak cocok dengan tipe data skema kunci indeks.
- Ukuran atribut melebihi panjang maksimum untuk atribut kunci indeks.
- Atribut kunci indeks memiliki String kosong atau nilai atribut Biner kosong.

Pelanggaran kunci indeks tidak mengganggu pembuatan indeks sekunder global. Namun, ketika indeks menjadi ACTIVE, kunci yang melanggar tidak ada dalam indeks.

DynamoDB menyediakan alat mandiri untuk menemukan dan menyelesaikan masalah ini. Untuk informasi selengkapnya, lihat [Mendeteksi dan mengoreksi pelanggaran kunci indeks](#).

Menambahkan Indeks Sekunder Global ke Large Table

Waktu yang dibutuhkan untuk membangun indeks sekunder global tergantung beberapa faktor, seperti berikut ini:

- Ukuran tabel
- Jumlah item dalam tabel yang memenuhi syarat untuk dimasukkan dalam indeks
- Jumlah atribut yang diproyeksikan ke dalam indeks
- Kapasitas tulis indeks yang disediakan
- Aktivitas tulis di tabel utama selama pembuatan indeks

Jika Anda menambahkan indeks sekunder global ke tabel yang sangat besar, mungkin perlu waktu lama untuk menyelesaikan proses pembuatan. Untuk memantau kemajuan dan menentukan apakah indeks memiliki kapasitas tulis yang memadai, lihat Amazon berikut CloudWatch metrik:

- `OnlineIndexPercentageProgress`
- `OnlineIndexConsumedWriteCapacity`
- `OnlineIndexThrottleEvents`

Note

Untuk informasi lebih lanjut tentang CloudWatch metrik yang terkait dengan DynamoDB, lihat [Metrik DynamoDB](#).

Jika pengaturan throughput tulis yang disediakan pada indeks terlalu rendah, pembuatan indeks akan memakan waktu lebih lama untuk diselesaikan. Untuk mempersingkat waktu yang diperlukan untuk membangun indeks sekunder global baru, Anda dapat meningkatkan kapasitas tulis yang disediakan untuk sementara.

Note

Sebagai aturan umum, sebaiknya atur kapasitas tulis indeks yang disediakan menjadi 1,5 kali kapasitas tulis tabel. Ini adalah pengaturan yang baik untuk banyak kasus penggunaan. Namun, persyaratan Anda yang sebenarnya mungkin lebih tinggi atau lebih rendah.

Saat indeks sedang melakukan backfill, DynamoDB menggunakan kapasitas sistem internal untuk membaca dari tabel. Hal ini untuk meminimalkan dampak pembuatan indeks dan untuk memastikan bahwa meja Anda tidak kehabisan kapasitas baca.

Namun, ada kemungkinan bahwa volume aktivitas tulis yang masuk mungkin melebihi kapasitas tulis yang disediakan dari indeks. Ini adalah skenario bottleneck, di mana pembuatan indeks membutuhkan lebih banyak waktu karena aktivitas penulisan ke indeks dibatasi. Selama pembuatan indeks, kami sarankan Anda memantau Amazon CloudWatch metrik indeks guna menentukan apakah kapasitas tulis yang digunakan melebihi kapasitas yang disediakan. Dalam skenario bottleneck, Anda harus meningkatkan kapasitas tulis yang disediakan pada indeks untuk menghindari throttling penulisan selama fase backfilling.

Setelah indeks dibuat, Anda harus mengatur kapasitas tulis yang disediakan untuk mencerminkan penggunaan normal aplikasi Anda.

Menghapus Indeks Sekunder Global

Jika Anda tidak lagi memerlukan indeks sekunder global, Anda dapat menghapusnya menggunakan operasi `UpdateTable`.

Anda hanya dapat menghapus satu indeks sekunder global per operasi `UpdateTable`.

Saat indeks sekunder global sedang dihapus, tidak ada efek pada aktivitas baca atau tulis apa pun di tabel induk. Saat penghapusan sedang berlangsung, Anda masih dapat mengubah throughput yang disediakan pada indeks lain.

Note

- Saat Anda menghapus tabel menggunakan tindakan `DeleteTable`, semua indeks sekunder global pada tabel tersebut juga dihapus.
- Akun Anda tidak akan dikenakan biaya untuk operasi penghapusan indeks sekunder global.

Memodifikasi Indeks Sekunder Global selama Pembuatan

Saat indeks sedang dibangun, Anda dapat menggunakan operasi `DescribeTable` untuk menentukan fase apa itu. Deskripsi untuk indeks menyertakan atribut `Boolean`, `Backfilling`, untuk menunjukkan apakah DynamoDB saat ini memuat indeks dengan item dari tabel. Jika nilai `Backfilling` adalah benar, fase alokasi sumber daya selesai dan indeks sekarang dilakukan `backfilling`.

Saat `backfilling` sedang berlangsung, Anda dapat memperbarui parameter throughput yang disediakan untuk indeks. Anda mungkin memutuskan untuk melakukan ini untuk mempercepat pembuatan indeks: Anda dapat meningkatkan kapasitas tulis indeks saat sedang dibangun, dan kemudian mengurangnya sesudahnya. Untuk mengubah pengaturan throughput indeks yang disediakan, gunakan operasi `UpdateTable`. Status indeks berubah menjadi `UPDATING`, dan `Backfilling` benar hingga indeks siap digunakan.

Selama fase `backfilling`, Anda dapat menghapus indeks yang sedang dibuat. Selama fase ini, Anda tidak dapat menambah atau menghapus indeks lain di tabel.

Note

Untuk indeks yang dibuat sebagai bagian dari operasi `CreateTable`, atribut `Backfilling` tidak muncul di output `DescribeTable`. Untuk informasi selengkapnya, lihat [Fase Pembuatan Indeks](#).

Mendeteksi dan mengoreksi pelanggaran kunci indeks

Selama fase backfilling pembuatan indeks sekunder global, Amazon DynamoDB memeriksa setiap item dalam tabel untuk menentukan apakah item tersebut memenuhi syarat untuk dimasukkan dalam indeks. Beberapa item mungkin tidak memenuhi syarat karena akan menyebabkan pelanggaran kunci indeks. Dalam kasus ini, item tetap berada di tabel, tetapi indeks tidak memiliki entri yang sesuai untuk item tersebut.

Pelanggaran kunci indeks terjadi dalam situasi berikut:

- Ada ketidakcocokan tipe data antara nilai atribut dan tipe data skema kunci indeks. Misalnya, salah satu item dalam tabel `GameScores` memiliki nilai `TopScore` tipe `String`. Jika Anda menambahkan indeks sekunder global dengan kunci partisi `TopScore`, tipe `Number`, item dari tabel akan melanggar kunci indeks.
- Nilai atribut dari tabel melebihi panjang maksimum untuk atribut kunci indeks. Panjang maksimum kunci partisi adalah 2048 byte, dan panjang maksimum kunci pengurutan adalah 1024 byte. Jika salah satu nilai atribut yang sesuai dalam tabel melebihi batas ini, item dari tabel akan melanggar kunci indeks.

Note

Jika nilai atribut `String` atau `Biner` ditetapkan untuk atribut yang digunakan sebagai kunci indeks, maka nilai atribut harus memiliki panjang lebih besar dari nol; jika tidak, item dari tabel akan melanggar kunci indeks.

Alat ini tidak menandai pelanggaran kunci indeks ini, saat ini.

Jika pelanggaran kunci indeks terjadi, fase backfilling berlanjut tanpa gangguan. Namun, item yang melanggar tidak termasuk dalam indeks. Setelah fase backfilling selesai, semua penulisan ke item yang melanggar skema kunci indeks baru akan ditolak.

Untuk mengidentifikasi dan memperbaiki nilai atribut dalam tabel yang melanggar kunci indeks, gunakan alat Detektor Pelanggaran. Untuk menjalankan Detektor Pelanggaran, Anda membuat file konfigurasi yang menentukan nama tabel yang akan dipindai, nama dan tipe data kunci partisi indeks sekunder global dan kunci pengurutan, dan tindakan apa yang harus diambil jika ditemukan pelanggaran kunci indeks. Detektor Pelanggaran dapat berjalan dalam salah satu dari dua mode berbeda:

- Mode deteksi — Mendeteksi pelanggaran kunci indeks. Menggunakan mode deteksi untuk melaporkan item dalam tabel yang akan menyebabkan pelanggaran kunci dalam indeks sekunder global. (Secara opsional, Anda dapat meminta agar item tabel yang melanggar ini segera dihapus saat ditemukan.) Output dari mode deteksi ditulis ke file, yang dapat Anda gunakan untuk analisis lebih lanjut.
- Mode koreksi — Memperbaiki pelanggaran kunci indeks. Dalam mode koreksi, Detektor Pelanggaran membaca file input dengan format yang sama dengan file output dari mode deteksi. Mode koreksi membaca catatan dari file input dan, untuk setiap catatan, mode ini menghapus atau memperbarui item yang sesuai dalam tabel. (Perhatikan bahwa jika Anda memilih untuk memperbarui item, Anda harus mengedit file input dan menetapkan nilai yang sesuai untuk pembaruan ini.)

Mengunduh dan menjalankan Detektor Pelanggaran

Detektor Pelanggaran tersedia sebagai Arsip Java (file `.jar`) yang dapat dijalankan, dan berjalan di komputer Windows, macOS, atau Linux. Detektor Pelanggaran membutuhkan Java 1.7 (atau lebih baru) dan Apache Maven.

- [Unduh Detektor Pelanggaran dari GitHub](#)

Ikuti instruksi dalam file `README.md` untuk mengunduh dan menginstal Detektor Pelanggaran menggunakan Maven.

Untuk memulai Detektor Pelanggaran, buka direktori tempat Anda membangun `ViolationDetector.java` dan masukkan perintah berikut.

```
java -jar ViolationDetector.jar [options]
```

Baris perintah Detektor Pelanggaran menerima opsi berikut:

- `-h` | `--help` — Mencetak ringkasan penggunaan dan opsi untuk Detektor Pelanggaran.
- `-p` | `--configFilePath value` — Nama yang memenuhi syarat dari file konfigurasi Detektor Pelanggaran. Untuk informasi selengkapnya, lihat [File konfigurasi Detektor Pelanggaran](#).
- `-t` | `--detect value` — Mendeteksi pelanggaran kunci indeks dalam tabel, dan menuliskannya ke file output Detektor Pelanggaran. Jika nilai parameter ini diatur ke `keep`, item dengan pelanggaran kunci tidak diubah. Jika nilai diatur ke `delete`, item dengan pelanggaran kunci dihapus dari tabel.

- `-c | --correct value` — Membaca pelanggaran kunci indeks dari file input, dan mengambil tindakan korektif pada item dalam tabel. Jika nilai parameter ini diatur ke `update`, item dengan pelanggaran utama diperbarui dengan nilai baru yang tidak melanggar. Jika nilai diatur ke `delete`, item dengan pelanggaran kunci dihapus dari tabel.

File konfigurasi Detektor Pelanggaran

Saat waktu aktif, alat Detektor Pelanggaran memerlukan file konfigurasi. Parameter dalam file ini menentukan sumber daya DynamoDB mana yang dapat diakses oleh Detektor Pelanggaran, dan berapa banyak throughput yang disediakan yang dapat digunakan oleh sumber daya tersebut. Tabel berikut menjelaskan parameter ini.

Nama parameter	Deskripsi	Wajib?
<code>awsCredentialsFile</code>	Nama file yang sepenuhnya memenuhi syarat yang berisi kredensial AWS Anda. File kredensial harus dalam format berikut: <div style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; margin: 10px 0;"> <pre>accessKey = access_key_id_goes_here secretKey = secret_key_goes_here</pre> </div>	Ya
<code>dynamoDBRegion</code>	Wilayah AWS tempat tabel berada. Misalnya: <code>us-west-2</code> .	Ya
<code>tableName</code>	Nama tabel DynamoDB yang akan dipindai.	Ya
<code>gsiHashKeyName</code>	Nama kunci partisi indeks.	Ya
<code>gsiHashKeyType</code>	Tipe data kunci partisi indeks —String, Number, atau Binary:	Ya

Nama parameter	Deskripsi	Wajib?
	S N B	
gsiRangeKeyName	Nama kunci pengurutan indeks. Jangan tentukan parameter ini jika indeks hanya memiliki kunci primer sederhana (kunci partisi).	Tidak
gsiRangeKeyType	Tipe data dari kunci pengurutan indeks— <code>String</code> , <code>Number</code> , atau <code>Binary</code> : S N B Jangan tentukan parameter ini jika indeks hanya memiliki kunci primer sederhana (kunci partisi).	Tidak
recordDetails	Apakah akan menulis rincian lengkap pelanggaran kunci indeks ke file output. Jika diatur ke <code>true</code> (default), informasi lengkap tentang item yang melanggar akan dilaporkan. Jika diatur ke <code>false</code> , hanya jumlah pelanggaran yang dilaporkan.	Tidak
recordGsiValueInViolationRecord	Apakah akan menulis nilai kunci indeks yang melanggar ke file output. Jika diatur ke <code>true</code> (default), nilai kunci dilaporkan. Jika diatur ke <code>false</code> , nilai kunci tidak dilaporkan.	Tidak

Nama parameter	Deskripsi	Wajib?
detectionOutputPath	<p>Jalur lengkap file output Detektor Pelanggaran. Parameter ini mendukung penulisan ke direktori lokal atau ke Amazon Simple Storage Service (Amazon S3). Berikut ini adalah contohnya:</p> <pre>detectionOutputPath = //local/path/filename.csv</pre> <pre>detectionOutputPath = s3://bucket/filename.csv</pre> <p>Informasi dalam file output muncul dalam format nilai yang dipisahkan koma (CSV). Jika Anda tidak mengatur <code>detectionOutputPath</code>, file outputnya dinamai <code>violation_detection.csv</code> dan ditulis ke direktori kerja Anda saat ini.</p>	Tidak

Nama parameter	Deskripsi	Wajib?
numOfSegments	<p>Jumlah segmen pemindaian paralel yang akan digunakan saat Detektor Pelanggaran memindai tabel. Nilai default adalah 1, artinya tabel dipindai secara berurutan. Jika nilainya 2 atau lebih tinggi, Detektor Pelanggaran membagi tabel menjadi banyak segmen logis dan jumlah utas pemindaian yang sama.</p> <p>Pengaturan maksimum untuk <code>numOfSegments</code> adalah 4096.</p> <p>Untuk tabel yang lebih besar, pemindaian paralel umumnya lebih cepat daripada pemindaian berurutan. Selain itu, jika tabel cukup besar untuk menjangkau beberapa partisi, pemindaian paralel mendistribusikan aktivitas bacanya secara merata di beberapa partisi.</p> <p>Untuk informasi selengkapnya tentang pemindaian paralel di DynamoDB, lihat Pemindaian paralel.</p>	Tidak

Nama parameter	Deskripsi	Wajib?
<code>numOfViolations</code>	Batas atas pelanggaran kunci indeks untuk menulis ke file output. Jika diatur ke -1 (default), seluruh tabel dipindai. Jika diatur ke bilangan bulat positif, Detektor Pelanggaran akan berhenti setelah menemukan jumlah pelanggaran tersebut.	Tidak
<code>numOfRecords</code>	Jumlah item dalam tabel yang akan dipindai. Jika diatur ke -1 (default), seluruh tabel dipindai. Jika diatur ke bilangan bulat positif, Detektor Pelanggaran berhenti setelah memindai sejumlah item itu dalam tabel.	Tidak
<code>readWriteIOPSPercent</code>	Mengatur persentase unit kapasitas baca yang disediakan yang digunakan selama pemindaian tabel. Nilai yang valid berkisar dari 1 sampai 100. Nilai default (25) berarti bahwa Detektor Pelanggaran akan menggunakan tidak lebih dari 25% dari throughput baca yang disediakan tabel.	Tidak

Nama parameter	Deskripsi	Wajib?
<code>correctionInputPath</code>	<p>Jalur lengkap file input koreksi Detektor Pelanggaran. Jika Anda menjalankan Detektor Pelanggaran dalam mode koreksi, isi file ini digunakan untuk mengubah atau menghapus item data dalam tabel yang melanggar indeks sekunder global.</p> <p>Format file <code>correctionInputPath</code> sama dengan format file <code>detectionOutputPath</code>. Ini memungkinkan Anda memproses output dari mode deteksi sebagai input dalam mode koreksi.</p>	Tidak

Nama parameter	Deskripsi	Wajib?
<code>correctionOutputPath</code>	<p>Jalur lengkap file output koreksi Detektor Pelanggaran. File ini dibuat hanya jika ada kesalahan pembaruan.</p> <p>Parameter ini mendukung penulisan ke direktori lokal atau ke Amazon S3. Berikut ini adalah contohnya:</p> <pre>correctionOutputPath = //local/path/ filename.csv</pre> <pre>correctionOutputPath = s3://bucket/filename. csv</pre> <p>Informasi dalam file output muncul dalam format CSV. Jika Anda tidak mengatur <code>correctionOutputPath</code>, file outputnya dinamai <code>violation_update_errors.csv</code> dan ditulis ke direktori kerja Anda saat ini.</p>	Tidak

Deteksi

Untuk mendeteksi pelanggaran kunci indeks, gunakan Detektor Pelanggaran dengan opsi baris perintah `--detect`. Untuk menunjukkan cara kerja opsi ini, perhatikan tabel `ProductCatalog` yang ditampilkan di [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#). Berikut ini adalah daftar item dalam tabel. Hanya kunci primer (`Id`) dan atribut `Price` yang ditampilkan.

Id (kunci utama)	Harga
101	5
102	20
103	200
201	100
202	200
203	300
204	400
205	500

Semua nilai untuk `Price` bertipe `Number`. Namun, karena DynamoDB tidak memiliki skema, Anda dapat menambahkan item dengan `Price` non-numerik. Misalnya, Anda menambahkan item lain ke tabel `ProductCatalog`.

Id (kunci utama)	Harga
999	"Hello"

Tabel sekarang memiliki total sembilan item.

Sekarang Anda menambahkan indeks sekunder global baru ke tabel: `PriceIndex`. Kunci primer untuk indeks ini adalah kunci partisi, `Price`, yang merupakan tipe `Number`. Setelah indeks dibuat, indeks akan berisi delapan item—tetapi tabel `ProductCatalog` memiliki sembilan item. Alasan perbedaan ini adalah bahwa nilai "Hello" adalah tipe `String`, tetapi `PriceIndex` memiliki kunci primer tipe `Number`. Nilai `String` melanggar kunci indeks sekunder global, sehingga tidak ada dalam indeks.

Untuk menggunakan Detektor Pelanggaran dalam skenario ini, Anda terlebih dahulu membuat file konfigurasi seperti berikut ini.

```
# 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
```

Selanjutnya, Anda menjalankan Detektor Pelanggaran seperti dalam contoh berikut.

```
$ 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
```

Jika parameter konfigurasi `recordDetails` diatur ke `true`, Detektor Pelanggaran menulis detail setiap pelanggaran ke file output, seperti dalam contoh berikut.

```
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,,
```

File output dalam format CSV. Baris pertama dalam file adalah header, diikuti oleh satu rekaman per item yang melanggar kunci indeks. Bidang catatan pelanggaran tersebut adalah sebagai berikut:

- Kunci hash tabel— Nilai kunci partisi dari item dalam tabel.
- Kunci rentang tabel— Nilai kunci pengurutan item dalam tabel.

- Nilai kunci hash GSI— Nilai kunci partisi dari indeks sekunder global.
- Jenis pelanggaran kunci hash GSI— Baik `Type Violation` atau `Size Violation`.
- Deskripsi pelanggaran kunci hash GSI— Penyebab pelanggaran.
- Nilai pembaruan kunci hash GSI (UNTUK PENGGUNA)— Dalam mode koreksi, nilai baru yang diberikan pengguna untuk atribut tersebut.
- Nilai kunci rentang GSI— Nilai kunci pengurutan dari indeks sekunder global.
- Jenis pelanggaran kunci rentang GSI— Baik `Type Violation` atau `Size Violation`.
- Deskripsi pelanggaran kunci rentang GSI— Penyebab pelanggaran.
- Nilai pembaruan kunci rentang GSI (UNTUK PENGGUNA)— Dalam mode koreksi, nilai baru yang diberikan pengguna untuk atribut tersebut.
- Hapus atribut kosong saat Memperbarui (Y/N)— Dalam mode koreksi, menentukan apakah akan menghapus (Y) atau menyimpan (N) item yang melanggar dalam tabel—tetapi hanya jika salah satu bidang berikut ini kosong:
 - GSI Hash Key Update Value(FOR USER)
 - GSI Range Key Update Value(FOR USER)

Jika salah satu dari bidang ini tidak kosong, `Delete Blank Attribute When Updating(Y/N)` tidak berpengaruh.

Note

Format output mungkin berbeda, tergantung file konfigurasi dan opsi baris perintah. Misalnya, jika tabel memiliki kunci primer sederhana (tanpa kunci pengurutan), tidak ada bidang kunci pengurutan yang akan ditampilkan dalam output.

Catatan pelanggaran dalam file mungkin tidak diurutkan.

Koreksi

Untuk memperbaiki pelanggaran kunci indeks, gunakan Detektor Pelanggaran dengan opsi baris perintah `--correct`. Dalam mode koreksi, Detektor Pelanggaran membaca file input yang ditentukan oleh parameter `correctionInputPath`. File ini memiliki format yang sama dengan file `detectionOutputPath`, sehingga Anda dapat menggunakan output dari deteksi sebagai input untuk koreksi.

Detektor Pelanggaran menyediakan dua cara berbeda untuk memperbaiki pelanggaran kunci indeks:

- Hapus pelanggaran — Menghapus item tabel yang memiliki nilai atribut yang melanggar.
- Perbarui pelanggaran — Memperbarui item tabel, mengganti atribut yang melanggar dengan nilai yang tidak melanggar.

Dalam kedua kasus, Anda dapat menggunakan file output dari mode deteksi sebagai input untuk mode koreksi.

Melanjutkan contoh `ProductCatalog`, misalkan Anda ingin menghapus item yang melanggar dari tabel. Untuk melakukannya, Anda menggunakan baris perintah berikut.

```
$ java -jar ViolationDetector.jar --configFilePath config.txt --correct delete
```

Pada tahap ini, Anda akan diminta untuk mengonfirmasi apakah Anda ingin menghapus item yang melanggar.

```
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
```

Sekarang baik `ProductCatalog` dan `PriceIndex` memiliki jumlah item yang sama.

Menggunakan Indeks Sekunder Global: Java

Anda dapat menggunakan API Dokumen AWS SDK for Java untuk membuat tabel Amazon DynamoDB dengan satu atau beberapa indeks sekunder global, mendeskripsikan indeks pada tabel, dan melakukan kueri menggunakan indeks.

Berikut ini adalah langkah-langkah umum untuk operasi tabel.

1. Buat instans kelas `DynamoDB`.
2. Berikan parameter wajib dan opsional untuk operasi dengan membuat objek permintaan yang sesuai.
3. Panggil metode sesuai yang ditentukan oleh klien yang Anda buat pada langkah sebelumnya.

Topik

- [Membuat tabel dengan Indeks Sekunder Global](#)
- [Mendeskripsikan tabel dengan Indeks Sekunder Global](#)
- [Mengkueri Indeks Sekunder Global](#)
- [Contoh: Indeks Sekunder Global menggunakan API dokumen AWS SDK for Java](#)

Membuat tabel dengan Indeks Sekunder Global

Anda dapat membuat indeks sekunder global pada saat membuat tabel. Untuk melakukannya, gunakan `CreateTable` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder global. Contoh kode Java berikut membuat tabel untuk menyimpan informasi tentang data cuaca. Kunci partisinya adalah `Location` dan kunci urutannya adalah `Date`. Indeks sekunder global bernama `PrecipIndex` memungkinkan akses cepat ke data curah hujan untuk berbagai lokasi.

Berikut adalah langkah-langkah untuk membuat tabel dengan indeks sekunder global, menggunakan API dokumen DynamoDB.

1. Buat instans kelas `DynamoDB`.
2. Buat instans kelas `CreateTableRequest` untuk memberikan informasi permintaan.

Anda harus memberikan nama tabel, kunci primernya, dan nilai throughput yang ditentukan. Untuk indeks sekunder global, Anda harus memberikan nama indeks, pengaturan throughput yang ditentukan, definisi atribut untuk kunci urutan indeks, skema kunci untuk indeks, dan proyeksi atribut.

3. Panggil metode `createTable` dengan menentukan objek permintaan sebagai parameter.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya. Kode tersebut membuat tabel (`WeatherData`) dengan indeks sekunder global (`PrecipIndex`). Kunci partisi indeksnya adalah `Date` dan kunci urutannya adalah `Precipitation`. Semua atribut tabel diproyeksikan ke dalam indeks. Pengguna dapat mengkueri indeks ini untuk mendapatkan data cuaca untuk tanggal tertentu, yang secara opsional mengurutkan data berdasarkan jumlah curah hujan.

Karena `Precipitation` bukan atribut kunci untuk tabel tersebut, maka tidak diperlukan. Namun, item `WeatherData` tanpa `Precipitation` tidak muncul di `PrecipIndex`.

```
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());
```

Anda harus menunggu hingga DynamoDB membuat tabel dan menetapkan status tabel menjadi ACTIVE. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel.

Mendeskripsikan tabel dengan Indeks Sekunder Global

Untuk mendapatkan informasi tentang indeks sekunder global pada sebuah tabel, gunakan `DescribeTable`. Untuk setiap indeks, Anda dapat mengakses namanya, skema kunci, dan atribut yang diproyeksikan.

Berikut ini adalah langkah-langkah untuk mengakses informasi indeks sekunder global sebuah tabel.

1. Buat instans kelas `DynamoDB`.
2. Buat instans kelas `Table` untuk mewakili indeks yang ingin Anda gunakan.
3. Panggil metode `describe` pada objek `Table`.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya.

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());
    }
}
```

Mengkueri Indeks Sekunder Global

Anda dapat menggunakan Query pada indeks sekunder global, sama seperti Anda Query tabel. Anda perlu menentukan nama indeks, kriteria kueri untuk kunci partisi indeks dan kunci urutan (jika ada), dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeksnya adalah `PrecipIndex`, yang memiliki kunci partisi `Date` dan kunci urutan `Precipitation`. Kueri indeks mengembalikan semua data cuaca untuk tanggal tertentu, dengan curah hujan lebih besar dari nol.

Berikut ini adalah langkah-langkah untuk mengkueri indeks sekunder global menggunakan API Dokumen AWS SDK for Java.

1. Buat instans kelas `DynamoDB`.
2. Buat instans kelas `Table` untuk mewakili indeks yang ingin Anda gunakan.
3. Buat instans kelas `Index` untuk indeks yang ingin Anda kueri.
4. Panggil metode `query` pada objek `Index`.

Nama atribut `Date` adalah kata yang disimpan DynamoDB. Oleh karena itu, Anda harus menggunakan nama atribut ekspresi sebagai placeholder di `KeyConditionExpression`.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya.

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());
}
```

Contoh: Indeks Sekunder Global menggunakan API dokumen AWS SDK for Java

Contoh kode Java berikut menunjukkan cara menggunakan indeks sekunder global. Contoh tersebut membuat tabel bernama `Issues`, yang dapat digunakan dalam sistem pelacakan bug sederhana untuk pengembangan perangkat lunak. Kunci partisinya adalah `IssueId` dan kunci urutannya adalah `Title`. Ada tiga indeks sekunder global pada tabel ini:

- `CreateDateIndex` — Kunci partisinya adalah `CreateDate` dan kunci urutannya adalah `IssueId`. Selain kunci tabel, atribut `Description` dan `Status` diproyeksikan ke dalam indeks.
- `TitleIndex` — Kunci partisinya adalah `Title` dan kunci urutannya adalah `IssueId`. Tidak ada atribut selain kunci tabel yang diproyeksikan ke dalam indeks.
- `DueDateIndex` — Kunci partisinya adalah `DueDate`, dan tidak ada kunci urutan. Semua atribut tabel diproyeksikan ke dalam indeks.

Setelah tabel `Issues` dibuat, program memuat tabel dengan data yang mewakili laporan bug perangkat lunak. Kemudian, data dikueri menggunakan indeks sekunder global. Terakhir, program menghapus tabel `Issues`.

Untuk step-by-step instruksi untuk menguji contoh berikut, lihat [Contoh kode Java](#).

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);
    }
}
```

Menggunakan Indeks Sekunder Global: .NET

Anda dapat menggunakan API tingkat rendah AWS SDK for .NET untuk membuat tabel Amazon DynamoDB dengan satu atau beberapa indeks sekunder global, mendeskripsikan indeks pada

tabel, dan melakukan kueri menggunakan indeks. Operasi ini dipetakan ke operasi DynamoDB yang sesuai. Untuk informasi selengkapnya, lihat [Referensi API Amazon DynamoDB](#).

Berikut ini adalah langkah-langkah umum untuk operasi tabel menggunakan API tingkat rendah .NET.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Berikan parameter wajib dan opsional untuk operasi dengan membuat objek permintaan yang sesuai.

Misalnya, buat objek `CreateTableRequest` untuk membuat tabel dan objek `QueryRequest` untuk mengkueri tabel atau indeks.

3. Jalankan metode sesuai yang ditentukan oleh klien yang Anda buat pada langkah sebelumnya.

Topik

- [Membuat tabel dengan Indeks Sekunder Global](#)
- [Mendeskripsikan tabel dengan Indeks Sekunder Global](#)
- [Mengkueri Indeks Sekunder Global](#)
- [Contoh: Indeks Sekunder Global menggunakan API tingkat rendah AWS SDK for .NET](#)

Membuat tabel dengan Indeks Sekunder Global

Anda dapat membuat indeks sekunder global pada saat membuat tabel. Untuk melakukannya, gunakan `CreateTable` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder global. Contoh kode C# berikut membuat tabel untuk menyimpan informasi tentang data cuaca. Kunci partisinya adalah `Location` dan kunci urutannya adalah `Date`. Indeks sekunder global bernama `PrecipIndex` memungkinkan akses cepat ke data curah hujan untuk berbagai lokasi.

Berikut ini adalah langkah-langkah untuk membuat tabel dengan indeks sekunder global, menggunakan API tingkat rendah .NET.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Buat instans kelas `CreateTableRequest` untuk memberikan informasi permintaan.

Anda harus memberikan nama tabel, kunci primernya, dan nilai throughput yang ditentukan. Untuk indeks sekunder global, Anda harus memberikan nama indeks, pengaturan throughput yang ditentukan, definisi atribut untuk kunci urutan indeks, skema kunci untuk indeks, dan proyeksi atribut.

3. Jalankan metode CreateTable dengan menyediakan objek permintaan sebagai parameter.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya. Kode tersebut membuat tabel (WeatherData) dengan indeks sekunder global (PrecipIndex). Kunci partisi indeksnya adalah Date dan kunci urutannya adalah Precipitation. Semua atribut tabel diproyeksikan ke dalam indeks. Pengguna dapat mengkueri indeks ini untuk mendapatkan data cuaca untuk tanggal tertentu, yang secara opsional mengurutkan data berdasarkan jumlah curah hujan.

Karena Precipitation bukan atribut kunci untuk tabel tersebut, maka tidak diperlukan. Namun, item WeatherData tanpa Precipitation tidak muncul di PrecipIndex.

```
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);
```

Anda harus menunggu hingga DynamoDB membuat tabel dan menetapkan status tabel menjadi ACTIVE. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel.

Mendeskripsikan tabel dengan Indeks Sekunder Global

Untuk mendapatkan informasi tentang indeks sekunder global pada sebuah tabel, gunakan `DescribeTable`. Untuk setiap indeks, Anda dapat mengakses namanya, skema kunci, dan atribut yang diproyeksikan.

Berikut ini adalah langkah-langkah untuk mengakses informasi indeks sekunder global untuk tabel menggunakan API tingkat rendah .NET.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Jalankan metode `describeTable` dengan menyediakan objek permintaan sebagai parameter.

Buat instans kelas `DescribeTableRequest` untuk memberikan informasi permintaan. Anda harus memberikan nama tabel.

3.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya.

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);
    }
}
```

Mengkueri Indeks Sekunder Global

Anda dapat menggunakan Query pada indeks sekunder global, sama seperti Anda Query tabel. Anda perlu menentukan nama indeks, kriteria kueri untuk kunci partisi indeks dan kunci urutan (jika ada), dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeksnya adalah `PrecipIndex`, yang memiliki kunci partisi `Date` dan kunci urutan `Precipitation`. Kueri indeks mengembalikan semua data cuaca untuk tanggal tertentu, dengan curah hujan lebih besar dari nol.

Berikut ini adalah langkah-langkah untuk mengkueri indeks sekunder global menggunakan API tingkat rendah .NET.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Buat instans kelas `QueryRequest` untuk memberikan informasi permintaan.
3. Jalankan metode `query` dengan menentukan objek permintaan sebagai parameter.

Nama atribut `Date` adalah kata yang disimpan DynamoDB. Oleh karena itu, Anda harus menggunakan nama atribut ekspresi sebagai placeholder di `KeyConditionExpression`.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya.

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();
}
```

Contoh: Indeks Sekunder Global menggunakan API tingkat rendah AWS SDK for .NET

Contoh kode C# berikut menunjukkan cara menggunakan indeks sekunder global. Contoh tersebut membuat tabel bernama *Issues*, yang dapat digunakan dalam sistem pelacakan bug sederhana untuk pengembangan perangkat lunak. Kunci partisinya adalah *IssueId* dan kunci urutannya adalah *Title*. Ada tiga indeks sekunder global pada tabel ini:

- **CreateDateIndex** — Kunci partisinya adalah *CreateDate* dan kunci urutannya adalah *IssueId*. Selain kunci tabel, atribut *Description* dan *Status* diproyeksikan ke dalam indeks.
- **TitleIndex** — Kunci partisinya adalah *Title* dan kunci urutannya adalah *IssueId*. Tidak ada atribut selain kunci tabel yang diproyeksikan ke dalam indeks.
- **DueDateIndex** — Kunci partisinya adalah *DueDate*, dan tidak ada kunci urutan. Semua atribut tabel diproyeksikan ke dalam indeks.

Setelah tabel Issues dibuat, program memuat tabel dengan data yang mewakili laporan bug perangkat lunak. Kemudian, data dikueri menggunakan indeks sekunder global. Terakhir, program menghapus tabel Issues.

Untuk step-by-step petunjuk pengujian sampel berikut, lihat [Contoh kode .NET](#).

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;
    }
}
}
```

Bekerja dengan Indeks Sekunder Global:AWS CLI

Anda dapat menggunakan AWS CLI untuk membuat tabel Amazon DynamoDB dengan satu atau beberapa indeks sekunder global, menjelaskan indeks pada tabel, dan melakukan kueri menggunakan indeks.

Topik

- [Membuat tabel dengan Indeks Sekunder Global](#)
- [Menambahkan Indeks Sekunder Global ke sebuah tabel yang ada](#)
- [Menjelaskan tabel dengan Indeks Sekunder Global](#)
- [Mengkueri Indeks Sekunder Global](#)

Membuat tabel dengan Indeks Sekunder Global

Indeks sekunder global dapat dibuat pada saat yang sama Anda membuat tabel. Untuk melakukannya, gunakan parameter `create-table` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder global. Contoh berikut membuat tabel bernama `GameScores` dengan indeks sekunder global yang disebut `GameTitleIndex`. Tabel dasar memiliki kunci partisi `UserId` dan kunci pengurutan `GameTitle`, memungkinkan Anda menemukan skor terbaik pengguna individu untuk game tertentu secara efisien, sedangkan GSI memiliki kunci partisi `GameTitle` dan kunci pengurutan `TopScore`, memungkinkan Anda untuk cepat menemukan skor tertinggi secara keseluruhan untuk game tertentu.

```
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
      }
    }
  ]"
```

Anda harus menunggu sampai DynamoDB membuat tabel dan menetapkan status tabelnya menjadi ACTIVE. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel. Anda dapat menggunakan [deskripsi-tabel](#) untuk menentukan status pembuatan tabel.

Menambahkan Indeks Sekunder Global ke sebuah tabel yang ada

Indeks sekunder global juga dapat ditambahkan atau dimodifikasi setelah pembuatan tabel. Untuk melakukannya, gunakan parameter `update-table` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder global. Contoh berikut menggunakan skema yang sama seperti contoh sebelumnya, tetapi mengasumsikan bahwa tabel telah dibuat dan kami akan menambahkan GSI nanti.

```
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\"]
      }
    }
  }
]"
```

Menjelaskan tabel dengan Indeks Sekunder Global

Untuk mendapatkan informasi tentang Global pada sebuah tabel, gunakan `describe-table` parameter. Untuk setiap indeks, Anda dapat mengakses nama, skema kunci, dan atribut yang diproyeksikan.

```
aws dynamodb describe-table --table-name GameScores
```

Mengkueri Indeks Sekunder Global

Anda dapat menggunakan operasi query pada indeks sekunder global dengan cara yang hampir sama seperti Anda query tabel. Anda harus menentukan nama indeks, kriteria kueri untuk kunci pengurutan indeks, dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeks adalah `GameTitleIndex` dan kunci pengurutan indeks adalah `GameTitle`.

Satu-satunya atribut yang dikembalikan adalah atribut yang telah diproyeksikan ke dalam indeks. Anda dapat memodifikasi kueri ini untuk memilih atribut non-kunci juga, tetapi ini akan memerlukan aktivitas pengambilan tabel yang relatif mahal. Untuk informasi selengkapnya tentang pengambilan tabel, lihat [Proyeksi atribut](#).

```
aws dynamodb query --table-name GameScores \
  --index-name GameTitleIndex \
  --key-condition-expression "GameTitle = :v_game" \
  --expression-attribute-values '{":v_game":{"S":"Alien Adventure"}}'
```

Indeks Sekunder Lokal

Beberapa aplikasi hanya perlu meminta data menggunakan kunci primer tabel dasar. Namun, mungkin ada situasi di mana kunci pengurutan alternatif akan membantu. Untuk memberi aplikasi Anda pilihan kunci pengurutan, Anda dapat membuat satu atau beberapa indeks sekunder lokal pada tabel Amazon DynamoDB dan mengeluarkan permintaan Query atau Scan terhadap indeks ini.

Topik

- [Skenario: Menggunakan Indeks Sekunder Lokal](#)
- [Proyeksi atribut](#)
- [Membuat Indeks Sekunder Lokal](#)
- [Membaca data dari Indeks Sekunder Lokal](#)
- [Item menulis dan Indeks Sekunder Lokal](#)
- [Pertimbangan throughput yang disediakan untuk Indeks Sekunder Lokal](#)
- [Pertimbangan penyimpanan untuk Indeks Sekunder Lokal](#)
- [Koleksi item dalam Indeks Sekunder Lokal](#)
- [Menggunakan Indeks Sekunder Lokal: Java](#)
- [Menggunakan Indeks Sekunder Lokal: .NET](#)
- [Bekerja dengan Indeks Sekunder Lokal: AWS CLI](#)

Skenario: Menggunakan Indeks Sekunder Lokal

Sebagai contoh, perhatikan tabel Thread yang didefinisikan dalam [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#). Tabel ini berguna untuk aplikasi seperti [AWSforum diskusi](#). Diagram berikut menunjukkan bagaimana item dalam tabel akan diatur. (Tidak semua atribut ditampilkan.)

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	...
...

DynamoDB menyimpan semua item dengan nilai kunci partisi yang sama secara terus menerus. Dalam contoh ini, diberikan `ForumName` tertentu, operasi `Query` dapat segera menemukan semua utas untuk forum itu. Dalam grup item dengan nilai kunci partisi yang sama, item diurutkan berdasarkan nilai kunci pengurutan. Jika kunci pengurutan (`Subject`) juga disediakan dalam kueri, DynamoDB dapat mempersempit hasil yang dikembalikan—misalnya, mengembalikan semua utas di forum "S3" yang memiliki tanda `Subject` yang diawali dengan huruf "a".

Beberapa permintaan mungkin memerlukan pola akses data yang lebih kompleks. Misalnya:

- Utas forum mana yang paling banyak dilihat dan dibalas?
- Utas mana di forum tertentu yang memiliki jumlah pesan terbanyak?
- Berapa banyak utas yang diposting di forum tertentu dalam jangka waktu tertentu?

Untuk menjawab pertanyaan-pertanyaan ini, tindakan `Query` tidak akan cukup. Sebaliknya, Anda harus `Scan` seluruh tabel. Untuk tabel dengan jutaan item, ini akan menghabiskan banyak `throughput` baca yang disediakan dan membutuhkan waktu lama untuk menyelesaikannya.

Namun, Anda dapat menentukan satu atau beberapa indeks sekunder lokal pada atribut non-kunci, seperti `Replies` atau `LastPostDateTime`.

Indeks sekunder lokal mempertahankan kunci pengurutan alternatif untuk nilai kunci partisi yang diberikan. Indeks sekunder lokal juga berisi salinan beberapa atau semua atribut dari tabel dasarnya. Anda menentukan atribut mana yang diproyeksikan ke indeks sekunder lokal saat Anda membuat tabel. Data dalam indeks sekunder lokal diatur oleh kunci partisi yang sama dengan tabel dasar,

tetapi dengan kunci pengurutan yang berbeda. Ini memungkinkan Anda mengakses item data secara efisien di seluruh dimensi yang berbeda ini. Untuk fleksibilitas kueri atau pemindaian yang lebih baik, Anda dapat membuat hingga lima indeks sekunder lokal per tabel.

Misalkan sebuah aplikasi perlu menemukan semua utas yang telah diposting dalam tiga bulan terakhir di forum tertentu. Tanpa indeks sekunder lokal, aplikasi harus Scan seluruh tabel Thread dan membuang posting apa pun yang tidak berada dalam kerangka waktu yang ditentukan. Dengan indeks sekunder lokal, operasi Query dapat menggunakan LastPostDateTime sebagai kunci pengurutan dan menemukan data dengan cepat.

Diagram berikut menunjukkan indeks sekunder lokal bernama LastPostIndex. Perhatikan bahwa kunci partisi sama dengan tabel Thread, tetapi kunci pengurutan adalah LastPostDateTime.

LastPostIndex

<i>ForumName</i>	<i>LastPostDateTime</i>	<i>Subject</i>
"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"
...

Setiap indeks sekunder lokal harus memenuhi syarat berikut:

- Kunci partisi sama dengan tabel dasarnya.
- Kunci pengurutan terdiri dari tepat satu atribut skalar.

- Kunci pengurutan dari tabel dasar diproyeksikan ke dalam indeks, di mana kunci tersebut bertindak sebagai atribut non-kunci.

Dalam contoh ini, kunci partisi adalah `ForumName` dan kunci pengurutan dari indeks sekunder lokal adalah `LastPostDateTime`. Selain itu, nilai kunci pengurutan dari tabel dasar (dalam contoh ini, `Subject`) diproyeksikan ke dalam indeks, tetapi bukan merupakan bagian dari kunci indeks. Jika aplikasi membutuhkan daftar yang didasarkan pada `ForumName` dan `LastPostDateTime`, aplikasi dapat mengeluarkan permintaan Query terhadap `LastPostIndex`. Hasil kueri diurutkan menurut `LastPostDateTime`, dan dapat dikembalikan dalam urutan menaik atau menurun. Kueri juga dapat menerapkan syarat kunci, seperti hanya mengembalikan item yang memiliki `LastPostDateTime` dalam rentang waktu tertentu.

Setiap indeks sekunder lokal secara otomatis berisi kunci partisi dan pengurutan dari tabel dasarnya; Anda dapat secara opsional memproyeksikan atribut non-kunci ke dalam indeks. Saat Anda membuat kueri indeks, DynamoDB dapat mengambil atribut yang diproyeksikan ini secara efisien. Saat Anda membuat kueri indeks sekunder lokal, kueri juga dapat mengambil atribut yang tidak diproyeksikan ke dalam indeks. DynamoDB secara otomatis mengambil atribut ini dari tabel dasar, tetapi pada latensi yang lebih besar dan dengan biaya throughput yang disediakan lebih tinggi.

Untuk indeks sekunder lokal apa pun, Anda dapat menyimpan hingga 10 GB data per nilai kunci partisi yang berbeda. Angka ini mencakup semua item dalam tabel dasar, ditambah semua item dalam indeks, yang memiliki nilai kunci partisi yang sama. Untuk informasi selengkapnya, lihat [Koleksi item dalam Indeks Sekunder Lokal](#).

Proyeksi atribut

Dengan `LastPostIndex`, sebuah aplikasi dapat menggunakan `ForumName` dan `LastPostDateTime` sebagai kriteria kueri. Namun, untuk mengambil atribut tambahan apa pun, DynamoDB harus melakukan operasi baca tambahan terhadap tabel `Thread`. Pembacaan ekstra ini dikenal sebagai pengambilan, dan pembacaan tersebut dapat meningkatkan jumlah total throughput yang disediakan yang diperlukan untuk kueri.

Misalkan Anda ingin mengisi halaman web dengan daftar semua utas di "S3" dan jumlah balasan untuk setiap utas, diurutkan berdasarkan tanggal/waktu balasan terakhir yang dimulai dengan balasan terbaru. Untuk mengisi daftar ini, Anda memerlukan atribut berikut:

- `Subject`
- `Replies`

- LastPostDateTime

Cara paling efisien untuk menanyakan data ini dan untuk menghindari operasi pengambilan adalah dengan memproyeksikan atribut Replies dari tabel ke dalam indeks sekunder lokal, seperti yang ditunjukkan dalam diagram ini.

LastPostIndex

<i>ForumName</i>	<i>LastPostDateTime</i>	<i>Subject</i>	<i>Replies</i>
"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
...

Proyeksi adalah kumpulan atribut yang disalin dari tabel ke indeks sekunder. Kunci partisi dan kunci pengurutan tabel selalu diproyeksikan ke dalam indeks; Anda dapat memproyeksikan atribut lain untuk mendukung persyaratan kueri aplikasi Anda. Saat Anda membuat kueri indeks, Amazon DynamoDB dapat mengakses atribut apa pun dalam proyeksi seolah-olah atribut tersebut ada dalam tabelnya sendiri.

Saat Anda membuat indeks sekunder, Anda perlu menentukan atribut yang akan diproyeksikan ke dalam indeks. DynamoDB menyediakan tiga opsi berbeda untuk ini:

- **KEYS_ONLY** – Setiap item dalam indeks hanya terdiri dari kunci partisi tabel dan nilai kunci pengurutan, ditambah nilai kunci indeks. Opsi **KEYS_ONLY** menghasilkan indeks sekunder sekecil mungkin.
- **INCLUDE** – Selain atribut yang dijelaskan dalam **KEYS_ONLY**, indeks sekunder akan menyertakan atribut non-kunci lainnya yang Anda tentukan.
- **ALL** – Indeks sekunder mencakup semua atribut dari tabel sumber. Karena semua data tabel diduplikasi dalam indeks, proyeksi **ALL** menghasilkan indeks sekunder terbesar yang mungkin.

Pada diagram sebelumnya, atribut non-kunci `Replies` diproyeksikan ke dalam `LastPostIndex`. Aplikasi dapat meminta `LastPostIndex` sebagai ganti dari tabel `Thread` lengkap untuk mengisi halaman web dengan `Subject`, `Replies`, dan `LastPostDateTime`. Jika ada atribut non-kunci lain yang diminta, DynamoDB perlu mengambil atribut tersebut dari tabel `Thread`.

Dari sudut pandang aplikasi, pengambilan atribut tambahan dari tabel dasar dilakukan secara otomatis dan transparan, jadi tidak perlu menulis ulang logika aplikasi apa pun. Namun, pengambilan tersebut dapat sangat mengurangi keuntungan performa menggunakan indeks sekunder lokal.

Saat Anda memilih atribut untuk diproyeksikan ke dalam indeks sekunder lokal, Anda harus mempertimbangkan tradeoff antara biaya throughput yang disediakan dan biaya penyimpanan:

- Jika Anda hanya perlu mengakses beberapa atribut dengan latensi serendah mungkin, pertimbangkan untuk hanya memproyeksikan atribut tersebut ke dalam indeks sekunder lokal. Semakin kecil indeks, semakin sedikit biaya untuk menyimpannya, dan semakin sedikit biaya tulis Anda. Jika ada atribut yang terkadang perlu Anda ambil, biaya untuk throughput yang disediakan mungkin lebih besar daripada biaya jangka panjang untuk menyimpan atribut tersebut.
- Jika aplikasi Anda sering mengakses beberapa atribut non-kunci, Anda harus mempertimbangkan untuk memproyeksikan atribut tersebut ke dalam indeks sekunder lokal. Biaya penyimpanan tambahan untuk indeks sekunder lokal mengimbangi biaya sering melakukan pemindaian tabel.
- Jika Anda perlu sering mengakses sebagian besar atribut non-kunci, Anda dapat memproyeksikan atribut ini—atau bahkan seluruh tabel dasar—ke dalam indeks sekunder lokal. Ini memberi Anda fleksibilitas maksimum dan penggunaan persediaan throughput terendah, karena pengambilan tidak diperlukan. Namun, biaya penyimpanan Anda akan meningkat, atau bahkan dua kali lipat jika Anda memproyeksikan semua atribut.
- Jika aplikasi Anda jarang melakukan kueri tabel, tetapi harus melakukan banyak penulisan atau pembaruan terhadap data dalam tabel, pertimbangkan untuk memproyeksikan **KEYS_ONLY**.

Indeks sekunder lokal berukuran minimal, tetapi akan tetap tersedia bila diperlukan untuk aktivitas kueri.

Membuat Indeks Sekunder Lokal

Untuk membuat satu atau beberapa indeks sekunder lokal pada tabel, gunakan parameter `LocalSecondaryIndexes` dari operasi `CreateTable`. Indeks sekunder lokal pada tabel dibuat saat tabel dibuat. Saat Anda menghapus tabel, indeks sekunder lokal apa pun di tabel itu juga akan dihapus.

Anda harus menentukan satu atribut non-kunci untuk bertindak sebagai kunci pengurutan dari indeks sekunder lokal. Atribut yang Anda pilih harus berupa skalar `String`, `Number`, atau `Binary`. Jenis skalar, jenis dokumen, dan jenis kumpulan lainnya tidak diperbolehkan. Untuk daftar lengkap tipe data, lihat [Tipe Data](#).

Important

Untuk tabel dengan indeks sekunder lokal, ada batas ukuran 10 GB per nilai kunci partisi. Tabel dengan indeks sekunder lokal dapat menyimpan sejumlah item, selama ukuran total untuk satu nilai kunci partisi tidak melebihi 10 GB. Untuk informasi selengkapnya, lihat [Batas ukuran koleksi barang](#).

Anda dapat memproyeksikan atribut tipe data apa pun ke dalam indeks sekunder lokal. Ini termasuk skalar, dokumen, dan kumpulan. Untuk daftar lengkap tipe data, lihat [Tipe Data](#).

Membaca data dari Indeks Sekunder Lokal

Anda dapat mengambil item dari indeks sekunder lokal menggunakan operasi `Query` dan `Scan`. Operasi `GetItem` dan `BatchGetItem` tidak dapat digunakan pada indeks sekunder lokal.

Mengkueri Indeks Sekunder Lokal

Dalam tabel DynamoDB, nilai kunci partisi gabungan dan nilai kunci pengurutan untuk setiap item harus unik. Namun, dalam indeks sekunder lokal, nilai kunci pengurutan tidak perlu unik untuk nilai kunci partisi yang diberikan. Jika ada beberapa item dalam indeks sekunder lokal yang memiliki nilai kunci pengurutan yang sama, operasi `Query` mengembalikan semua item yang memiliki nilai kunci partisi yang sama. Sebagai tanggapan, item yang cocok tidak dikembalikan dalam urutan tertentu.

Anda dapat melakukan kueri indeks sekunder lokal menggunakan bacaan akhir konsisten atau bacaan konsisten yang kuat. Untuk menentukan jenis konsistensi yang Anda inginkan, gunakan parameter `ConsistentRead` dari operasi `Query`. Bacaan konsisten yang kuat dari indeks sekunder lokal selalu mengembalikan nilai terbaru yang diperbarui. Jika kueri perlu mengambil atribut tambahan dari tabel dasar, atribut tersebut akan konsisten dengan indeks.

Example

Pertimbangkan data berikut yang dikembalikan dari `Query` yang meminta data dari utas diskusi di forum tertentu.

```
{
  "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"}
  }
}
```

Dalam kueri ini:

- DynamoDB mengakses `LastPostIndex`, menggunakan kunci partisi `ForumName` untuk menemukan item indeks untuk "EC2". Semua item indeks dengan kunci ini disimpan berdekatan satu sama lain untuk pengambilan cepat.
- Dalam forum ini, DynamoDB menggunakan indeks untuk mencari kunci yang cocok dengan syarat `LastPostDateTime` yang ditentukan.
- Karena atribut `Replies` diproyeksikan ke dalam indeks, DynamoDB dapat mengambil atribut ini tanpa menggunakan throughput tambahan yang disediakan.
- Atribut `Tags` tidak diproyeksikan ke dalam indeks, jadi DynamoDB harus mengakses tabel `Thread` dan mengambil atribut ini.
- Hasilnya dikembalikan, diurutkan berdasarkan `LastPostDateTime`. Entri indeks diurutkan menurut nilai kunci partisi dan kemudian menurut nilai kunci pengurutan, dan `Query`

mengembalikannya sesuai urutan penyimpanannya. (Anda dapat menggunakan parameter `ScanIndexForward` untuk mengembalikan hasil dalam urutan menurun.)

Karena atribut `Tags` tidak diproyeksikan ke indeks sekunder lokal, DynamoDB harus menggunakan unit kapasitas baca tambahan untuk mengambil atribut ini dari tabel dasar. Jika Anda perlu sering menjalankan kueri ini, Anda harus memproyeksikan `Tags` ke `LastPostIndex` untuk menghindari pengambilan dari tabel dasar. Namun, jika Anda hanya perlu mengakses `Tags` sesekali, biaya penyimpanan tambahan untuk memproyeksikan `Tags` ke dalam indeks mungkin tidak bermanfaat.

Memindai Indeks Sekunder Lokal

Anda dapat menggunakan `Scan` untuk mengambil semua data dari indeks sekunder lokal. Anda harus memberikan nama tabel dasar dan nama indeks dalam permintaan. Dengan `Scan`, DynamoDB membaca semua data dalam indeks dan mengembalikannya ke aplikasi. Anda juga dapat meminta agar hanya sebagian data yang dikembalikan, dan data yang tersisa harus dibuang. Untuk melakukan ini, gunakan parameter `FilterExpression` dari `Scan` API. Untuk informasi selengkapnya, lihat [Ekspresi filter untuk pemindaian](#).

Item menulis dan Indeks Sekunder Lokal

DynamoDB secara otomatis membuat semua indeks sekunder lokal tersinkronisasi dengan tabel dasar masing-masing. Aplikasi tidak pernah menulis langsung ke file indeks. Namun, penting bagi Anda untuk memahami implikasi dari cara DynamoDB mempertahankan indeks ini.

Saat Anda membuat indeks sekunder lokal, Anda menentukan atribut untuk dijadikan sebagai kunci pengurutan untuk indeks. Anda juga menentukan tipe data untuk atribut tersebut. Ini berarti bahwa setiap kali Anda menulis item ke tabel dasar, jika item tersebut mendefinisikan atribut kunci indeks, tipenya harus cocok dengan tipe data skema kunci indeks. Dalam kasus `LastPostIndex`, kunci pengurutan `LastPostDateTime` dalam indeks didefinisikan sebagai tipe data `String`. Jika Anda mencoba menambahkan item ke tabel `Thread` dan menentukan tipe data yang berbeda untuk `LastPostDateTime` (seperti `Number`), DynamoDB mengembalikan `ValidationException` karena tipe data tidak cocok.

Tidak ada persyaratan untuk one-to-one hubungan antara item dalam tabel dasar dan item dalam indeks sekunder lokal. Bahkan, perilaku ini dapat menguntungkan untuk banyak aplikasi.

Tabel dengan banyak indeks sekunder lokal menimbulkan biaya lebih tinggi untuk aktivitas tulis daripada tabel dengan indeks lebih sedikit. Untuk informasi selengkapnya, lihat [Pertimbangan throughput yang disediakan untuk Indeks Sekunder Lokal](#).

⚠ Important

Untuk tabel dengan indeks sekunder lokal, ada batas ukuran 10 GB per nilai kunci partisi. Tabel dengan indeks sekunder lokal dapat menyimpan sejumlah item, selama ukuran total untuk satu nilai kunci partisi tidak melebihi 10 GB. Untuk informasi selengkapnya, lihat [Batas ukuran koleksi barang](#).

Pertimbangan throughput yang disediakan untuk Indeks Sekunder Lokal

Saat Anda membuat tabel di DynamoDB, Anda menyediakan unit kapasitas baca dan tulis untuk beban kerja tabel yang diharapkan. Beban kerja tersebut mencakup aktivitas baca dan tulis pada indeks sekunder lokal tabel.

Untuk melihat tarif saat ini untuk kapasitas throughput yang disediakan, lihat [harga Amazon DynamoDB](#).

Baca unit kapasitas

Saat Anda membuat kueri indeks sekunder lokal, jumlah unit kapasitas baca yang digunakan bergantung pada cara data diakses.

Seperti kueri tabel, kueri indeks dapat menggunakan bacaan akhir konsisten atau bacaan konsisten yang kuat bergantung pada nilai `ConsistentRead`. Satu bacaan konsisten yang kuat menghabiskan satu unit kapasitas baca; bacaan akhir konsisten hanya menghabiskan setengah dari itu. Jadi, dengan memilih bacaan akhir konsisten, Anda dapat mengurangi biaya unit kapasitas baca Anda.

Untuk kueri indeks yang hanya meminta kunci indeks dan atribut yang diproyeksikan, DynamoDB menghitung aktivitas baca yang disediakan dengan cara yang sama seperti kueri terhadap tabel. Satu-satunya perbedaan adalah bahwa perhitungan didasarkan pada ukuran entri indeks, bukan ukuran item dalam tabel dasar. Jumlah unit kapasitas baca adalah jumlah dari semua ukuran atribut yang diproyeksikan di semua item yang dikembalikan; hasilnya kemudian dibulatkan ke batas 4 KB berikutnya. Untuk informasi selengkapnya tentang cara DynamoDB menghitung penggunaan throughput yang disediakan, lihat [Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB](#).

Untuk kueri indeks yang membaca atribut yang tidak diproyeksikan ke indeks sekunder lokal, DynamoDB perlu mengambil atribut tersebut dari tabel dasar, selain membaca atribut yang diproyeksikan dari indeks. Pengambilan ini terjadi saat Anda menyertakan atribut yang tidak

diproyeksikan dalam parameter `Select` atau `ProjectionExpression` dari operasi `Query`. Pengambilan menyebabkan latensi tambahan dalam respons kueri, dan juga menimbulkan biaya throughput yang disediakan lebih tinggi: Selain pembacaan dari indeks sekunder lokal yang dijelaskan sebelumnya, Anda dikenakan biaya untuk unit kapasitas baca untuk setiap item tabel dasar yang diambil. Biaya ini untuk membaca setiap item dari tabel, bukan hanya atribut yang diminta.

Ukuran maksimum hasil yang dikembalikan oleh operasi `Query` adalah 1 MB. Ini termasuk ukuran semua nama dan nilai atribut di semua item yang dikembalikan. Namun, jika Kueri terhadap indeks sekunder lokal menyebabkan DynamoDB mengambil atribut item dari tabel dasar, ukuran maksimum data dalam hasil mungkin lebih rendah. Dalam hal ini, ukuran hasil adalah jumlah dari:

- Ukuran item yang cocok dalam indeks, dibulatkan ke atas hingga 4 KB berikutnya.
- Ukuran setiap item yang cocok di tabel dasar, dengan setiap item dibulatkan satu per satu ke 4 KB berikutnya.

Menggunakan rumus ini, ukuran maksimum hasil yang dikembalikan oleh operasi Kueri masih 1 MB.

Misalnya, pertimbangkan tabel di mana ukuran setiap item adalah 300 byte. Ada indeks sekunder lokal di tabel itu, tetapi hanya 200 byte dari setiap item yang diproyeksikan ke dalam indeks. Sekarang anggaplah Anda `Query` indeks ini, bahwa kueri memerlukan pengambilan tabel untuk setiap item, dan kueri mengembalikan 4 item. DynamoDB merangkum hal berikut:

- Ukuran item yang cocok dalam indeks: $200 \text{ byte} \times 4 \text{ item} = 800 \text{ byte}$; ini kemudian dibulatkan menjadi 4 KB.
- Ukuran setiap item yang cocok di tabel dasar: $(300 \text{ byte, dibulatkan hingga } 4 \text{ KB}) \times 4 \text{ item} = 16 \text{ KB}$.

Oleh karena itu, ukuran total data dalam hasil adalah 20 KB.

Menulis unit kapasitas

Saat item dalam tabel ditambahkan, diperbarui, atau dihapus, memperbarui indeks sekunder lokal menggunakan unit kapasitas tulis yang disediakan untuk tabel. Total biaya throughput yang disediakan untuk penulisan adalah jumlah unit kapasitas tulis yang digunakan dengan menulis ke tabel dan yang digunakan dengan memperbarui indeks sekunder lokal.

Biaya menulis item ke indeks sekunder lokal bergantung pada beberapa faktor:

- Jika Anda menulis item baru ke tabel yang menentukan atribut yang diindeks, atau Anda memperbarui item yang ada untuk menentukan atribut terindeks yang sebelumnya tidak ditentukan, satu operasi tulis diperlukan untuk memasukkan item ke dalam indeks.
- Jika pembaruan ke tabel mengubah nilai atribut kunci yang diindeks (dari A ke B), dua penulisan diperlukan: satu untuk menghapus item sebelumnya dari indeks dan penulisan lainnya untuk memasukkan item baru ke dalam indeks.
- Jika item ada dalam indeks, tetapi penulisan ke tabel menyebabkan atribut yang diindeks dihapus, satu penulisan diperlukan untuk menghapus proyeksi item lama dari indeks.
- Jika item tidak ada dalam indeks sebelum atau setelah item diperbarui, tidak ada biaya penulisan tambahan untuk indeks.

Semua faktor ini mengasumsikan bahwa ukuran setiap item dalam indeks kurang dari atau sama dengan ukuran item 1 KB untuk menghitung unit kapasitas tulis. Entri indeks yang lebih besar memerlukan unit kapasitas tulis tambahan. Anda dapat meminimalkan biaya penulisan dengan mempertimbangkan atribut mana yang perlu dikembalikan oleh kueri Anda dan hanya memproyeksikan atribut tersebut ke dalam indeks.

Pertimbangan penyimpanan untuk Indeks Sekunder Lokal

Saat aplikasi menulis item ke tabel, DynamoDB secara otomatis menyalin subset atribut yang benar ke indeks sekunder lokal tempat atribut tersebut akan muncul. Akun AWS Anda dikenakan biaya untuk penyimpanan item di tabel dasar dan juga untuk penyimpanan atribut di indeks sekunder lokal apa pun di tabel itu.

Jumlah ruang yang digunakan oleh item indeks adalah jumlah dari berikut ini:

- Ukuran dalam byte dari kunci primer tabel dasar (kunci partisi dan kunci pengurutan)
- Ukuran dalam byte atribut kunci indeks
- Ukuran dalam byte dari atribut yang diproyeksikan (jika ada)
- 100 byte dari overhead per item indeks

Untuk memperkirakan persyaratan penyimpanan untuk indeks sekunder lokal, Anda dapat memperkirakan ukuran rata-rata item dalam indeks dan kemudian mengalikannya dengan jumlah item dalam indeks.

Jika tabel berisi item di mana atribut tertentu tidak ditentukan, tetapi atribut tersebut didefinisikan sebagai kunci pengurutan indeks, maka DynamoDB tidak menulis data apa pun untuk item tersebut ke indeks.

Koleksi item dalam Indeks Sekunder Lokal

Note

Bagian ini hanya berkaitan dengan tabel yang memiliki indeks sekunder lokal.

Di DynamoDB, kumpulan item adalah grup item apa pun yang memiliki nilai kunci partisi yang sama dalam tabel dan semua indeks sekunder lokalnya. Dalam contoh yang digunakan di seluruh bagian ini, kunci partisi untuk tabel Thread adalah ForumName, dan kunci partisi untuk LastPostIndex juga ForumName. Semua item tabel dan indeks dengan ForumName yang sama adalah bagian dari kumpulan item yang sama. Misalnya, pada tabel Thread dan indeks sekunder lokal LastPostIndex, terdapat kumpulan item untuk forum EC2 dan kumpulan item yang berbeda untuk forum RDS.

Diagram berikut menunjukkan kumpulan item untuk forum S3.

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
...

Dalam diagram ini, kumpulan item terdiri dari semua item di Thread dan LastPostIndex di mana nilai kunci partisi ForumName adalah "S3". Jika ada indeks sekunder lokal lainnya di atas meja, item apa pun dalam indeks tersebut dengan ForumName sama dengan "S3" juga akan menjadi bagian dari kumpulan item.

Anda dapat menggunakan salah satu operasi berikut di DynamoDB untuk mengembalikan informasi tentang kumpulan item:

- BatchWriteItem
- DeleteItem
- PutItem
- UpdateItem
- TransactWriteItems

Masing-masing operasi ini mendukung parameter ReturnItemCollectionMetrics. Saat Anda mengatur parameter ini ke SIZE, Anda dapat melihat informasi tentang ukuran setiap kumpulan item dalam indeks.

Example

Berikut ini adalah contoh dari output operasi UpdateItem pada tabel Thread, dengan ReturnItemCollectionMetrics diatur ke SIZE. Item yang diperbarui memiliki nilai ForumName "EC2", jadi outputnya mencakup informasi tentang kumpulan item tersebut.

```
{
  ItemCollectionMetrics: {
    ItemCollectionKey: {
      ForumName: "EC2"
    },
    SizeEstimateRangeGB: [0.0, 1.0]
  }
}
```

Objek SizeEstimateRangeGB menunjukkan bahwa ukuran kumpulan item ini antara 0 dan 1 GB. DynamoDB memperbarui perkiraan ukuran ini secara berkala, sehingga jumlahnya mungkin berbeda saat item diubah lagi.

Batas ukuran koleksi barang

Ukuran maksimum koleksi item apa pun untuk tabel yang memiliki satu atau lebih indeks sekunder lokal adalah 10 GB. Ini tidak berlaku untuk koleksi item dalam tabel tanpa indeks sekunder lokal, dan juga tidak berlaku untuk koleksi item dalam indeks sekunder global. Hanya tabel yang memiliki satu atau lebih indeks sekunder lokal yang terpengaruh.

Jika koleksi item melebihi batas 10 GB, DynamoDB mengembalikan `ItemCollectionSizeLimitExceededException`, dan Anda tidak akan bisa menambahkan lebih banyak item ke koleksi item atau menambah ukuran item yang ada di kumpulan item. (Operasi baca dan tulis yang mengecilkan ukuran kumpulan item masih diperbolehkan.) Anda masih dapat menambahkan item ke kumpulan item lainnya.

Untuk mengurangi ukuran kumpulan item, Anda dapat melakukan salah satu hal berikut:

- Hapus item yang tidak perlu dengan nilai kunci partisi yang dimaksud. Saat Anda menghapus item ini dari tabel dasar, DynamoDB juga menghapus entri indeks yang memiliki nilai kunci partisi yang sama.
- Perbarui item dengan menghapus atribut atau dengan mengurangi ukuran atribut. Jika atribut ini diproyeksikan ke indeks sekunder lokal apa pun, DynamoDB juga mengurangi ukuran entri indeks terkait.
- Buat tabel baru dengan kunci partisi dan kunci pengurutan yang sama, lalu pindahkan item dari tabel lama ke tabel baru. Ini mungkin pendekatan yang baik jika tabel memiliki data historis yang jarang diakses. Anda juga dapat mempertimbangkan untuk mengarsipkan data historis ini ke Amazon Simple Storage Service (Amazon S3).

Ketika ukuran total kumpulan item turun di bawah 10 GB, Anda dapat sekali lagi menambahkan item dengan nilai kunci partisi yang sama.

Sebagai praktik terbaik, sebaiknya Anda melengkapi aplikasi Anda untuk memantau ukuran kumpulan item Anda. Salah satu cara untuk melakukannya adalah dengan mengatur parameter `ReturnItemCollectionMetrics` ke `SIZE` setiap kali Anda menggunakan `BatchWriteItem`, `DeleteItem`, `PutItem`, atau `UpdateItem`. Aplikasi Anda harus memeriksa objek `ReturnItemCollectionMetrics` dalam output dan mencatat pesan kesalahan setiap kali koleksi item melebihi batas yang ditentukan pengguna (8 GB, misalnya). Menetapkan batas yang kurang dari 10 GB akan memberikan sistem peringatan dini sehingga Anda tahu bahwa koleksi item mendekati batas waktu untuk melakukan tindakan terhadap hal itu.

Koleksi dan partisi item

Dalam tabel dengan satu atau lebih indeks sekunder lokal, setiap koleksi item disimpan dalam satu partisi. Ukuran total koleksi barang semacam itu terbatas pada kemampuan partisi itu: 10 GB. Untuk aplikasi di mana model data menyertakan koleksi item yang ukurannya tidak terbatas, atau di mana Anda mungkin mengharapkan beberapa koleksi item tumbuh melebihi 10 GB di masa mendatang, Anda harus mempertimbangkan untuk menggunakan indeks sekunder global sebagai gantinya.

Anda harus merancang aplikasi Anda sehingga data tabel didistribusikan secara merata di seluruh nilai kunci partisi yang berbeda. Untuk tabel dengan indeks sekunder lokal, aplikasi Anda tidak boleh membuat "hot spot" aktivitas baca dan tulis dalam satu kumpulan item pada satu partisi.

Menggunakan Indeks Sekunder Lokal: Java

Anda dapat menggunakan API Dokumen AWS SDK for Java untuk membuat tabel Amazon DynamoDB dengan satu atau beberapa indeks sekunder lokal, mendeskripsikan indeks pada tabel, dan melakukan kueri menggunakan indeks.

Berikut ini adalah langkah-langkah umum untuk operasi tabel menggunakan API Dokumen AWS SDK for Java.

1. Buat instans kelas DynamoDB.
2. Berikan parameter wajib dan opsional untuk operasi dengan membuat objek permintaan yang sesuai.
3. Panggil metode sesuai yang ditentukan oleh klien yang Anda buat pada langkah sebelumnya.

Topik

- [Membuat tabel dengan Indeks Sekunder Lokal](#)
- [Mendeskripsikan tabel dengan Indeks Sekunder Lokal](#)
- [Mengkueri Indeks Sekunder Lokal](#)
- [Contoh: Indeks Sekunder Lokal menggunakan API dokumen Java](#)

Membuat tabel dengan Indeks Sekunder Lokal

Indeks sekunder lokal harus dibuat pada saat Anda membuat tabel. Untuk melakukannya, gunakan metode `createTable` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder lokal. Contoh kode Java berikut membuat tabel untuk menyimpan informasi tentang lagu dalam koleksi

musik. Kunci partisinya adalah `Artist` dan kunci urutannya adalah `SongTitle`. Indeks sekunder, `AlbumTitleIndex`, memfasilitasi kueri berdasarkan judul album.

Berikut adalah langkah-langkah untuk membuat tabel dengan indeks sekunder lokal, menggunakan API dokumen DynamoDB.

1. Buat instans kelas `DynamoDB`.
2. Buat instans kelas `CreateTableRequest` untuk memberikan informasi permintaan.

Anda harus memberikan nama tabel, kunci primernya, dan nilai throughput yang ditentukan. Untuk indeks sekunder lokal, Anda harus memberikan nama indeks, nama dan jenis data untuk kunci urutan indeks, skema kunci untuk indeks, dan proyeksi atribut.

3. Panggil metode `createTable` dengan menentukan objek permintaan sebagai parameter.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya. Kode ini membuat tabel (`Music`) dengan indeks sekunder pada atribut `AlbumTitle`. Kunci urutan dan kunci partisi tabel, ditambah kunci urutan indeks, adalah satu-satunya atribut yang diproyeksikan ke dalam indeks.

```
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());
```

Anda harus menunggu hingga DynamoDB membuat tabel dan menetapkan status tabel menjadi ACTIVE. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel.

Mendeskripsikan tabel dengan Indeks Sekunder Lokal

Untuk mendapatkan informasi tentang indeks sekunder lokal pada tabel, gunakan metode `describeTable`. Untuk setiap indeks, Anda dapat mengakses namanya, skema kunci, dan atribut yang diproyeksikan.

Berikut ini adalah langkah-langkah untuk mengakses informasi indeks sekunder lokal suatu tabel menggunakan API Dokumen AWS SDK for Java.

1. Buat instans kelas `DynamoDB`.
2. Buat instans kelas `Table`. Anda harus memberikan nama tabel.
3. Panggil metode `describeTable` pada objek `Table`.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya.

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());
}
}
```

Mengkueri Indeks Sekunder Lokal

Anda dapat menggunakan operasi Query pada indeks sekunder lokal dengan cara yang hampir sama seperti Anda Query tabel. Anda harus menentukan nama indeks, kriteria kueri untuk kunci urutan indeks, dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeks adalah AlbumTitleIndex dan kunci urutan indeks adalah AlbumTitle.

Satu-satunya atribut yang dikembalikan adalah atribut yang telah diproyeksikan ke dalam indeks. Anda dapat memodifikasi kueri ini untuk memilih atribut non-kunci juga, tetapi ini akan memerlukan aktivitas pengambilan tabel yang relatif mahal. Untuk informasi selengkapnya tentang pengambilan tabel, lihat [Proyeksi atribut](#).

Berikut ini adalah langkah-langkah untuk mengkueri indeks sekunder lokal menggunakan API Dokumen AWS SDK for Java.

1. Buat instans kelas DynamoDB.
2. Buat instans kelas Table. Anda harus memberikan nama tabel.
3. Buat instans kelas Index. Anda harus memberikan nama indeks.
4. Panggil metode query dari kelas Index.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya.

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());
}
```

Contoh: Indeks Sekunder Lokal menggunakan API dokumen Java

Contoh kode Java berikut menunjukkan cara menggunakan indeks sekunder lokal di Amazon DynamoDB. Contoh membuat tabel bernama `CustomerOrders` dengan kunci partisi `CustomerId` dan kunci urutan `OrderId`. Ada dua indeks sekunder lokal di tabel ini:

- `OrderCreationDateIndex` — Kunci urutannya adalah `OrderCreationDate`, dan atribut berikut diproyeksikan ke dalam indeks:
 - `ProductCategory`
 - `ProductName`
 - `OrderStatus`
 - `ShipmentTrackingId`
- `IsOpenIndex` — Kunci urutannya adalah `IsOpen`, dan semua atribut tabel diproyeksikan ke dalam indeks.

Setelah tabel `CustomerOrders` dibuat, program memuat tabel dengan data yang mewakili pesanan pelanggan. Kemudian mengkueri data menggunakan indeks sekunder lokal. Terakhir, program menghapus tabel `CustomerOrders`.

Untuk step-by-step petunjuk pengujian sampel berikut, lihat [Contoh kode Java](#).

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;
}
}
```


Menggunakan Indeks Sekunder Lokal: .NET

Topik

- [Membuat tabel dengan Indeks Sekunder Lokal](#)
- [Mendeskripsikan tabel dengan Indeks Sekunder Lokal](#)
- [Mengkueri Indeks Sekunder Lokal](#)
- [Contoh: Indeks Sekunder Lokal menggunakan API tingkat rendah AWS SDK for .NET](#)

Anda dapat menggunakan API tingkat rendah AWS SDK for .NET untuk membuat tabel Amazon DynamoDB dengan satu atau beberapa indeks sekunder lokal, mendeskripsikan indeks pada tabel, dan melakukan kueri menggunakan indeks. Operasi ini dipetakan ke tindakan API DynamoDB tingkat rendah yang sesuai. Untuk informasi selengkapnya, lihat [Contoh kode .NET](#).

Berikut ini adalah langkah-langkah umum untuk operasi tabel menggunakan API tingkat rendah .NET.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Berikan parameter wajib dan opsional untuk operasi dengan membuat objek permintaan yang sesuai.

Misalnya, buat objek `CreateTableRequest` untuk membuat tabel dan objek `QueryRequest` untuk mengkueri tabel atau indeks.

3. Jalankan metode sesuai yang ditentukan oleh klien yang Anda buat pada langkah sebelumnya.

Membuat tabel dengan Indeks Sekunder Lokal

Indeks sekunder lokal harus dibuat pada saat Anda membuat tabel. Untuk melakukannya, gunakan `CreateTable` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder lokal. Contoh kode C# berikut membuat tabel untuk menyimpan informasi tentang lagu dalam koleksi musik. Kunci partisinya adalah `Artist` dan kunci urutannya adalah `SongTitle`. Indeks sekunder, `AlbumTitleIndex`, memfasilitasi kueri berdasarkan judul album.

Berikut ini adalah langkah-langkah untuk membuat tabel dengan indeks sekunder lokal, menggunakan API tingkat rendah .NET.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Buat instans kelas `CreateTableRequest` untuk memberikan informasi permintaan.

Anda harus memberikan nama tabel, kunci primernya, dan nilai throughput yang ditentukan. Untuk indeks sekunder lokal, Anda harus memberikan nama indeks, nama dan jenis data kunci urutan indeks, skema kunci untuk indeks, dan proyeksi atribut.

3. Jalankan metode `CreateTable` dengan menentukan objek permintaan sebagai parameter.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya. Kode ini membuat tabel (`Music`) dengan indeks sekunder pada atribut `AlbumTitle`. Kunci urutan dan kunci partisi tabel, ditambah kunci urutan indeks, adalah satu-satunya atribut yang diproyeksikan ke dalam indeks.

```
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);
```

Anda harus menunggu hingga DynamoDB membuat tabel dan menetapkan status tabel menjadi ACTIVE. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel.

Mendeskripsikan tabel dengan Indeks Sekunder Lokal

Untuk mendapatkan informasi tentang indeks sekunder lokal pada tabel, gunakan API `DescribeTable`. Untuk setiap indeks, Anda dapat mengakses namanya, skema kunci, dan atribut yang diproyeksikan.

Berikut ini adalah langkah-langkah untuk mengakses informasi indeks sekunder lokal suatu tabel menggunakan API tingkat rendah .NET.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Buat instans kelas `DescribeTableRequest` untuk memberikan informasi permintaan. Anda harus memberikan nama tabel.
3. Jalankan metode `describeTable` dengan menentukan objek permintaan sebagai parameter.
- 4.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya.

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);
    }
}
}
```

Mengkueri Indeks Sekunder Lokal

Anda dapat menggunakan Query pada indeks sekunder lokal, sama seperti Anda Query tabel. Anda harus menentukan nama indeks, kriteria kueri untuk kunci urutan indeks, dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeksnya adalah `AlbumTitleIndex`, dan kunci urutan indeksnya adalah `AlbumTitle`.

Satu-satunya atribut yang dikembalikan adalah atribut yang telah diproyeksikan ke dalam indeks. Anda dapat memodifikasi kueri ini untuk memilih atribut non-kunci juga, tetapi ini akan memerlukan aktivitas pengambilan tabel yang relatif mahal. Untuk informasi selengkapnya tentang pengambilan tabel, lihat [Proyeksi atribut](#)

Berikut ini adalah langkah-langkah untuk mengkueri indeks sekunder lokal menggunakan API tingkat rendah .NET.

1. Buat instans kelas `AmazonDynamoDBClient`.
2. Buat instans kelas `QueryRequest` untuk memberikan informasi permintaan.
3. Jalankan metode `query` dengan menentukan objek permintaan sebagai parameter.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya.

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();
}
```

Contoh: Indeks Sekunder Lokal menggunakan API tingkat rendah AWS SDK for .NET

Contoh kode C# berikut menunjukkan cara menggunakan indeks sekunder lokal di Amazon DynamoDB. Contoh membuat tabel bernama `CustomerOrders` dengan kunci partisi `CustomerId` dan kunci urutan `OrderId`. Ada dua indeks sekunder lokal di tabel ini:

- `OrderCreationDateIndex` — Kunci urutannya adalah `OrderCreationDate`, dan atribut berikut diproyeksikan ke dalam indeks:
 - `ProductCategory`
 - `ProductName`
 - `OrderStatus`
 - `ShipmentTrackingId`
- `IsOpenIndex` — Kunci urutannya adalah `IsOpen`, dan semua atribut tabel diproyeksikan ke dalam indeks.

Setelah tabel `CustomerOrders` dibuat, program memuat tabel dengan data yang mewakili pesanan pelanggan. Kemudian mengkueri data menggunakan indeks sekunder lokal. Terakhir, program menghapus tabel `CustomerOrders`.

Untuk step-by-step instruksi untuk menguji contoh berikut, lihat [Contoh kode .NET](#).

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;
        }
    }
}
```

```

    }
  }
}

```

Bekerja dengan Indeks Sekunder Lokal: AWS CLI

Anda dapat menggunakan AWS CLI untuk membuat tabel Amazon DynamoDB dengan satu atau beberapa Indeks Sekunder Lokal, menjelaskan indeks pada tabel, dan melakukan kueri menggunakan indeks.

Topik

- [Membuat tabel dengan Indeks Sekunder Lokal Lokal](#)
- [Menjelaskan tabel dengan Indeks Sekunder Lokal Lokal](#)
- [Mengkueri Indeks Sekunder Lokal](#)

Membuat tabel dengan Indeks Sekunder Lokal Lokal

Indeks Sekunder Lokal harus dibuat pada saat yang sama Anda membuat tabel. Untuk melakukannya, gunakan `create-table` parameter dan berikan spesifikasi Anda untuk satu atau beberapa Indeks Sekunder Lokal Lokal. Contoh berikut membuat tabel (`Music`) untuk menyimpan informasi tentang lagu dalam koleksi musik. Kunci partisinya adalah `Artist` dan kunci pengurutannya adalah `SongTitle`. Indeks sekunder, `AlbumTitleIndex` pada atribut `AlbumTitle` memfasilitasi kueri berdasarkan judul album.

```

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"]}]]"

```

Anda harus menunggu sampai DynamoDB membuat tabel dan menetapkan status tabelnya menjadi ACTIVE. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel. Anda dapat menggunakan [deskripsi-tabel](#) untuk menentukan status pembuatan tabel.

Menjelaskan tabel dengan Indeks Sekunder Lokal Lokal

Untuk mendapatkan informasi tentang indeks sekunder lokal pada tabel, gunakan parameter `describe-table`. Untuk setiap indeks, Anda dapat mengakses nama, skema kunci, dan atribut yang diproyeksikan.

```
aws dynamodb describe-table --table-name Music
```

Mengkueri Indeks Sekunder Lokal

Anda dapat menggunakan `query` operasi pada Indeks Sekunder Lokal dalam banyak cara yang sama bahwa Anda `query` sebuah meja. Anda harus menentukan nama indeks, kriteria kueri untuk kunci pengurutan indeks, dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeks adalah `AlbumTitleIndex` dan kunci pengurutan indeks adalah `AlbumTitle`.

Satu-satunya atribut yang dikembalikan adalah atribut yang telah diproyeksikan ke dalam indeks. Anda dapat memodifikasi kueri ini untuk memilih atribut non-kunci juga, tetapi ini akan memerlukan aktivitas pengambilan tabel yang relatif mahal. Untuk informasi selengkapnya tentang pengambilan tabel, lihat [Proyeksi atribut](#).

```
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"} }'
```

Mengelola alur kerja kompleks dengan DynamoDB Transactions

Transaksi Amazon DynamoDB menyederhanakan pengalaman pengembang dalam membuat terkoordinasi `all-or-nothing`, perubahan ke beberapa item baik di dalam maupun di seluruh tabel. Transaksi memberikan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) di DynamoDB, yang memungkinkan Anda menjaga kebenaran data dalam aplikasi Anda dengan lebih mudah.

Anda dapat menggunakan API baca dan tulis transaksional DynamoDB untuk mengelola alur kerja bisnis kompleks yang memerlukan penambahan, pembaruan, atau penghapusan beberapa item

sebagai operasi tunggal. all-or-nothing Misalnya, developer video game dapat memastikan bahwa profil pemain diperbarui dengan benar saat mereka bertukar item dalam game atau melakukan pembelian dalam game.

Dengan transaksi API tulis, Anda dapat mengelompokkan beberapa tindakan Put, Update, Delete, dan ConditionCheck. Anda kemudian dapat mengirimkan tindakan sebagai operasi TransactWriteItems tunggal yang baik berhasil atau gagal sebagai unit. Hal yang sama berlaku untuk beberapa tindakan Get, yang dapat Anda kelompokkan dan kirimkan sebagai operasi TransactGetItems tunggal.

Tidak ada biaya tambahan untuk mengaktifkan transaksi untuk tabel DynamoDB Anda. Anda hanya membayar untuk baca atau tulis yang merupakan bagian dari transaksi Anda. DynamoDB melakukan dua baca atau tulis mendasar dari setiap item dalam transaksi: satu untuk mempersiapkan transaksi dan satu untuk melakukan transaksi. Dua operasi baca/tulis yang mendasari ini terlihat di metrik Amazon CloudWatch Anda.

Untuk memulai DynamoDB transactions, unduh AWS SDK atau AWS Command Line Interface (AWS CLI) terbaru. Kemudian ikuti [Contoh DynamoDB Transactions](#).

Bagian berikut memberikan gambaran umum mendetail tentang API transaksi dan cara penggunaannya dalam DynamoDB.

Topik

- [Amazon DynamoDB Transactions: Cara kerjanya](#)
- [Menggunakan IAM dengan DynamoDB transactions](#)
- [Contoh DynamoDB Transactions](#)

Amazon DynamoDB Transactions: Cara kerjanya

Dengan transaksi Amazon DynamoDB, Anda dapat mengelompokkan beberapa tindakan bersama-sama dan mengirimkannya sebagai satu atau all-or-nothing TransactWriteItems operasi. TransactGetItems Bagian berikut menjelaskan operasi API, manajemen kapasitas, praktik terbaik, dan detail lainnya tentang menggunakan operasi transaksional di DynamoDB.

Topik

- [TransactWriteItems API](#)
- [TransactGetItems API](#)

- [Tingkat isolasi untuk DynamoDB transactions](#)
- [Penanganan konflik transaksi di DynamoDB](#)
- [Menggunakan API Transaksional di DynamoDB Accelerator \(DAX\)](#)
- [Manajemen kapasitas untuk transaksi](#)
- [Praktik terbaik untuk transactions](#)
- [Menggunakan API transaksional dengan tabel global](#)
- [Transaksi DynamoDB AWS Labs vs. perpustakaan klien transaksi](#)

TransactWriteItems API

`TransactWriteItems` adalah operasi penulisan sinkron dan idempoten yang mengelompokkan hingga 100 tindakan penulisan dalam satu operasi. `all-or-nothing` Tindakan ini dapat menargetkan hingga 100 item yang berbeda dalam satu atau lebih tabel DynamoDB dalam akun AWS yang sama dan di Wilayah yang sama. Ukuran agregat item dalam transaksi tidak dapat melebihi 4 MB. Tindakan tersebut diselesaikan secara atomik, sehingga semuanya berhasil atau tidak satu pun yang berhasil.

Note

- Operasi `TransactWriteItems` berbeda dari operasi `BatchWriteItem` dalam hal semua tindakan yang di dalamnya harus berhasil diselesaikan, atau tidak ada perubahan yang dibuat sama sekali. Dengan operasi `BatchWriteItem`, mungkin bahwa hanya beberapa tindakan dalam batch yang berhasil sementara yang lain tidak.
- Transaksi tidak dapat dilakukan dengan menggunakan indeks.

Anda tidak dapat menargetkan item yang sama dengan beberapa operasi dalam transaksi yang sama. Misalnya, Anda tidak dapat melakukan `ConditionCheck` dan juga tindakan `Update` pada item yang sama dalam transaksi yang sama.

Anda dapat menambahkan jenis tindakan berikut pada transaksi:

- `Put` — Memulai operasi `PutItem` untuk membuat item baru atau mengganti item lama dengan item baru, kondisional atau tanpa menentukan syarat apa pun.
- `Update` — Memulai operasi `UpdateItem` untuk mengedit atribut item yang ada atau menambahkan item baru ke tabel jika belum ada. Gunakan tindakan ini untuk menambah,

menghapus, atau memperbarui atribut pada item yang ada secara dengan syarat maupun tanpa syarat.

- `Delete` — Memulai operasi `DeleteItem` untuk menghapus satu item dalam tabel yang diidentifikasi oleh kunci primernya.
- `ConditionCheck` — Memeriksa bahwa item ada atau periksa kondisi atribut tertentu dari item.

Setelah transaksi selesai, perubahan yang dibuat dalam transaksi yang dipropagasikan ke indeks sekunder global (GSI), stream, dan cadangan. Karena propagasi tidak langsung atau instan, jika tabel dipulihkan dari backup ([RestoreTableFromBackup](#)) atau diekspor ke titik waktu ([ExportTableToPointInTime](#)) pertengahan propagasi, mungkin berisi beberapa tetapi tidak semua perubahan yang dibuat selama transaksi baru-baru ini.

Idempotensi

Anda dapat menyertakan token klien ketika Anda membuat panggilan `TransactWriteItems` untuk memastikan bahwa permintaan tersebut idempoten. Menjadikan transaksi Anda idempoten membantu mencegah kesalahan aplikasi jika operasi yang sama diajukan beberapa kali karena waktu habis koneksi atau masalah konektivitas lainnya.

Jika panggilan `TransactWriteItems` asli berhasil, panggilan `TransactWriteItems` berikutnya dengan token klien yang sama berhasil kembali tanpa membuat perubahan apa pun. Jika parameter `ReturnConsumedCapacity` diatur, panggilan `TransactWriteItems` awal mengembalikan jumlah unit kapasitas tulis yang dikonsumsi dalam membuat perubahan. Panggilan `TransactWriteItems` selanjutnya dengan token klien yang sama mengembalikan jumlah unit kapasitas baca yang dikonsumsi dalam membaca item.

Poin penting tentang idempotensi

- Token klien berlaku selama 10 menit setelah permintaan yang menggunakannya selesai. Setelah 10 menit, setiap permintaan yang menggunakan token klien yang sama diperlakukan sebagai permintaan baru. Anda tidak boleh menggunakan kembali token klien yang sama untuk permintaan yang sama setelah 10 menit.
- Jika Anda mengulangi permintaan dengan token klien yang sama dalam jendela idempotensi 10 menit tetapi mengubah beberapa parameter permintaan lainnya, DynamoDB mengembalikan pengecualian `IdempotentParameterMismatch`.

Penanganan kesalahan untuk penulisan

Transaksi tulis tidak berhasil dalam situasi berikut:

- Ketika syarat di salah satu ekspresisyarat tidak terpenuhi.
- Ketika kesalahan validasi transaksi terjadi karena lebih dari satu tindakan dalam operasi `TransactWriteItems` yang sama menargetkan item yang sama.
- Saat permintaan `TransactWriteItems` bertentangan dengan operasi `TransactWriteItems` yang sedang berlangsung pada satu atau lebih item dalam permintaan `TransactWriteItems`. Dalam kasus ini, permintaan gagal dengan `TransactionCanceledException`.
- Ketika ada kapasitas yang ditetapkan tidak mencukupi untuk penyelesaian transaksi.
- Ketika ukuran item menjadi terlalu besar (lebih besar dari 400 KB), atau indeks sekunder lokal (LSI) menjadi terlalu besar, atau kesalahan validasi serupa terjadi karena perubahan yang dilakukan oleh transaksi.
- Ketika ada kesalahan pengguna, seperti format data yang tidak valid.

Untuk informasi selengkapnya tentang bagaimana pertentangan dengan operasi `TransactWriteItems` ditangani, lihat [Penanganan konflik transaksi di DynamoDB](#).

TransactGetItems API

`TransactGetItems` adalah operasi baca sinkron yang mengelompokkan hingga 100 tindakan `Get` menjadi satu. Tindakan ini dapat menargetkan hingga 100 item yang berbeda dalam satu atau lebih tabel DynamoDB dalam akun AWS dan Wilayah yang sama. Ukuran agregat item dalam transaksi tidak dapat melebihi 4 MB.

Tindakan `Get` dilakukan secara atom sehingga semuanya berhasil atau semuanya gagal:

- `Get` — Memulai operasi `GetItem` untuk mengambil satu set atribut untuk item dengan kunci primer yang diberikan. Jika tidak ditemukan item yang cocok, `Get` tidak mengembalikan data apa pun.

Penanganan kesalahan untuk pembacaan

Transaksi baca tidak berhasil dalam situasi berikut:

- Saat permintaan `TransactGetItems` bertentangan dengan operasi `TransactWriteItems` yang sedang berlangsung pada satu atau lebih item dalam permintaan `TransactGetItems`. Dalam kasus ini, permintaan gagal dengan `TransactionCanceledException`.
- Ketika ada kapasitas yang ditetapkan tidak mencukupi untuk penyelesaian transaksi.
- Ketika ada kesalahan pengguna, seperti format data yang tidak valid.

Untuk informasi selengkapnya tentang bagaimana pertentangan dengan operasi `TransactGetItems` ditangani, lihat [Penanganan konflik transaksi di DynamoDB](#).

Tingkat isolasi untuk DynamoDB transactions

Tingkat isolasi operasi transaksional (`TransactWriteItems` atau `TransactGetItems`) dan operasi lainnya adalah sebagai berikut.

DAPAT DISERIALKAN

Isolasi yang dapat diserialkan memastikan bahwa hasil dari beberapa operasi bersamaan sama seperti jika tidak ada operasi dimulai hingga operasi sebelumnya telah selesai.

Ada isolasi serialisasi antara jenis operasi berikut:

- Antara operasi transaksional dan operasi tulis standar (`PutItem`, `UpdateItem`, atau `DeleteItem`).
- Antara operasi transaksional dan operasi baca standar (`GetItem`).
- Antara operasi `TransactWriteItems` dan operasi `TransactGetItems`.

Meskipun ada isolasi serialisasi antara operasi transaksional, dan setiap standar individu menulis dalam suatu `BatchWriteItem` operasi, tidak ada isolasi serial antara transaksi dan operasi sebagai satu unit. `BatchWriteItem`

Demikian pula, tingkat isolasi antara operasi transaksional dan individu `GetItems` dalam operasi `BatchGetItem` dapat diserialisasikan. Tetapi tahap isolasi antara transaksi dan operasi `BatchGetItem` sebagai unit adalah read-committed.

Permintaan `GetItem` tunggal dapat diserialisasikan sehubungan dengan permintaan `TransactWriteItems` dalam salah satu dari dua cara, baik sebelum atau setelah permintaan `TransactWriteItems`. Beberapa permintaan `GetItem`, terhadap kunci dalam permintaan

`TransactWriteItems` bersamaan dapat dijalankan dalam urutan apa pun, dan karena itu hasilnya `read-committed`.

Misalnya, jika permintaan `GetItem` untuk item A dan item B dijalankan bersamaan dengan permintaan `TransactWriteItems` yang memodifikasi item A dan item B, ada empat kemungkinan:

- Kedua permintaan `GetItem` dijalankan sebelum permintaan `TransactWriteItems`.
- Kedua permintaan `GetItem` dijalankan setelah permintaan `TransactWriteItems`.
- Permintaan `GetItem` untuk item A dijalankan sebelum permintaan `TransactWriteItems`. Untuk item B, `GetItem` dijalankan setelah `TransactWriteItems`.
- Permintaan `GetItem` untuk item B dijalankan sebelum permintaan `TransactWriteItems`. Untuk item A, `GetItem` dijalankan setelah `TransactWriteItems`.

Anda harus menggunakan `TransactGetItems` jika Anda lebih suka tingkat isolasi serial untuk beberapa permintaan. `GetItem`

Jika pembacaan non-transaksional dilakukan pada beberapa item yang merupakan bagian dari permintaan penulisan transaksi yang sama dalam penerbangan, Anda mungkin dapat membaca status baru dari beberapa item dan status lama item lainnya. Anda akan dapat membaca status baru dari semua item yang merupakan bagian dari permintaan penulisan transaksi hanya ketika respons berhasil diterima untuk penulisan transaksional.

READ-COMMITTED

Isolasi `read-commit` memastikan bahwa operasi baca selalu mengembalikan komitmen untuk suatu item - pembacaan tidak akan pernah menampilkan tampilan ke item yang mewakili keadaan dari penulisan transaksional yang pada akhirnya tidak berhasil. Isolasi komitmen baca tidak mencegah modifikasi item segera setelah operasi baca.

Tingkat isolasi `read-committed` antara operasi transaksional dan operasi baca yang melibatkan beberapa baca standar (`BatchGetItem`, `Query`, atau `Scan`). Jika transaksional tulis memperbarui item di tengah operasi `BatchGetItem`, `Query`, atau `Scan`, bagian selanjutnya dari operasi baca mengembalikan nilai berkomitmen baru (dengan `ConsistentRead`) atau mungkin nilai berkomitmen sebelumnya (bacaan akhir konsisten).

Ringkasan operasi

Untuk meringkas, tabel berikut menunjukkan tingkat isolasi antara operasi transaksi (`TransactWriteItems` atau `TransactGetItems`) dan operasi lainnya.

Operasi	Tingkat Isolasi
DeleteItem	Dapat diserialkan
PutItem	Dapat diserialkan
UpdateItem	Dapat diserialkan
GetItem	Dapat diserialkan
BatchGetItem	Read-committed*
BatchWriteItem	Tidak Dapat Diserialkan*
Query	Read-committed
Scan	Read-committed
Operasi transaksional lainnya	Dapat diserialkan

Tingkat yang ditandai dengan tanda bintang (*) berlaku untuk operasi sebagai sebuah unit. Namun, tindakan individu dalam operasi tersebut memiliki tingkat isolasi yang dapat diserialkan.

Penanganan konflik transaksi di DynamoDB

Konflik transaksional dapat terjadi selama permintaan tingkat item bersamaan pada item dalam transaksi. Konflik transaksi dapat terjadi dalam skenario berikut:

- Permintaan PutItem, UpdateItem, atau DeleteItem untuk item yang bertentangan dengan permintaan TransactWriteItems yang sedang berlangsung yang mencakup item yang sama.
- Item dalam permintaan TransactWriteItems adalah bagian dari permintaan TransactWriteItems lain yang sedang berlangsung.
- Item dalam permintaan TransactGetItems adalah bagian dari permintaan TransactWriteItems, BatchWriteItem, PutItem, UpdateItem, atau DeleteItem yang sedang berlangsung.

Note

- Saat permintaan `PutItem`, `UpdateItem`, atau `DeleteItem` ditolak, permintaan gagal dengan `TransactionConflictException`.
- Jika ada permintaan tingkat item dalam `TransactWriteItems` atau `TransactGetItems` ditolak, permintaan gagal dengan `TransactionCanceledException`. Jika permintaan tersebut gagal, SDK AWS tidak mencoba lagi permintaan.

Jika Anda menggunakan AWS SDK for Java, pengecualian berisi daftar [CancellationReasons](#), diurutkan sesuai dengan daftar item dalam parameter `TransactItems` permintaan. Untuk bahasa lain, representasi string daftar disertakan dalam pesan kesalahan pengecualian.

- Jika operasi `TransactWriteItems` atau `TransactGetItems` yang sedang berlangsung bertentangan dengan permintaan `GetItem` bersamaan, kedua operasi dapat berhasil.

[TransactionConflict CloudWatch Metrik](#) ditambahkan untuk setiap permintaan tingkat item yang gagal.

Menggunakan API Transaksional di DynamoDB Accelerator (DAX)

`TransactWriteItems` dan `TransactGetItems` keduanya didukung di DynamoDB Accelerator (DAX) dengan tingkat isolasi yang sama seperti di DynamoDB.

`TransactWriteItems` menulis melalui DAX. DAX menyampaikan panggilan `TransactWriteItems` ke DynamoDB dan mengembalikan respons. Untuk mempopulasi cache setelah menulis, DAX memanggil `TransactGetItems` di latar belakang untuk setiap item dalam operasi `TransactWriteItems`, yang mengonsumsi unit kapasitas baca tambahan. (Untuk informasi selengkapnya, lihat [Manajemen kapasitas untuk transaksi](#).) Fungsionalitas ini memungkinkan Anda untuk menjaga logika aplikasi Anda sederhana dan menggunakan DAX untuk kedua operasi transaksional dan nontransaksional.

Panggilan `TransactGetItems` disampaikan melalui DAX tanpa item di-cache secara lokal. Ini adalah perilaku yang sama seperti untuk API bacaan sangat konsisten di DAX.

Manajemen kapasitas untuk transaksi

Tidak ada biaya tambahan untuk mengaktifkan transaksi untuk tabel DynamoDB Anda. Anda hanya membayar untuk baca atau tulis yang merupakan bagian dari transaksi Anda. DynamoDB melakukan dua baca atau tulis mendasar dari setiap item dalam transaksi: satu untuk mempersiapkan transaksi dan satu untuk melakukan transaksi. Dua operasi baca/tulis yang mendasarinya terlihat di metrik Amazon CloudWatch Anda.

Rencana untuk baca dan tulis tambahan yang diperlukan oleh API transaksional ketika Anda menyediakan kapasitas untuk tabel Anda. Misalnya, anggaplah aplikasi Anda menjalankan satu transaksi per detik, dan setiap transaksi menulis tiga item 500-byte dalam tabel Anda. Setiap item membutuhkan dua unit kapasitas tulis (WCU): satu untuk mempersiapkan transaksi dan satu untuk melakukan transaksi. Oleh karena itu, Anda akan perlu untuk menyediakan enam WCU pada tabel.

Jika Anda menggunakan DynamoDB Accelerator (DAX) dalam contoh sebelumnya, Anda juga akan menggunakan dua unit kapasitas baca (RCU) untuk setiap item di panggilan `TransactWriteItems`. Jadi Anda akan perlu untuk menyediakan enam RCU tambahan pada tabel.

Demikian pula, jika aplikasi Anda menjalankan satu transaksi baca per detik, dan setiap transaksi membaca tiga item berukuran 500 byte di tabel Anda, Anda perlu menyediakan enam unit kapasitas baca (RCU) ke tabel tersebut. Membaca setiap item membutuhkan dua RCU: satu untuk mempersiapkan transaksi dan satu untuk melakukan transaksi.

Selain itu, perilaku SDK default adalah untuk mencoba lagi transaksi dalam kasus pengecualian `TransactionInProgressException`. Rencanakan tambahan unit kapasitas baca (RCU) yang digunakan oleh percobaan ulang ini. Hal yang sama juga berlaku jika Anda mencoba ulang transaksi dalam kode Anda sendiri menggunakan `ClientRequestToken`.

Praktik terbaik untuk transactions

Pertimbangkan praktik yang disarankan berikut saat menggunakan DynamoDB Transactions.

- Aktifkan penskalaan otomatis pada tabel Anda, atau pastikan bahwa Anda telah menyediakan kapasitas throughput yang cukup untuk melakukan dua operasi baca atau tulis untuk setiap item dalam transaksi Anda.
- Jika Anda tidak menggunakan SDK yang disediakan AWS, sertakan atribut `ClientRequestToken` ketika Anda membuat panggilan `TransactWriteItems` untuk memastikan bahwa permintaan tersebut idempoten.

- Jangan mengelompokkan operasi bersama-sama dalam sebuah transaksi jika tidak diperlukan. Misalnya, jika satu transaksi dengan 10 operasi dapat dipecah menjadi beberapa transaksi tanpa mengurangi kebenaran aplikasi, sebaiknya pisahkan transaksi tersebut. Transaksi yang lebih sederhana meningkatkan throughput dan lebih mungkin untuk berhasil.
- Beberapa transaksi memperbarui item yang sama secara bersamaan dapat menyebabkan konflik yang membatalkan transaksi. Kami merekomendasikan praktik terbaik DynamoDB berikut untuk pemodelan data untuk meminimalkan konflik tersebut.
- Jika satu set atribut sering diperbarui di beberapa item sebagai bagian dari transaksi tunggal, pertimbangkan pengelompokan atribut menjadi satu item untuk mengurangi lingkup transaksi.
- Hindari penggunaan transaksi untuk menyerap data secara massal. Untuk tulis massal, lebih baik menggunakan `BatchWriteItem`.

Menggunakan API transaksional dengan tabel global

Operasi yang terkandung dalam transaksi DynamoDB hanya dijamin transaksional di wilayah tempat transaksi awalnya dijalankan. Transaksionalitas tidak dipertahankan ketika perubahan yang diterapkan dalam transaksi direplikasi di seluruh Wilayah ke replika tabel global.

Transaksi DynamoDB AWS Labs vs. perpustakaan klien transaksi

Transaksi DynamoDB menyediakan penggantian yang lebih hemat biaya, kuat, dan berkinerja untuk perpustakaan klien transaksi. [AWS Labs](#) Kami menyarankan Anda memperbarui aplikasi Anda untuk menggunakan API transaksi asli sisi server.

Menggunakan IAM dengan DynamoDB transactions

Anda dapat menggunakan AWS Identity and Access Management (IAM) untuk membatasi tindakan yang dapat dilakukan operasi transaksional di Amazon DynamoDB. Untuk informasi selengkapnya tentang menggunakan kebijakan IAM dalam DynamoDB, lihat [Kebijakan berbasis identitas untuk DynamoDB](#).

Izin untuk tindakan `Put`, `Update`, `Delete`, dan `Get` diatur oleh izin yang digunakan untuk operasi `PutItem`, `UpdateItem`, `DeleteItem`, dan `GetItem` yang mendasarinya. Untuk tindakan `ConditionCheck`, Anda dapat menggunakan izin `dynamodb:ConditionCheck` dalam kebijakan IAM.

Berikut ini adalah contoh kebijakan IAM yang dapat Anda gunakan untuk mengonfigurasi DynamoDB transactions.

Contoh 1: Mengizinkan operasi transaksional

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:ConditionCheckItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb>DeleteItem",
        "dynamodb:GetItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:*:*:table/table04"
      ]
    }
  ]
}
```

Contoh 2: Hanya mengizinkan operasi transaksional

```
{
  "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"
          ]
        }
      }
    }
  ]
}
```

```

    ]
  }
}
]
}

```

Contoh 3: Mengizinkan baca dan tulis nontransaksional, dan memblokir baca dan tulis transaksional

```

{
  "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"
    ]
}

```

Contoh 4: Mencegah informasi dikembalikan pada ConditionCheck kegagalan

```

{
  "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"
        }
      }
    }
  ]
}

```

Contoh DynamoDB Transactions

Sebagai contoh situasi di mana Amazon DynamoDB transactions dapat berguna, pertimbangkan contoh aplikasi Java untuk marketplace online.

Aplikasi ini memiliki tiga tabel DynamoDB di backend:

- **Customers** — Tabel ini menyimpan detail tentang pelanggan marketplace. Kunci utamanya adalah `CustomerId` pengidentifikasi unik.
- **ProductCatalog** — Tabel ini menyimpan detail seperti harga dan ketersediaan tentang produk yang dijual di marketplace. Kunci utamanya adalah `ProductId` pengidentifikasi unik.

- **Orders** — Tabel ini menyimpan detail tentang pesanan dari marketplace. Kunci utamanya adalah `OrderId` pengidentifikasi unik.

Membuat Pesanan

Potongan kode berikut menggambarkan cara menggunakan DynamoDB transaction untuk mengkoordinasikan beberapa langkah yang diperlukan untuk membuat dan memproses pesanan. Menggunakan tunggal all-or-nothing operasi memastikan bahwa jika ada bagian dari transaksi gagal, tidak ada tindakan dalam transaksi dijalankan dan tidak ada perubahan yang dibuat.

Dalam contoh ini, Anda mengatur pesanan dari pelanggan yang `customerId` adalah `09e8e9c8-ec48`. Anda kemudian menjalankannya sebagai transaksi tunggal menggunakan alur kerja pemrosesan urutan sederhana berikut:

1. Menentukan bahwa ID pelanggan valid.
2. Pastikan bahwa produk tersebut `IN_STOCK`, dan perbarui status produk ke `SOLD`.
3. Pastikan pesanan belum ada, dan buat pesanan.

Memvalidasi Pelanggan

Pertama, tentukan tindakan untuk memverifikasi bahwa pelanggan dengan `customerId` setara dengan `09e8e9c8-ec48` ada dalam tabel pelanggan.

```
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 + ")");
```

Memperbarui status produk

Berikutnya, tentukan tindakan untuk memperbarui status produk untuk `SOLD` jika syarat status produk saat ini diatur ke `IN_STOCK` adalah `true`. Mengatur parameter

`ReturnValuesOnConditionCheckFailure` mengembalikan item jika atribut status produk item tidak setara dengan `IN_STOCK`.

```
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);
```

Membuat Pesanan

Terakhir, buat pesanan selama pesanan dengan `OrderId` belum ada.

```
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 + ")");
```

Jalankan Transaksi

Contoh berikut menggambarkan cara menjalankan tindakan yang didefinisikan sebelumnya sebagai tunggal all-or-nothing operasi.

```
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());
}
```

Membaca Detail Pesanan

Contoh berikut menunjukkan cara membaca pesanan yang selesai secara transaksional di tabel Orders dan ProductCatalog.

```
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());
}
```

Contoh tambahan

- [Menggunakan transaksi dari DynamoDBMapper](#)

Ubah tangkapan data dengan Amazon DynamoDB

Banyak aplikasi mendapatkan manfaat dari menangkap perubahan ke item yang disimpan dalam tabel DynamoDB pada titik waktu ketika perubahan terjadi. Berikut adalah beberapa contoh kasus penggunaan:

- Sebuah aplikasi seluler populer memodifikasi data dalam tabel DynamoDB, pada tingkat ribuan pembaruan per detik. Aplikasi lain menangkap dan menyimpan data tentang pembaruan ini, menyediakan metrik near-real-time penggunaan untuk aplikasi seluler.
- Sebuah aplikasi keuangan memodifikasi data pasar saham dalam tabel DynamoDB. Berbagai aplikasi yang berjalan secara paralel melacak perubahan ini secara real time, menghitung value-at-risk, dan secara otomatis menyeimbangkan portofolio berdasarkan pergerakan harga saham.

- Sensor dalam kendaraan transportasi dan peralatan industri mengirim data ke meja DynamoDB. Aplikasi yang berbeda memantau kinerja dan mengirim pesan peringatan ketika masalah terdeteksi, memprediksi potensi cacat dengan menerapkan algoritma machine learning, dan mengompres serta mengarsip data ke Amazon Simple Storage Service (Amazon S3).
- Sebuah aplikasi secara otomatis mengirimkan pemberitahuan ke perangkat seluler dari semua teman dalam grup segera setelah satu teman mengunggah gambar baru.
- Seorang pelanggan baru menambahkan data ke tabel DynamoDB. Acara ini memanggil aplikasi lain yang mengirimkan email selamat datang ke pelanggan baru.

DynamoDB mendukung streaming catatan tangkapan data perubahan tingkat item yang dalam hampir waktu nyata. Anda dapat membangun aplikasi yang mengkonsumsi stream ini dan mengambil tindakan berdasarkan isinya.

Video berikut akan memberi Anda tampilan pengantar tentang konsep pengambilan data perubahan.

[Mode kapasitas tabel](#)

Topik

- [Opsi streaming untuk tangkapan data perubahan](#)
- [Menggunakan Kinesis Data Streams untuk menangkap perubahan pada DynamoDB](#)
- [Ubah tangkapan data DynamoDB Streams](#)

Opsi streaming untuk tangkapan data perubahan

DynamoDB menawarkan dua model streaming untuk penangkapan data perubahan: Kinesis Data Streams untuk DynamoDB dan DynamoDB Streams.

Untuk membantu Anda memilih solusi yang tepat untuk aplikasi Anda, tabel berikut merangkum fitur dari setiap model streaming.

Properti	Aliran Data Kinesis untuk DynamoDB	DynamoDB Streams
Retensi data	Hingga 1 tahun .	24 jam.
Dukungan Perpustakaan Klien Kinesis (KCL)	Mendukung KCL versi 1.X dan 2.X .	Mendukung KCL versi 1.X .

Properti	Aliran Data Kinesis untuk DynamoDB	DynamoDB Streams
Jumlah konsumen	Hingga 5 konsumen simultan per serpihan, atau hingga 20 konsumen simultan per serpihan dengan fan-out yang ditingkatkan .	Hingga 2 konsumen simultan per serpihan.
Kuota throughput	Tidak terbatas.	Tunduk pada kuota throughput oleh tabel DynamoDB dan Wilayah. AWS
Model pengiriman catatan	Tarik model melalui HTTP menggunakan GetRecord dan dengan fan-out yang disempurnakan , Kinesis Data Streams mendorong catatan melalui HTTP/2 dengan menggunakan Subscribe ToShard	Tarik model melalui HTTP menggunakan GetRecords .
Pengurutan catatan	Atribut stempel waktu pada setiap catatan stream dapat digunakan untuk mengidentifikasi urutan aktual di mana perubahan terjadi dalam tabel DynamoDB.	Untuk setiap item yang diubah dalam tabel DynamoDB, catatan stream muncul dalam urutan yang sama seperti modifikasi aktual untuk item.
Catatan duplikat	Catatan duplikat terkadang muncul di stream.	Tidak ada catatan duplikat muncul di stream.
Opsi pemrosesan stream	Memproses rekaman aliran menggunakan AWS Lambda , Layanan Terkelola Amazon untuk Apache Flink , Kinesis data firehose , atau streaming ETL AWS Glue	Proses catatan stream menggunakan AWS Lambda atau Adaptor Kinesis DynamoDB Streams .

Properti	Aliran Data Kinesis untuk DynamoDB	DynamoDB Streams
Tingkat daya tahan	Zone ketersediaan untuk menyediakan failover otomatis tanpa gangguan.	Zone ketersediaan untuk menyediakan failover otomatis tanpa gangguan.

Anda dapat mengaktifkan kedua model streaming pada tabel DynamoDB yang sama.

Video berikut berbicara lebih banyak tentang perbedaan antara dua opsi tersebut.

[DynamoDB Streams vs Kinesis Data Streams](#)

Menggunakan Kinesis Data Streams untuk menangkap perubahan pada DynamoDB

Gunakan Amazon Kinesis Data Streams untuk menangkap perubahan pada Amazon DynamoDB

Kinesis Data Streams menangkap modifikasi tingkat item di tabel DynamoDB apa pun dan mereplikasikannya ke [Kinesis data stream](#). Aplikasi Anda dapat mengakses stream ini dan melihat perubahan tingkat item dalam hampir waktu nyata. Anda dapat terus menangkap dan menyimpan data berukuran terabyte per jam. Anda dapat memanfaatkan waktu penyimpanan data yang lebih lama—dan dengan kemampuan fan-out yang ditingkatkan, Anda dapat menjangkau dua atau lebih aplikasi hilir secara bersamaan. Manfaat lainnya termasuk audit tambahan dan transparansi keamanan.

Kinesis Data Streams juga memberi Anda akses [ke Amazon Data Firehose dan Amazon Managed Service](#) untuk Apache Flink. Layanan ini dapat membantu Anda membangun aplikasi yang mendukung dasbor real-time, menghasilkan peringatan, menerapkan penetapan harga dan periklanan dinamis, serta menerapkan analisis data dan algoritme pembelajaran mesin yang canggih.

Note

Penggunaan Kinesis Data Streams untuk DynamoDB tunduk pada [harga Kinesis Data Streams](#) untuk aliran data dan [harga DynamoDB](#) untuk tabel sumber.

Bagaimana Kinesis Data Streams bekerja dengan DynamoDB

Ketika aliran data Kinesis diaktifkan untuk tabel DynamoDB, tabel mengirimkan rekaman data yang mencatat perubahan apa pun pada data tabel tersebut. Catatan data ini meliputi:

- Waktu spesifik item mana pun baru saja dibuat, diperbarui, atau dihapus
- Kunci utama item itu
- Cuplikan catatan sebelum modifikasi
- Cuplikan catatan setelah modifikasi

Catatan data ini diambil dan dipublikasikan hampir secara waktu nyata. Setelah ditulis ke aliran data Kinesis, data tersebut dapat dibaca seperti catatan lainnya. Anda dapat menggunakan Kinesis Client Library, menggunakan AWS Lambda, memanggil Kinesis Data Streams API, dan menggunakan layanan terhubung lainnya. Untuk informasi selengkapnya, lihat [Membaca Data dari Amazon Kinesis Data Streams](#) dalam Panduan Developer Amazon Kinesis Data Streams.

Perubahan pada data ini juga ditangkap secara asinkron. Kinesis tidak memiliki dampak kinerja pada tabel asal streamingnya. Catatan aliran yang disimpan dalam aliran data Kinesis Anda juga dienkripsi saat tidak digunakan. Untuk informasi selengkapnya, lihat [Perlindungan Data di Amazon Kinesis Data Streams](#).

Catatan aliran data Kinesis mungkin muncul dalam urutan yang berbeda dari saat perubahan item terjadi. Notifikasi item yang sama mungkin juga muncul lebih dari satu kali di aliran. Anda dapat memeriksa `ApproximateCreationDateTime` atribut untuk mengidentifikasi urutan modifikasi item terjadi, dan untuk mengidentifikasi catatan duplikat.

Saat Anda mengaktifkan aliran data Kinesis sebagai tujuan streaming tabel DynamoDB, Anda dapat mengonfigurasi presisi `ApproximateCreationDateTime` nilai dalam milidetik atau mikrodetik. Secara default, `ApproximateCreationDateTime` menunjukkan waktu perubahan dalam milidetik. Selain itu, Anda dapat mengubah nilai ini di tujuan streaming aktif. Setelah pembaruan seperti itu, catatan aliran yang ditulis ke Kinesis akan memiliki `ApproximateCreationDateTime` nilai presisi yang diinginkan.

Nilai biner yang ditulis ke DynamoDB harus dikodekan dalam [format berenkode base64](#). Namun, ketika catatan data ditulis ke aliran data Kinesis, nilai biner yang dikodekan ini dikodekan dengan pengkodean base64 untuk kedua kalinya. Saat membaca catatan ini dari aliran data Kinesis, untuk mengambil nilai biner mentah, aplikasi harus mendekode nilai ini dua kali.

DynamoDB mengenakan biaya untuk penggunaan Kinesis Data Streams dalam unit pengambilan data perubahan. Perubahan 1 KB per item dihitung sebagai satu unit pengambilan data perubahan. KB perubahan di setiap item dihitung berdasarkan gambar "sebelum" dan "sesudah" yang lebih besar dari item yang ditulis ke aliran, menggunakan logika yang sama dengan [konsumsi unit kapasitas untuk operasi tulis](#). Mirip dengan bagaimana mode DynamoDB [sesuai permintaan](#) bekerja, Anda tidak perlu menyediakan throughput kapasitas untuk unit penangkapan data perubahan.

Mengaktifkan aliran data Kinesis untuk tabel DynamoDB Anda

Anda dapat mengaktifkan atau menonaktifkan streaming ke Kinesis dari tabel DynamoDB yang ada dengan menggunakan, AWS SDK AWS Management Console, atau (). AWS Command Line Interface AWS CLI

- Anda hanya dapat mengalirkan data dari DynamoDB ke Kinesis Data Streams di AWS akun dan Wilayah yang sama dengan tabel Anda. AWS
- Anda hanya dapat melakukan streaming data dari tabel DynamoDB ke satu Kinesis data stream.

Membuat perubahan pada tujuan Kinesis Data Streams pada tabel DynamoDB

Secara default, semua catatan aliran data Kinesis menyertakan atribut `ApproximateCreationDateTime`. Atribut ini mewakili stempel waktu dalam milidetik dari perkiraan waktu ketika setiap rekaman dibuat. Anda dapat mengubah ketepatan nilai-nilai ini dengan menggunakan <https://console.aws.amazon.com/kinesis>, SDK atau AWS CLI

Memulai Kinesis Data Streams untuk Amazon DynamoDB

Bagian ini menjelaskan cara menggunakan Kinesis Data Streams untuk tabel Amazon DynamoDB dengan konsol Amazon AWS Command Line Interface AWS CLI DynamoDB, (), dan API.

Semua contoh ini menggunakan tabel `Music` DynamoDB yang dibuat sebagai bagian dari tutorial [Memulai dengan DynamoDB](#).

Untuk mempelajari lebih lanjut tentang cara membangun konsumen dan menghubungkan aliran data Kinesis Anda ke AWS layanan lain, lihat [Membaca data dari Kinesis Data Streams di panduan pengembang Amazon Kinesis Data Streams](#).

Note

Saat Anda pertama kali menggunakan serpihan KDS, sebaiknya atur serpihan Anda untuk ditingkatkan dan diturunkan skalanya sesuai dengan pola penggunaan. Setelah mengumpulkan lebih banyak data tentang pola penggunaan, Anda dapat menyesuaikan serpihan di aliran agar sesuai.

Console

1. [Masuk ke AWS Management Console dan buka konsol Kinesis di https://console.aws.amazon.com/kinesis/](https://console.aws.amazon.com/kinesis/).
2. Pilih Buat aliran data dan ikuti petunjuk untuk membuat stream yang disebut `samplestream`.
3. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
4. Di panel navigasi di sisi kiri konsol, pilih Tabel.
5. Pilih tabel Musik.
6. Pilih tab Ekspor dan aliran.

Music Actions Explore table items

Overview | Indexes | Monitor | Global tables | Backups | **Exports and streams** | Additional settings

Exports to S3 (0) Info View details Export to S3

Showing all export jobs from the last 90 days.

< 1 > ⚙️

Export ARN	Destination S3 bucket	Status	Export job start time (UTC+00:00)	Export type
No exports				

Export to S3

Amazon Kinesis data stream details Edit record timestamp precision Turn off

Amazon Kinesis Data Streams for DynamoDB captures item-level changes in your table, and replicates the changes to a Kinesis data stream. You then can consume and manage the change information from Kinesis. [Learn more](#)

Status	Destination stream
🟢 On	test
Record timestamp precision	
Microsecond	

- (Opsional) Di bawah detail aliran data Amazon Kinesis, Anda dapat mengubah presisi stempel waktu rekaman dari mikrodetik (default) menjadi milidetik.
- Pilih samplestream dari daftar menurun.
- Pilih tombol Hidupkan.

AWS CLI

- Buat Kinesis Data Streams bernama `samplestream` dengan menggunakan [perintah create-stream](#).

```
aws kinesis create-stream --stream-name samplestream --shard-count 3
```

Lihat [Pertimbangan manajemen pecahan untuk Kinesis Data Streams](#) sebelum menetapkan jumlah serpihan untuk Kinesis data stream.

- Periksa apakah stream Kinesis aktif dan siap digunakan dengan menggunakan [perintah describe-stream](#).


```
aws kinesis describe-stream --stream-name samplestream
```

3. Aktifkan Kinesis streaming pada tabel DynamoDB dengan menggunakan perintah `enable-kinesis-streaming-destination` DynamoDB. Ganti `stream-arn` nilai dengan yang ditampilkan `describe-stream` pada langkah sebelumnya. Secara opsional, aktifkan streaming dengan presisi nilai stempel waktu yang lebih granular (mikrodetik) yang dikembalikan pada setiap catatan.

Aktifkan streaming dengan presisi stempel waktu mikrodetik:

```
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
```

Atau aktifkan streaming dengan presisi stempel waktu default (milidetik):

```
aws dynamodb enable-kinesis-streaming-destination \  
  --table-name Music \  
  --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
```

4. Periksa apakah streaming Kinesis aktif pada tabel DynamoDB dengan menggunakan perintah `describe-kinesis-streaming-destination` DynamoDB.

```
aws dynamodb describe-kinesis-streaming-destination --table-name Music
```

5. Menulis data ke Daftar Tabel DynamoDB dengan menggunakan perintah `put-item`, seperti yang dijelaskan dalam [Panduan Developer DynamoDB](#).

```
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": "Call Me Today"}, "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'
```

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

- Gunakan perintah CLI Kinesis [get-records](#) untuk mengambil konten stream Kinesis. Kemudian gunakan potongan kode berikut untuk melakukan deserialisasi konten stream.

```
/**  
 * 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. Ikuti petunjuk dalam panduan developer Kinesis Data Streams untuk [menciptakan](#) Kinesis data stream bernama `samplestream` menggunakan Java.

Lihat [Pertimbangan manajemen pecahan untuk Kinesis Data Streams](#) sebelum menetapkan jumlah serpihan untuk Kinesis data stream.

2. Gunakan cuplikan kode berikut untuk mengaktifkan Kinesis streaming pada tabel DynamoDB. Secara opsional, aktifkan streaming dengan presisi nilai stempel waktu yang lebih granular (mikrodetik) yang dikembalikan pada setiap catatan.

Aktifkan streaming dengan presisi stempel waktu mikrodetik:

```
EnableKinesisStreamingConfiguration enableKdsConfig =
    EnableKinesisStreamingConfiguration.builder()

        .approximateCreationDateTimePrecision(ApproximateCreationDateTimePrecision.MICROSECOND)
        .build();

EnableKinesisStreamingDestinationRequest enableKdsRequest =
    EnableKinesisStreamingDestinationRequest.builder()
        .tableName(tableName)
        .streamArn(kdsArn)
        .enableKinesisStreamingConfiguration(enableKdsConfig)
        .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
    ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

Atau aktifkan streaming dengan presisi stempel waktu default (milidetik):

```
EnableKinesisStreamingDestinationRequest enableKdsRequest =
    EnableKinesisStreamingDestinationRequest.builder()
        .tableName(tableName)
        .streamArn(kdsArn)
        .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
    ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

- Ikuti instruksi dalam panduan pengembang Kinesis Data Streams untuk [membaca](#) dari aliran data yang dibuat.
- Gunakan potongan kode berikut untuk melakukan deserialisasi konten stream

```
/**
 * 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());
}
```

Membuat perubahan pada aliran data Amazon Kinesis yang aktif

Bagian ini menjelaskan cara membuat perubahan pada Kinesis Data Streams aktif untuk penyiapan DynamoDB dengan menggunakan konsol, dan API. AWS CLI

AWS Management Console

1. [Buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/](https://console.aws.amazon.com/dynamodb/)
2. Pergi ke meja Anda.
3. Pilih Ekspor dan Aliran.

AWS CLI

1. Hubungi `describe-kinesis-streaming-destination` untuk mengonfirmasi bahwa streaming tersebut `ACTIVE`.
2. Panggil `UpdateKinesisStreamingDestination`, seperti dalam contoh ini:

```
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. Hubungi `describe-kinesis-streaming-destination` untuk mengonfirmasi bahwa streaming tersebut `UPDATING`.
4. Panggil `describe-kinesis-streaming-destination` secara berkala hingga status streaming `ACTIVE` kembali. Biasanya diperlukan waktu hingga 5 menit agar pembaruan presisi stempel waktu diterapkan. Setelah status ini diperbarui, itu menunjukkan bahwa pembaruan selesai dan nilai presisi baru akan diterapkan pada catatan masa depan.
5. Tulis ke meja menggunakan `putItem`.
6. Gunakan `get-records` perintah Kinesis untuk mendapatkan konten aliran.
7. Konfirmasikan bahwa penulisan memiliki presisi yang diinginkan.
`ApproximateCreationDateTime`

API Java

1. Berikan cuplikan kode yang membuat `UpdateKinesisStreamingDestination` permintaan dan respons. `UpdateKinesisStreamingDestination`

2. Berikan cuplikan kode yang membuat `DescribeKinesisStreamingDestination` permintaan dan file. `DescribeKinesisStreamingDestination` response
3. Panggil `describe-kinesis-streaming-destination` secara berkala hingga status streaming ACTIVE kembali, menunjukkan bahwa pembaruan selesai dan nilai presisi baru akan diterapkan pada catatan masa depan.
4. Lakukan menulis ke meja.
5. Baca dari aliran dan deserialisasi konten streaming.
6. Konfirmasikan bahwa penulisan memiliki presisi yang diinginkan.
`ApproximateCreationDateTime`

Mengkonfigurasi pecahan dan memantau pengambilan data perubahan dengan Kinesis Data Streams di DynamoDB

Pertimbangan manajemen pecahan untuk Kinesis Data Streams

Kinesis data stream, menghitung throughputnya dalam [pecahan](#). Di Amazon Kinesis Data stream, Anda dapat memilih antara mode sesuai permintaan dan mode yang disediakan untuk aliran data Anda.

Sebaiknya gunakan mode sesuai permintaan untuk Kinesis Data Stream Anda jika beban kerja penulisan DynamoDB Anda sangat bervariasi dan tidak dapat diprediksi. Dengan mode on-demand, tidak ada perencanaan kapasitas yang diperlukan karena Kinesis Data Streams secara otomatis mengelola pecahan untuk menyediakan throughput yang diperlukan.

Untuk beban kerja yang dapat diprediksi, Anda dapat menggunakan mode yang disediakan untuk Kinesis Data Stream Anda. Dengan mode yang disediakan, Anda harus menentukan jumlah pecahan untuk aliran data untuk mengakomodasi catatan pengambilan data perubahan dari DynamoDB. Untuk menentukan jumlah pecahan yang dibutuhkan aliran data Kinesis untuk mendukung tabel DynamoDB Anda, Anda memerlukan nilai input berikut:

- Ukuran rata-rata catatan tabel DynamoDB Anda dalam byte (`average_record_size_in_bytes`).
- Jumlah maksimum operasi tulis yang tabel DynamoDB Anda akan melakukan per detik. Ini termasuk membuat, menghapus, dan memperbaiki operasi yang dilakukan oleh aplikasi Anda, serta operasi yang dihasilkan secara otomatis seperti Time to Live generated delete operations (`write_throughput`).

- Persentase operasi pembaruan dan penyimpanan yang Anda lakukan pada tabel Anda, dibandingkan dengan membuat atau menghapus operasi (`percentage_of_updates`). Perlu diingat bahwa operasi pembaruan dan penyimpanan mereplikasi gambar lama dan baru dari item yang dimodifikasi ke stream. Ini menghasilkan dua kali ukuran item DynamoDB.

Anda dapat menghitung jumlah pecahan (`number_of_shards`) yang dibutuhkan aliran data Kinesis Anda dengan menggunakan nilai input dalam rumus berikut:

```
number_of_shards = ceiling( max( ((write_throughput * (4+percentage_of_updates) *
average_record_size_in_bytes) / 1024 / 1024), (write_throughput/1000)), 1)
```

Misalnya, Anda mungkin memiliki throughput maksimum 1040 operasi tulis per detik (`write_throughput`) dengan ukuran catatan rata-rata 800 byte (`average_record_size_in_bytes`). Jika 25 persen dari operasi penulisan tersebut adalah operasi pembaruan (`percentage_of_updates`), maka Anda akan memerlukan dua pecahan (`number_of_shards`) untuk mengakomodasi throughput streaming DynamoDB Anda:

```
ceiling( max( ((1040 * (4+25/100) * 800)/ 1024 / 1024), (1040/1000)), 1).
```

Pertimbangkan hal berikut sebelum menggunakan rumus untuk menghitung jumlah pecahan yang diperlukan dengan mode yang disediakan untuk aliran data Kinesis:

- Rumus ini membantu memperkirakan jumlah pecahan yang akan diperlukan untuk mengakomodasi catatan data perubahan DynamoDB Anda. Ini tidak mewakili jumlah total pecahan yang dibutuhkan dalam aliran data Kinesis Anda, seperti jumlah pecahan yang diperlukan untuk mendukung konsumen aliran data Kinesis tambahan.
- Anda mungkin masih mengalami pengecualian throughput baca dan tulis dalam mode yang disediakan jika Anda tidak mengonfigurasi aliran data untuk menangani throughput puncak Anda. Dalam hal ini, Anda harus menskalakan aliran data Anda secara manual untuk mengakomodasi lalu lintas data Anda.
- Rumus ini mempertimbangkan bloat tambahan yang dihasilkan oleh DynamoDB sebelum mengalirkan catatan data log perubahan ke Kinesis Data Stream.

Untuk mempelajari lebih lanjut tentang mode kapasitas pada Kinesis Data Stream lihat [Memilih Mode Kapasitas Aliran Data](#). Untuk mempelajari selengkapnya tentang perbedaan harga antara mode kapasitas yang berbeda, lihat harga [Amazon Kinesis Data Streams](#).

Memantau pengambilan data perubahan dengan Kinesis Data Streams

DynamoDB menyediakan beberapa metrik CloudWatch Amazon untuk membantu Anda memantau replikasi pengambilan data perubahan ke Kinesis. Untuk daftar lengkap CloudWatch metrik, lihat [Dimensi dan Metrik DynamoDB](#).

Untuk menentukan apakah streaming Anda memiliki kapasitas yang memadai, sebaiknya pantau item berikut selama pengaktifan streaming dan produksi:

- **ThrottledPutRecordCount**: Jumlah catatan yang dibatasi oleh aliran data Kinesis Anda karena kapasitas aliran data Kinesis tidak mencukupi. Anda mungkin mengalami beberapa throttling selama puncak penggunaan yang luar biasa, namun **ThrottledPutRecordCount** harus tetap serendah mungkin. DynamoDB mencoba lagi mengirimkan catatan yang dibatasi ke aliran data Kinesis, namun hal ini mungkin mengakibatkan latensi replikasi yang lebih tinggi.

Jika Anda mengalami throttling yang berlebihan dan secara rutin, maka Anda mungkin perlu meningkatkan jumlah serpihan aliran Kinesis secara proporsional dengan throughput tulis yang diamati pada tabel Anda. Untuk mempelajari selengkapnya tentang menentukan ukuran Kinesis data stream, lihat [Menentukan Ukuran Awal Kinesis Data Stream](#).

- **AgeOfOldestUnreplicatedRecord**: Waktu yang berlalu sejak perubahan tingkat item terlama yang belum direplikasi ke aliran data Kinesis muncul di tabel DynamoDB. Dalam operasi normal, **AgeOfOldestUnreplicatedRecord** harus dalam urutan milidetik. Jumlah ini bertambah berdasarkan upaya replikasi yang gagal karena pilihan konfigurasi yang dikontrol pelanggan.

Jika **AgeOfOldestUnreplicatedRecord** metrik melebihi 168 jam, replikasi perubahan tingkat item dari tabel DynamoDB ke aliran data Kinesis akan dinonaktifkan secara otomatis.

Contoh konfigurasi yang dikontrol pelanggan yang menyebabkan upaya replikasi gagal adalah kapasitas aliran data Kinesis yang kurang tersedia sehingga menyebabkan throttling berlebihan, atau pembaruan manual pada kebijakan akses aliran data Kinesis Anda yang mencegah DynamoDB menambahkan data ke aliran data Anda. Untuk menjaga metrik ini serendah mungkin, Anda mungkin perlu memastikan penyediaan kapasitas aliran data Kinesis yang tepat, dan memastikan bahwa izin DynamoDB tidak berubah.

- **FailedToReplicateRecordCount**: Jumlah catatan yang gagal direplikasi DynamoDB ke Kinesis data stream Anda. Item tertentu yang lebih besar dari 34 KB mungkin bertambah besar ukurannya untuk mengubah catatan data yang lebih besar dari batas ukuran item 1 MB Kinesis Data Streams. Perluasan ukuran ini terjadi ketika item yang lebih besar dari 34 KB ini menyertakan sejumlah besar nilai atribut Boolean atau kosong. Nilai atribut Boolean dan kosong disimpan

sebagai 1 byte di DynamoDB, namun diperluas hingga 5 byte saat diserialkan menggunakan JSON standar untuk replikasi Kinesis Data Streams. DynamoDB tidak dapat mereplikasi catatan perubahan tersebut ke aliran data Kinesis Anda. DynamoDB akan melewatkan catatan data perubahan ini, dan secara otomatis akan tetap melakukan replikasi atas catatan-catatan berikutnya.

Anda dapat membuat CloudWatch alarm Amazon yang mengirim pesan Amazon Simple Notification Service (Amazon SNS) untuk pemberitahuan ketika salah satu metrik sebelumnya melebihi ambang batas tertentu. Untuk informasi selengkapnya, lihat [Membuat CloudWatch alarm untuk memantau DynamoDB](#).

Menggunakan kebijakan IAM untuk Amazon Kinesis Data Streams dan Amazon DynamoDB

Saat pertama kali Anda mengaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB, DynamoDB secara otomatis membuat peran terkait layanan (IAM) untuk Anda. AWS Identity and Access Management Peran ini, `AWSRoleForDynamoDBKinesisDataStreamsReplication`, memungkinkan DynamoDB untuk mengelola replikasi perubahan tingkat item ke Kinesis Data Streams atas nama Anda. Jangan hapus peran tertaut layanan ini.

Untuk informasi selengkapnya tentang peran tertaut layanan, lihat [Menggunakan peran tertaut layanan](#) di Panduan Pengguna IAM.

Untuk mengaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB, Anda harus memiliki izin berikut di tabel:

- `dynamodb:EnableKinesisStreamingDestination`
- `kinesis:ListStreams`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`

Untuk menjelaskan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel DynamoDB tertentu, Anda harus memiliki izin berikut pada tabel.

- `dynamodb:DescribeKinesisStreamingDestination`

- `kinesis:DescribeStreamSummary`
- `kinesis:DescribeStream`

Untuk menonaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB, Anda harus memiliki izin berikut pada tabel.

- `dynamodb:DisableKinesisStreamingDestination`

Untuk memperbarui Amazon Kinesis Data Streams untuk Amazon DynamoDB, Anda harus memiliki izin berikut di atas tabel.

- `dynamodb:UpdateKinesisStreamingDestination`

Contoh berikut menunjukkan cara menggunakan kebijakan IAM untuk memberikan izin untuk Amazon Kinesis Data Streams untuk Amazon DynamoDB.

Contoh: Mengaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB

Kebijakan IAM berikut memberikan izin untuk mengaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel. `Music` itu tidak memberikan izin untuk menonaktifkan, memperbarui atau menjelaskan Kinesis Data Streams untuk DynamoDB untuk tabel. `Music`

```
{
  "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-2:12345678901:table/Music"
  }
]
}
```

Contoh: Perbarui Amazon Kinesis Data Streams untuk Amazon DynamoDB

Kebijakan IAM berikut memberikan izin untuk memperbarui Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel. Music Itu tidak memberikan izin untuk mengaktifkan, menonaktifkan, atau menjelaskan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel. Music

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateKinesisStreamingDestination"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
    }
  ]
}
```

Contoh: Menonaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB

Kebijakan IAM berikut memberikan izin untuk menonaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel. Music Itu tidak memberikan izin untuk mengaktifkan, memperbarui, atau menjelaskan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel. Music

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DisableKinesisStreamingDestination"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
    }
  ]
}
```

```
]
}
```

Contoh: Selektif menerapkan izin untuk Amazon Kinesis Data Streams untuk Amazon DynamoDB berdasarkan sumber daya

Kebijakan IAM berikut memberikan izin untuk mengaktifkan dan menjelaskan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel, dan menolak izin untuk menonaktifkan Amazon Kinesis Data Streams Music untuk Amazon DynamoDB untuk tabel. `Orders`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:EnableKinesisStreamingDestination",
        "dynamodb:DescribeKinesisStreamingDestination"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
    },
    {
      "Effect": "Deny",
      "Action": [
        "dynamodb:DisableKinesisStreamingDestination"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Orders"
    }
  ]
}
```

Menggunakan peran tertaut layanan untuk Kinesis Data Streams untuk DynamoDB

[Amazon Kinesis Data Streams untuk Amazon DynamoDB menggunakan peran terkait layanan \(AWS Identity and Access Management IAM\)](#). Peran tertaut layanan adalah jenis IAM role unik yang ditautkan langsung ke Kinesis Data Streams untuk DynamoDB. Peran terkait layanan telah ditentukan sebelumnya oleh Kinesis Data Streams untuk DynamoDB dan menyertakan semua izin yang diperlukan layanan untuk memanggil layanan lain atas nama Anda. AWS

Peran tertaut layanan membuat pengaturan Kinesis Data Streams untuk DynamoDB lebih mudah karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. Kinesis Data Streams

untuk DynamoDB menentukan izin peran terkait layanannya, dan kecuali ditentukan lain, hanya Kinesis Data Streams untuk DynamoDB yang dapat mengambil peran tersebut. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, dan kebijakan izin tersebut tidak dapat dilampirkan ke entitas IAM lainnya.

Untuk informasi tentang layanan lain yang mendukung peran terkait layanan, lihat [Layanan AWS yang Berfungsi dengan IAM](#) dan cari layanan yang memiliki Ya di kolom Peran Terkait Layanan. Pilih Ya dengan tautan untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Izin peran tertaut layanan untuk Kinesis Data Streams untuk DynamoDB

Kinesis Data Streams untuk DynamoDB menggunakan peran terkait layanan bernama `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`. Tujuan dari peran tertaut layanan adalah untuk memungkinkan Amazon DynamoDB mengelola replikasi perubahan tingkat item pada Kinesis Data Streams, atas nama Anda.

Peran tertaut layanan `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` memercayai layanan berikut untuk mengambil peran tersebut:

- `kinesisreplication.dynamodb.amazonaws.com`

Kebijakan izin peran memungkinkan Kinesis Data Streams for DynamoDB menyelesaikan tindakan berikut pada sumber daya yang ditentukan:

- Tindakan: `Put records and describe` pada Kinesis stream
- Tindakan: `Generate data keys` aktif AWS KMS untuk menempatkan data pada aliran Kinesis yang dienkripsi menggunakan kunci Buatan Pengguna. AWS KMS

Untuk isi dokumen kebijakan yang tepat, lihat [DynamoDB KinesisReplicationServiceRolePolicy](#).

Anda harus mengonfigurasi izin untuk mengizinkan entitas IAM (seperti pengguna, grup, atau peran) untuk membuat, mengedit, atau menghapus peran terkait layanan. Untuk informasi lebih lanjut, lihat [Izin Peran Tertaut Layanan](#) di Panduan Pengguna IAM.

Membuat peran tertaut layanan untuk Kinesis Data Streams untuk DynamoDB

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda mengaktifkan Kinesis Data Streams untuk AWS Management Console DynamoDB di AWS CLI, atau AWS API, Kinesis Data Streams untuk DynamoDB akan membuat peran terkait layanan untuk Anda.

Jika Anda menghapus peran terkait layanan ini, dan ingin membuatnya lagi, Anda dapat mengulangi proses yang sama untuk membuat kembali peran tersebut di akun Anda. Saat Anda mengaktifkan Kinesis Data Streams untuk DynamoDB, Kinesis Data Streams untuk DynamoDB menciptakan peran terkait layanan untuk Anda lagi.

Mengedit peran tertaut layanan untuk Kinesis Data Streams untuk DynamoDB

Kinesis Data Streams untuk DynamoDB tidak mengizinkan Anda mengedit peran tertaut layanan `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`. Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin mereferensikan peran tersebut. Namun, Anda dapat mengedit deskripsi peran ini menggunakan IAM. Untuk informasi selengkapnya, lihat [Mengedit Peran terkait Layanan](#) di Panduan Pengguna IAM.

Menghapus peran tertaut layanan untuk Kinesis Data Streams untuk DynamoDB

Anda juga dapat menggunakan konsol IAM, AWS CLI atau AWS API untuk menghapus peran terkait layanan secara manual. Untuk melakukan ini, Anda harus membersihkan sumber daya untuk peran yang terhubung dengan layanan lalu Anda dapat menghapusnya secara manual.

Note

Jika layanan Kinesis Data Streams for DynamoDB menggunakan peran tersebut saat Anda mencoba menghapus sumber daya, maka penghapusan mungkin gagal. Jika hal itu terjadi, tunggu beberapa menit dan coba lagi.

Untuk menghapus peran terkait layanan secara manual menggunakan IAM

Gunakan konsol IAM, the AWS CLI, atau AWS API untuk menghapus peran `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` terkait layanan. Untuk informasi selengkapnya, lihat [Menghapus Peran Terkait Layanan](#) di Panduan Pengguna IAM.

Ubah tangkapan data DynamoDB Streams

DynamoDB Streams menangkap urutan waktu modifikasi tingkat item di tabel DynamoDB mana pun dan menyimpan informasi ini dalam log hingga 24 jam. Aplikasi dapat mengakses log ini dan melihat item data seperti yang muncul sebelum dan setelah diubah, hampir secara waktu nyata.

Enkripsi saat istirahat mengenkripsi data dalam DynamoDB streams. Untuk informasi selengkapnya, lihat [Enkripsi DynamoDB saat diam](#).

Streami DynamoDB adalah aliran berurutan informasi tentang perubahan item dalam tabel DynamoDB. Saat Anda mengaktifkan aliran pada tabel, DynamoDB menangkap informasi tentang setiap modifikasi pada item data dalam tabel.

Setiap kali aplikasi membuat, memperbarui, atau menghapus item dalam tabel, DynamoDB Streams menulis rekaman aliran dengan atribut kunci primer dari item yang diubah. Catatan stream berisi informasi tentang modifikasi data pada satu item dalam tabel DynamoDB. Anda dapat mengonfigurasi aliran sehingga rekaman aliran menangkap informasi tambahan, seperti gambar "sebelum" dan "sesudah" dari item yang diubah.

DynamoDB Streams membantu memastikan hal berikut:

- Setiap catatan stream muncul tepat sekali dalam stream.
- Untuk setiap item yang diubah dalam tabel DynamoDB, catatan stream muncul dalam urutan yang sama seperti modifikasi aktual untuk item.

DynamoDB Streams menulis catatan aliran hampir secara real-time sehingga Anda dapat membangun aplikasi yang menggunakan aliran ini dan mengambil tindakan berdasarkan kontennya.

Topik

- [Titik akhir untuk DynamoDB Streams](#)
- [Mengaktifkan aliran](#)
- [Membaca dan memproses aliran](#)
- [DynamoDB Streams dan Waktu untuk Tayang](#)
- [Menggunakan adaptor DynamoDB Streams Kinesis untuk memproses catatan aliran](#)
- [API tingkat rendah DynamoDB Streams: Contoh Java](#)
- [DynamoDB Streams dan pemicu AWS Lambda](#)

Titik akhir untuk DynamoDB Streams

AWS mempertahankan endpoint terpisah untuk DynamoDB dan DynamoDB Streams. Untuk bekerja dengan tabel dan indeks database, aplikasi Anda harus mengakses titik akhir DynamoDB. Untuk membaca dan memproses catatan DynamoDB Streams, aplikasi Anda harus mengakses titik akhir DynamoDB Streams di Wilayah yang sama.

Konvensi penamaan untuk titik akhir DynamoDB Streams adalah `streams.dynamodb.<region>.amazonaws.com`. Misalnya, jika Anda menggunakan titik akhir

`dynamodb.us-west-2.amazonaws.com` untuk mengakses DynamoDB, Anda akan menggunakan titik akhir `streams.dynamodb.us-west-2.amazonaws.com` untuk mengakses DynamoDB Streams.

Note

Untuk daftar lengkap Wilayah dan titik akhir DynamoDB dan DynamoDB Streams, lihat [Wilayah dan titik akhir](#) di Referensi Umum AWS.

AWS SDK menyediakan klien terpisah untuk DynamoDB dan DynamoDB Streams. Bergantung pada kebutuhan Anda, aplikasi Anda dapat mengakses titik akhir DynamoDB, titik akhir DynamoDB Streams, atau keduanya secara bersamaan. Untuk terhubung ke kedua titik akhir, aplikasi Anda harus membuat instance dua klien—satu untuk DynamoDB dan satu lagi untuk DynamoDB Streams.

Mengaktifkan aliran

Anda dapat mengaktifkan aliran pada tabel baru saat Anda membuatnya menggunakan AWS CLI atau salah satu AWS SDK. Anda juga dapat mengaktifkan atau menonaktifkan aliran pada tabel yang sudah ada, atau mengubah pengaturan aliran. DynamoDB Streams beroperasi secara asinkron, sehingga tidak ada dampak kinerja pada tabel jika Anda mengaktifkan aliran.

Cara termudah untuk mengelola DynamoDB Streams adalah dengan menggunakan AWS Management Console.

1. [Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/.](https://console.aws.amazon.com/dynamodb/)
2. Pada dasbor konsol DynamoDB, pilih Tabel dan pilih tabel yang ada.
3. Pilih tab Ekspor dan aliran.
4. Di bagian detail aliran DynamoDB, pilih Aktifkan.
5. Pada halaman Aliran DynamoDB, pilih informasi yang akan ditulis ke aliran setiap kali data dalam tabel diubah:
 - Atribut kunci saja — Hanya atribut kunci dari item yang dimodifikasi.
 - Gambar baru — Seluruh item, saat muncul setelah diubah.
 - Gambar lama — Seluruh item, saat muncul sebelum diubah.
 - Gambar baru dan lama — Baik gambar baru dan lama dari item.

Ketika pengaturan seperti yang Anda inginkan, pilih Aktifkan aliran.

6. (Opsional) Untuk menonaktifkan aliran yang ada, pilih Matikan di bawah detail aliran DynamoDB.

Anda juga dapat menggunakan operasi `CreateTable` atau `UpdateTable` API untuk mengaktifkan atau memodifikasi aliran. Parameter `StreamSpecification` menentukan bagaimana aliran dikonfigurasi:

- `StreamEnabled` — Menentukan apakah stream diaktifkan (`true`) atau dinonaktifkan (`false`) untuk tabel.
- `StreamViewType` — Menentukan informasi yang akan ditulis ke stream setiap kali data dalam tabel dimodifikasi:
 - `KEYS_ONLY` — Hanya atribut kunci dari item yang dimodifikasi.
 - `NEW_IMAGE` — Keseluruhan item, seperti yang muncul setelah diubah.
 - `OLD_IMAGE` — Keseluruhan item, seperti yang terlihat sebelum diubah.
 - `NEW_AND_OLD_IMAGES` — Baik gambar baru dan lama dari item.

Anda dapat mengaktifkan atau menonaktifkan stream kapan saja. Namun, Anda menerima `ResourceInUseException` jika Anda mencoba untuk mengaktifkan stream pada tabel yang sudah memiliki stream. Anda menerima `ValidationException` jika Anda mencoba menonaktifkan stream pada tabel yang tidak memiliki stream.

Ketika Anda mengatur `StreamEnabled` ke `true`, DynamoDB menciptakan stream baru dengan deskriptor stream unik yang ditugaskan untuk itu. Jika Anda menonaktifkan dan kemudian mengaktifkan kembali stream pada tabel, stream baru dibuat dengan deskriptor stream yang berbeda.

Setiap stream diidentifikasi secara unik oleh Amazon Resource Name (ARN). Berikut ini adalah contoh ARN untuk stream pada tabel DynamoDB bernama `TestTable`.

```
arn:aws:dynamodb:us-west-2:111122223333:table/TestTable/stream/2015-05-11T21:21:33.291
```

Untuk menentukan deskriptor stream terbaru untuk tabel, terbitkan permintaan `DescribeTable` DynamoDB dan cari elemen `LatestStreamArn` dalam respons.

Note

Hal ini tidak mungkin untuk mengedit `StreamViewType` setelah stream telah diatur. Jika Anda perlu melakukan perubahan pada stream setelah pengaturan, Anda harus menonaktifkan stream saat ini dan membuat yang baru.

Membaca dan memproses aliran

Untuk membaca dan memproses aliran, aplikasi Anda harus terhubung ke titik akhir DynamoDB Streams dan menerbitkan permintaan API.

Stream terdiri dari catatan aliran. Setiap catatan stream mewakili modifikasi data tunggal dalam tabel DynamoDB di mana aliran berada. Setiap catatan stream ditugaskan nomor urut, mencerminkan urutan catatan diterbitkan ke aliran.

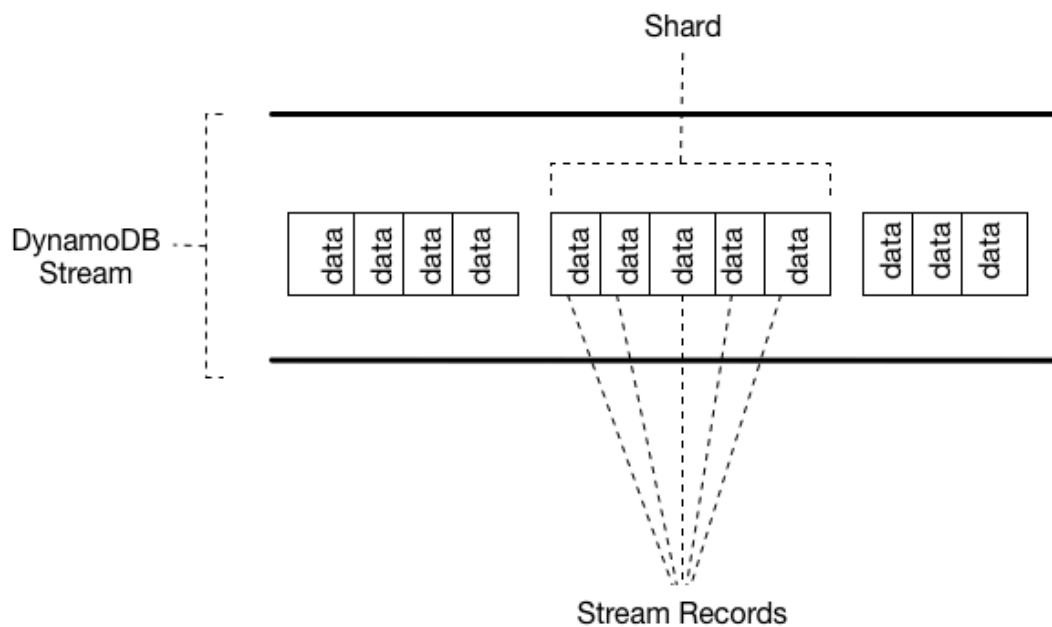
Rekaman streaming disusun ke dalam grup, atau serpihan. Setiap serpihan bertindak sebagai kontainer untuk beberapa catatan aliran, dan berisi informasi yang diperlukan untuk mengakses dan iterasi melalui catatan-catatan ini. Catatan stream dalam serpihan dihapus secara otomatis setelah 24 jam.

Serpihan bersifat fana: Mereka dibuat dan dihapus secara otomatis, sesuai kebutuhan. Setiap serpihan juga dapat dibagi menjadi beberapa serpihan baru; ini juga terjadi secara otomatis. (Pecahan induk juga mungkin hanya memiliki satu pecahan turunan.) Sebuah pecahan mungkin terpecah sebagai respons terhadap aktivitas tulis tingkat tinggi pada tabel induknya, sehingga aplikasi dapat memproses rekaman dari beberapa pecahan secara paralel.

Jika Anda menonaktifkan aliran, semua pecahan yang terbuka akan ditutup. Data dalam aliran akan terus dapat dibaca selama 24 jam.

Karena serpihan memiliki garis keturunan (induk dan turunan), aplikasi harus selalu memproses serpihan induk sebelum memproses serpihan anak. Hal ini membantu memastikan bahwa rekaman aliran juga diproses dalam urutan yang benar. (Jika Anda menggunakan DynamoDB Streams Kinesis Adapter, hal ini akan ditangani untuk Anda. Aplikasi Anda memproses pecahan dan catatan aliran dalam urutan yang benar. Secara otomatis menangani pecahan baru atau yang sudah habis masa berlakunya, selain pecahan yang terpecah saat aplikasi berjalan. Misalnya informasi lebih lanjut, lihat [Menggunakan adaptor DynamoDB Streams Kinesis untuk memproses catatan aliran.](#))

Diagram berikut menunjukkan hubungan antara aliran, serpihan dalam aliran, dan rekaman aliran dalam serpihan.



Note

Jika Anda melakukan operasi `PutItem` atau `UpdateItem` yang tidak mengubah data dalam item, DynamoDB Streams tidak menulis catatan stream untuk operasi itu.

Untuk mengakses aliran dan memproses rekaman aliran di dalamnya, Anda harus melakukan hal berikut:

- Tentukan ARN unik aliran yang ingin Anda akses.
- Tentukan pecahan mana dalam aliran yang berisi rekaman aliran yang Anda minati.
- Akses pecahan dan ambil rekaman aliran yang Anda inginkan.

Note

Maksimum dua proses harus membaca dari pecahan aliran yang sama pada waktu yang sama. Memiliki lebih dari dua pembaca per shard dapat mengakibatkan throttling.

DynamoDB Streams API menyediakan tindakan berikut untuk digunakan oleh program aplikasi:

- [ListStreams](#) — Mengembalikan daftar deskriptor stream untuk akun dan titik akhir saat ini. Anda secara opsional dapat meminta hanya deskriptor stream untuk nama tabel tertentu.
- [DescribeStream](#) — Mengembalikan informasi mendetail tentang stream tertentu. Output termasuk daftar serpihan terkait dengan stream, termasuk ID serpihan.
- [GetShardIterator](#) — Mengembalikan iterator serpihan, yang menggambarkan lokasi di dalam serpihan. Anda dapat meminta iterator yang menyediakan akses ke titik terlama, titik terbaru, atau titik tertentu dalam serpihan.
- [GetRecords](#) - Mengembalikan catatan aliran dari dalam serpihan yang diberikan. Anda harus memberikan iterator serpihan yang dikembalikan dari permintaan `GetShardIterator`.

Untuk deskripsi lengkap operasi API ini, termasuk contoh permintaan dan tanggapan, lihat [Referensi API Amazon DynamoDB Streams](#).

Batas retensi data untuk DynamoDB Streams

Semua data dalam DynamoDB Streams tunduk pada masa hidup 24 jam. Anda dapat mengambil dan menganalisis 24 jam terakhir aktivitas untuk setiap tabel yang ditentukan. Namun, data yang lebih lama dari 24 jam rentan terhadap pemangkasan (penghapusan) setiap saat.

Jika Anda menonaktifkan stream pada tabel, data di stream terus dapat dibaca selama 24 jam. Setelah waktu ini, data kedaluwarsa dan catatan stream secara otomatis dihapus. Tidak ada mekanisme untuk menghapus stream yang ada secara manual. Anda harus menunggu hingga batas retensi berakhir (24 jam), dan semua catatan stream akan dihapus.

DynamoDB Streams dan Waktu untuk Tayang

Anda dapat mencadangkan, atau memproses, item yang dihapus oleh [Waktu untuk Tayang](#) (TTL) dengan mengaktifkan Amazon DynamoDB Stream pada tabel dan memproses catatan stream dari item yang kedaluwarsa. Untuk informasi selengkapnya, lihat [Membaca dan memproses aliran](#).

Catatan stream berisi bidang identitas pengguna `Records[<index>].userIdentity`.

Item yang dihapus oleh proses Waktu untuk Tayang setelah kedaluwarsa memiliki bidang berikut:

- `Records[<index>].userIdentity.type`
"Service"
- `Records[<index>].userIdentity.principalId`

```
"dynamodb.amazonaws.com"
```

Note

Saat Anda menggunakan TTL dalam tabel global, wilayah tempat TTL dilakukan akan memiliki `userIdentity` bidang yang ditetapkan. Bidang ini tidak akan disetel di wilayah lain saat penghapusan direplikasi.

JSON berikut menunjukkan bagian yang relevan dari catatan stream tunggal.

```
"Records": [  
  {  
    ...  
    "userIdentity": {  
      "type": "Service",  
      "principalId": "dynamodb.amazonaws.com"  
    }  
    ...  
  }  
]
```

Menggunakan DynamoDB Streams dan Lambda untuk mengarsipkan item TTL yang dihapus

Menggabungkan [Waktu untuk Tayang \(TTL\) DynamoDB, DynamoDB Streams](#), dan [AWS Lambda](#) dapat membantu menyederhanakan pengarsipan [data, mengurangi biaya penyimpanan DynamoDB](#), dan mengurangi kompleksitas kode. Menggunakan Lambda sebagai konsumen aliran memberikan banyak keuntungan, terutama pengurangan biaya dibandingkan dengan konsumen lain seperti Kinesis Client Library (KCL). Anda tidak dikenakan biaya untuk panggilan API `GetRecords` di aliran DynamoDB saat menggunakan Lambda untuk mengkonsumsi peristiwa, dan Lambda dapat menyediakan pemfilteran peristiwa dengan mengidentifikasi pola JSON dalam peristiwa streaming. Dengan pemfilteran konten pola peristiwa, Anda dapat menentukan hingga lima filter berbeda untuk mengontrol peristiwa mana yang dikirim ke Lambda untuk diproses. Hal ini membantu mengurangi pemanggilan fungsi Lambda Anda, menyederhanakan kode, dan mengurangi biaya keseluruhan.

Meskipun DynamoDB Streams berisi semua modifikasi data, seperti tindakan `Create`, `Modify`, dan `Remove`, hal ini dapat mengakibatkan pemanggilan yang tidak diinginkan pada fungsi Lambda arsip Anda. Misalnya, Anda memiliki tabel dengan 2 juta modifikasi data per jam yang mengalir ke aliran, namun kurang dari 5 persen di antaranya adalah penghapusan item yang akan kedaluwarsa melalui proses TTL dan perlu diarsipkan. Dengan [filter sumber acara Lambda](#), fungsi Lambda hanya akan dipanggil 100.000 kali per jam. Hasil dari pemfilteran peristiwa adalah Anda hanya dikenakan biaya untuk pemanggilan yang diperlukan, bukan 2 juta pemanggilan yang akan Anda dapatkan tanpa pemfilteran peristiwa.

Pemfilteran peristiwa diterapkan pada [pemetaan sumber peristiwa Lambda](#), yang merupakan sumber daya yang membaca dari peristiwa yang dipilih—aliran DynamoDB—dan memanggil fungsi Lambda. Dalam diagram berikut, Anda dapat melihat bagaimana item Time to Live yang dihapus digunakan oleh fungsi Lambda menggunakan aliran dan filter peristiwa.



Pola filter peristiwa DynamoDB Waktu untuk Tayang

Menambahkan JSON berikut ke [kriteria filter](#) pemetaan sumber peristiwa Anda memungkinkan invokasi fungsi Lambda Anda hanya untuk item yang dihapus TTL:

```
{
  "Filters": [
    {
      "Pattern": { "userIdentity": { "type": ["Service"], "principalId":
["dynamodb.amazonaws.com"] } }
    }
  ]
}
```

Membuat pemetaan sumber AWS Lambda acara

Gunakan cuplikan kode berikut untuk membuat pemetaan sumber peristiwa terfilter yang dapat Anda sambungkan ke aliran DynamoDB tabel. Setiap blok kode menyertakan pola filter peristiwa.

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"]
  }
}

```

Menggunakan adaptor DynamoDB Streams Kinesis untuk memproses catatan aliran

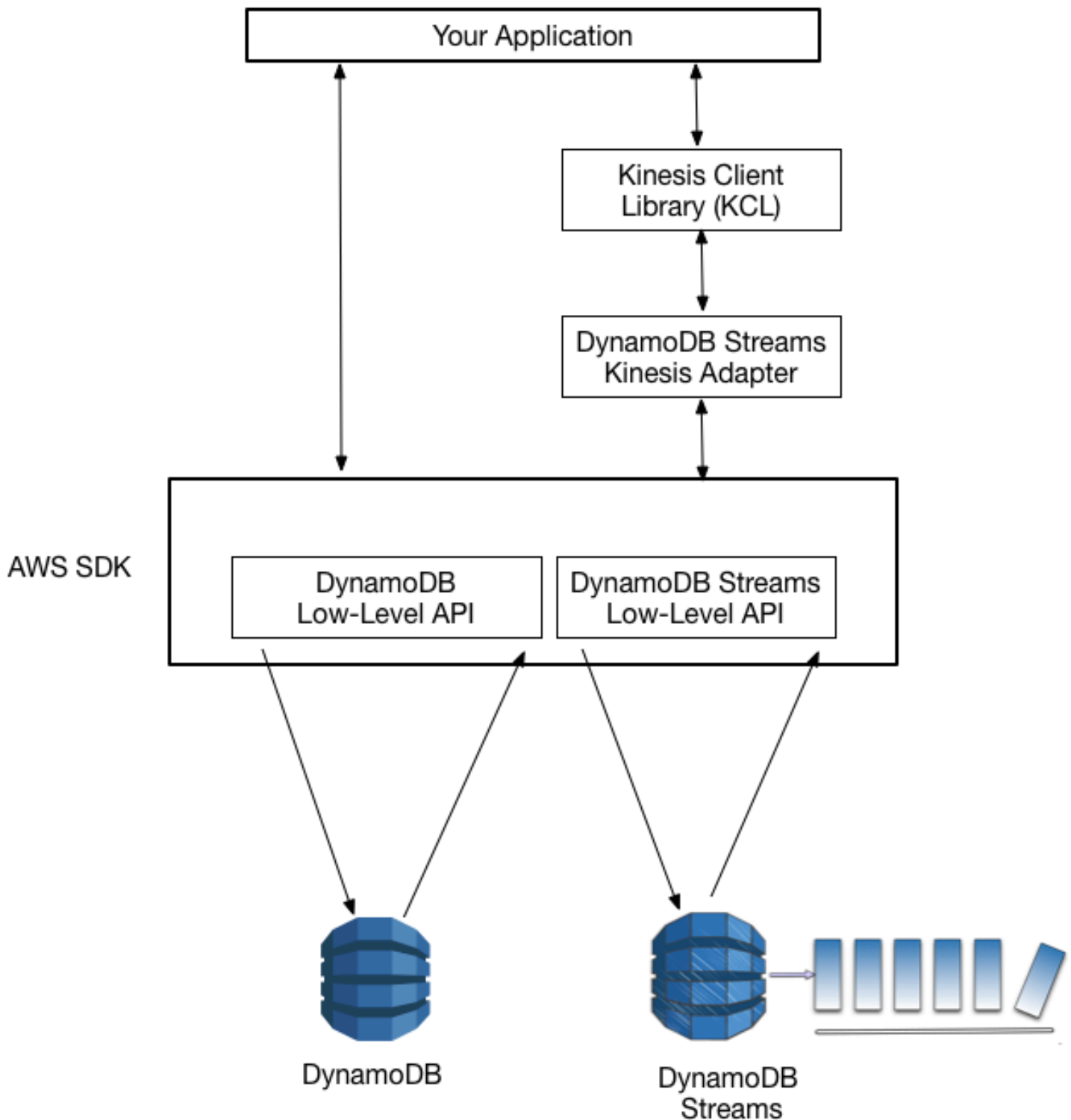
Menggunakan Adaptor Amazon Kinesis adalah cara yang disarankan untuk menggunakan aliran dari Amazon DynamoDB. API DynamoDB Streams sengaja dibuat serupa dengan Kinesis Data Streams, sebuah layanan untuk pemrosesan data streaming secara real-time dalam skala besar. Di kedua layanan, aliran data terdiri dari pecahan, yang merupakan wadah untuk rekaman aliran. API kedua layanan berisi operasi `ListStreams`, `DescribeStream`, `GetShards`, dan `GetShardIterator`. (Meskipun tindakan DynamoDB Streams ini serupa dengan tindakan serupa di Kinesis Data Streams, tindakan tersebut tidak 100 persen identik.)

Anda dapat menulis aplikasi untuk Kinesis Data Streams menggunakan Kinesis Client Library (KCL). KCL menyederhanakan pengkodean dengan menyediakan abstraksi yang berguna di atas Kinesis Data Streams API tingkat rendah. Untuk informasi selengkapnya tentang KCL, lihat [Mengembangkan konsumen menggunakan Kinesis client library](#) dalam Panduan Developer Amazon Kinesis Data Streams.

Sebagai pengguna DynamoDB Streams, Anda dapat menggunakan pola desain yang ditemukan dalam KCL untuk memproses serpihan dan rekaman aliran DynamoDB Streams. Untuk melakukan ini, Anda menggunakan Adaptor DynamoDB Streams Kinesis. Adaptor Kinesis mengimplementasikan antarmuka Kinesis Data Streams sehingga KCL dapat digunakan untuk menggunakan dan

memproses catatan dari DynamoDB Streams. [Untuk petunjuk tentang cara mengatur dan menginstal Adaptor Kinesis DynamoDB Streams, lihat repositori. GitHub](#)

Diagram berikut menunjukkan bagaimana perpustakaan ini berinteraksi satu sama lain.



Dengan Adaptor Kinesis DynamoDB Streams, Anda dapat mulai mengembangkan antarmuka KCL, dengan panggilan API diarahkan secara mulus ke titik akhir DynamoDB Streams.

Saat aplikasi Anda dimulai, aplikasi akan memanggil KCL untuk membuat instance pekerja. Anda harus memberi pekerja informasi konfigurasi untuk aplikasi, seperti deskriptor aliran dan AWS kredensial, dan nama kelas prosesor rekaman yang Anda berikan. Saat menjalankan kode di pemroses rekaman, pekerja melakukan tugas-tugas berikut:

- Menghubungkan ke aliran
- Menghitung pecahan dalam aliran
- Mengkoordinasikan asosiasi serpihan dengan pekerja lain (jika ada)
- Membuat instance pemroses rekaman untuk setiap pecahan yang dikelolanya
- Menarik catatan dari aliran
- Mendorong rekaman ke pemroses rekaman yang sesuai
- Catatan yang diproses di pos pemeriksaan
- Menyeimbangkan asosiasi pekerja pecahan ketika jumlah instans pekerja berubah
- Menyeimbangkan asosiasi pekerja pecahan saat pecahan dipisahkan

Note

Untuk deskripsi konsep KCL yang tercantum di sini, lihat [Mengembangkan konsumen menggunakan Kinesis client library](#) di Panduan Pengembang Amazon Kinesis Data Streams. Untuk informasi lebih lanjut tentang menggunakan stream dengan lihat [AWS Lambda DynamoDB Streams dan pemicu AWS Lambda](#)

Panduan: Adaptor DynamoDB Streams Kinesis

Bagian ini adalah panduan aplikasi Java yang menggunakan Perpustakaan Klien Amazon Kinesis dan Adaptor Amazon DynamoDB Streams Kinesis. Aplikasi ini memperlihatkan contoh replikasi data, di mana aktivitas penulisan dari satu tabel diterapkan ke tabel kedua, dengan konten kedua tabel tetap sinkron. Untuk kode sumber, lihat [Program lengkap: Adaptor DynamoDB Streams Kinesis](#).

Program ini melakukan hal berikut:

1. Menciptakan dua tabel DynamoDB bernama KCL-Demo-src dan KCL-Demo-dst. Masing-masing tabel ini memiliki stream yang diaktifkan.

2. Menghasilkan aktivitas pembaruan dalam tabel sumber dengan menambahkan, memperbarui, dan menghapus item. Hal ini menyebabkan data akan ditulis ke stream tabel.
3. Membaca catatan dari stream, merekonstruksinya sebagai permintaan DynamoDB, dan menerapkan permintaan ke tabel tujuan.
4. Memindai tabel sumber dan tujuan untuk memastikan bahwa isinya identik.
5. Membersihkan dengan menghapus tabel.

Langkah-langkah ini dijelaskan di bagian berikut, dan aplikasi lengkap ditampilkan di akhir panduan.

Topik

- [Langkah 1: Buat tabel DynamoDB](#)
- [Langkah 2: Hasilkan aktivitas pembaruan di tabel sumber](#)
- [Langkah 3: Proses alirannya](#)
- [Langkah 4: Pastikan bahwa kedua tabel memiliki isi identik](#)
- [Langkah 5: Bersihkan](#)
- [Program lengkap: Adaptor DynamoDB Streams Kinesis](#)

Langkah 1: Buat tabel DynamoDB

Langkah pertama adalah membuat dua tabel DynamoDB—tabel sumber dan tabel tujuan. `StreamViewType` pada aliran tabel sumber adalah `NEW_IMAGE`. Ini berarti bahwa setiap kali item diubah dalam tabel ini, gambar "setelah" item tersebut ditulis ke aliran. Dengan cara ini, aliran melacak semua aktivitas penulisan di tabel.

Contoh berikut menunjukkan kode yang digunakan untuk membuat kedua tabel.

```
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)
```

Langkah 2: Hasilkan aktivitas pembaruan di tabel sumber

Langkah selanjutnya adalah menghasilkan beberapa aktivitas menulis pada tabel sumber. Saat aktivitas ini berlangsung, aliran tabel sumber juga diperbarui hampir secara waktu nyata.

Aplikasi ini mendefinisikan kelas pembantu dengan metode yang memanggil operasi `PutItem`, `UpdateItem`, dan `API DeleteItem` untuk menulis data. Contoh kode berikut menunjukkan bagaimana metode ini digunakan.

```
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");
```

Langkah 3: Proses alirannya

Sekarang program mulai memproses aliran. Adaptor Kinesis DynamoDB Streams bertindak sebagai lapisan transparan antara KCL dan titik akhir DynamoDB Streams, sehingga kode dapat sepenuhnya menggunakan KCL daripada harus melakukan panggilan DynamoDB Streams tingkat rendah. Program ini melakukan tugas-tugas berikut:

- Ini mendefinisikan kelas prosesor catatan, `StreamsRecordProcessor`, dengan metode yang sesuai dengan definisi antarmuka KCL: `initialize`, `processRecords`, dan `shutdown`. Metode

`processRecords` berisi logika yang diperlukan untuk membaca dari stream tabel sumber dan menulis ke tabel tujuan.

- Ini mendefinisikan sebuah pabrik kelas untuk kelas prosesor catatan (`StreamsRecordProcessorFactory`). Hal ini diperlukan untuk program Java yang menggunakan KCL.
- Ini menginstanskan `KCL Worker` baru, yang terkait dengan pabrik kelas.
- Ini mematikan `Worker` saat pemrosesan catatan selesai.

Untuk mempelajari lebih lanjut tentang definisi antarmuka KCL, lihat [Mengembangkan konsumen menggunakan Kinesis client library](#) di Panduan Pengembang Amazon Kinesis Data Streams.

Contoh kode berikut menunjukkan loop utama dalam `StreamsRecordProcessor`. Pernyataan `case` menentukan apa tindakan apa yang harus dilakukan, berdasarkan `OperationType` yang muncul dalam catatan stream.

```
for (Record record : records) {
    String data = new String(record.getData().array(), Charset.forName("UTF-8"));
    System.out.println(data);
    if (record instanceof RecordAdapter) {
        com.amazonaws.services.dynamodbv2.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();
        }
    }
}
```

Langkah 4: Pastikan bahwa kedua tabel memiliki isi identik

Pada titik ini, isi sumber dan tujuan tabel tersinkronisasi. Aplikasi menerbitkan permintaan Scan terhadap kedua tabel untuk memverifikasi bahwa isinya, pada kenyataannya, identik.

Kelas DemoHelper berisi metode ScanTable yang memanggil API Scan tingkat rendah. Contoh berikut menunjukkan cara penggunaannya.

```
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!");
}
```

Langkah 5: Bersihkan

Demo selesai, sehingga aplikasi menghapus tabel sumber dan tujuan. Lihat contoh kode berikut. Bahkan setelah tabel dihapus, alirannya tetap tersedia hingga 24 jam, setelah itu tabel akan dihapus secara otomatis.

```
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
```

Program lengkap: Adaptor DynamoDB Streams Kinesis

Berikut ini adalah program Java lengkap yang melakukan tugas yang dijelaskan dalam [Panduan: Adaptor DynamoDB Streams Kinesis](#). Saat Anda menjalankannya, Anda akan melihat output yang serupa dengan yang seperti berikut.

```
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

Untuk menjalankan program ini, pastikan bahwa aplikasi klien memiliki akses ke DynamoDB dan CloudWatch Amazon menggunakan kebijakan. Untuk informasi selengkapnya, lihat [Kebijakan berbasis identitas untuk DynamoDB](#).

Kode sumber terdiri dari empat file .java:

- StreamsAdapterDemo.java
- StreamsRecordProcessor.java
- StreamsRecordProcessorFactory.java
- StreamsAdapterDemoHelper.java

StreamsAdapterDemo.java

```
package com.amazonaws.codesamples;

import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreams;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreamsClientBuilder;
import com.amazonaws.services.dynamodbv2.model.DeleteTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableResult;
import
    com.amazonaws.services.dynamodbv2.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import com.amazonaws.services.dynamodbv2.streamsadapter.StreamsWorkerFactory;
```



```
import
  com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
import
  com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;

public class StreamsAdapterDemo {
  private static Worker worker;
  private static KinesisClientLibConfiguration workerConfig;
  private static IRecordProcessorFactory recordProcessorFactory;

  private static AmazonDynamoDB dynamoDBClient;
  private static AmazonCloudWatch cloudWatchClient;
  private static AmazonDynamoDBStreams dynamoDBStreamsClient;
  private static AmazonDynamoDBStreamsAdapterClient adapterClient;

  private static String tablePrefix = "KCL-Demo";
  private static String streamArn;

  private static Regions awsRegion = Regions.US_EAST_2;

  private static AWSCredentialsProvider awsCredentialsProvider =
    DefaultAWSCredentialsProviderChain.getInstance();

  /**
   * @param args
   */
  public static void main(String[] args) throws Exception {
    System.out.println("Starting demo...");

    dynamoDBClient = AmazonDynamoDBClientBuilder.standard()
      .withRegion(awsRegion)
      .build();
    cloudWatchClient = AmazonCloudWatchClientBuilder.standard()
      .withRegion(awsRegion)
      .build();
    dynamoDBStreamsClient = AmazonDynamoDBStreamsClientBuilder.standard()
      .withRegion(awsRegion)
      .build();
    adapterClient = new AmazonDynamoDBStreamsAdapterClient(dynamoDBStreamsClient);
    String srcTable = tablePrefix + "-src";
    String destTable = tablePrefix + "-dest";
```

```
        recordProcessorFactory = new StreamsRecordProcessorFactory(dynamoDBClient,
destTable);

        setUpTables();

        workerConfig = new KinesisClientLibConfiguration("streams-adapter-demo",
            streamArn,
            awsCredentialsProvider,
            "streams-demo-worker")
            .withMaxRecords(1000)
            .withIdleTimeBetweenReadsInMillis(500)
            .withInitialPositionInStream(InitialPositionInStream.TRIM_HORIZON);

        System.out.println("Creating worker for stream: " + streamArn);
        worker =
StreamsWorkerFactory.createDynamoDbStreamsWorker(recordProcessorFactory, workerConfig,
adapterClient,
            dynamoDBClient, cloudWatchClient);
        System.out.println("Starting worker...");
        Thread t = new Thread(worker);
        t.start();

        Thread.sleep(25000);
        worker.shutdown();
        t.join();

        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!");
        }

        System.out.println("Done.");
        cleanupAndExit(0);
    }

    private static void setUpTables() {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        streamArn = StreamsAdapterDemoHelper.createTable(dynamoDBClient, srcTable);
        StreamsAdapterDemoHelper.createTable(dynamoDBClient, destTable);
    }
}
```

```
        awaitTableCreation(srcTable);

        performOps(srcTable);
    }

    private static void awaitTableCreation(String tableName) {
        Integer retries = 0;
        Boolean created = false;
        while (!created && retries < 100) {
            DescribeTableResult result =
StreamsAdapterDemoHelper.describeTable(dynamoDBClient, tableName);
            created = result.getTable().getTableStatus().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(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");
    }

    private static void cleanupAndExit(Integer returnValue) {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
        dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
        System.exit(returnValue);
    }
}
```

```
}
```

StreamsRecordProcessor.java

```
package com.amazonaws.codesamples;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.streamsadapter.model.RecordAdapter;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
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;
import com.amazonaws.services.kinesis.model.Record;

import java.nio.charset.Charset;

public class StreamsRecordProcessor implements IRecordProcessor {
    private Integer checkpointCounter;

    private final AmazonDynamoDB dynamoDBClient;
    private final String tableName;

    public StreamsRecordProcessor(AmazonDynamoDB dynamoDBClient2, String tableName) {
        this.dynamoDBClient = dynamoDBClient2;
        this.tableName = tableName;
    }

    @Override
    public void initialize(InitializationInput initializationInput) {
        checkpointCounter = 0;
    }

    @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) {
```

```
        com.amazonaws.services.dynamodbv2.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 {
            processRecordsInput.getCheckpoint().checkpoint();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

@Override
public void shutdown(ShutdownInput shutdownInput) {
    if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
        try {
            shutdownInput.getCheckpoint().checkpoint();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
}
```

StreamsRecordProcessorFactory.java

```
package com.amazonaws.codesamples;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements IRecordProcessorFactory {
    private final String tableName;
    private final AmazonDynamoDB dynamoDBClient;

    public StreamsRecordProcessorFactory(AmazonDynamoDB dynamoDBClient, String
tableName) {
        this.tableName = tableName;
        this.dynamoDBClient = dynamoDBClient;
    }

    @Override
    public IRecordProcessor createProcessor() {
        return new StreamsRecordProcessor(dynamoDBClient, tableName);
    }
}
```

StreamsAdapterDemoHelper.java

```
package com.amazonaws.codesamples;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.model.AttributeAction;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.AttributeValueUpdate;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.CreateTableResult;
import com.amazonaws.services.dynamodbv2.model.DeleteItemRequest;
```

```
import com.amazonaws.services.dynamodbv2.model.DescribeTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableResult;
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.PutItemRequest;
import com.amazonaws.services.dynamodbv2.model.ResourceInUseException;
import com.amazonaws.services.dynamodbv2.model.ScanRequest;
import com.amazonaws.services.dynamodbv2.model.ScanResult;
import com.amazonaws.services.dynamodbv2.model.StreamSpecification;
import com.amazonaws.services.dynamodbv2.model.StreamViewType;
import com.amazonaws.services.dynamodbv2.model.UpdateItemRequest;

public class StreamsAdapterDemoHelper {

    /**
     * @return StreamArn
     */
    public static String createTable(AmazonDynamoDB client, String tableName) {
        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)
```

```
        try {
            System.out.println("Creating table " + tableName);
            CreateTableResult result = client.createTable(createTableRequest);
            return result.getTableDescription().getLatestStreamArn();
        } catch (ResourceInUseException e) {
            System.out.println("Table already exists.");
            return describeTable(client, tableName).getTable().getLatestStreamArn();
        }
    }

    public static DescribeTableResult describeTable(AmazonDynamoDB client, String
tableName) {
        return client.describeTable(new
DescribeTableRequest().withTableName(tableName));
    }

    public static ScanResult scanTable(AmazonDynamoDB dynamoDBClient, String tableName)
{
        return dynamoDBClient.scan(new ScanRequest().withTableName(tableName));
    }

    public static void putItem(AmazonDynamoDB dynamoDBClient, String tableName, String
id, String val) {
        java.util.Map<String, AttributeValue> item = new HashMap<String,
AttributeValue>();
        item.put("Id", new AttributeValue().withN(id));
        item.put("attribute-1", new AttributeValue().withS(val));

        PutItemRequest putItemRequest = new
PutItemRequest().withTableName(tableName).withItem(item);
        dynamoDBClient.putItem(putItemRequest);
    }

    public static void putItem(AmazonDynamoDB dynamoDBClient, String tableName,
        java.util.Map<String, AttributeValue> items) {
        PutItemRequest putItemRequest = new
PutItemRequest().withTableName(tableName).withItem(items);
        dynamoDBClient.putItem(putItemRequest);
    }

    public static void updateItem(AmazonDynamoDB dynamoDBClient, String tableName,
String id, String val) {
        java.util.Map<String, AttributeValue> key = new HashMap<String,
AttributeValue>();
```



```
key.put("Id", new AttributeValue().withN(id));

Map<String, AttributeValueUpdate> attributeUpdates = new HashMap<String,
AttributeValueUpdate>();
AttributeValueUpdate update = new
AttributeValueUpdate().withAction(AttributeAction.PUT)
.withValue(new AttributeValue().withS(val));
attributeUpdates.put("attribute-2", update);

UpdateItemRequest updateItemRequest = new
UpdateItemRequest().withTableName(tableName).withKey(key)
.withAttributeUpdates(attributeUpdates);
dynamoDBClient.updateItem(updateItemRequest);
}

public static void deleteItem(AmazonDynamoDB dynamoDBClient, String tableName,
String id) {
    java.util.Map<String, AttributeValue> key = new HashMap<String,
AttributeValue>();
    key.put("Id", new AttributeValue().withN(id));

    DeleteItemRequest deleteItemRequest = new
DeleteItemRequest().withTableName(tableName).withKey(key);
    dynamoDBClient.deleteItem(deleteItemRequest);
}
}
```

API tingkat rendah DynamoDB Streams: Contoh Java

Note

Kode di halaman ini tidak lengkap dan tidak menangani semua skenario untuk menggunakan Amazon DynamoDB Streams. Cara yang disarankan untuk menggunakan catatan aliran dari DynamoDB adalah melalui Adaptor Amazon Kinesis menggunakan Kinesis Client Library (KCL), seperti yang dijelaskan dalam [Menggunakan adaptor DynamoDB Streams Kinesis untuk memproses catatan aliran](#).

Bagian ini berisi program Java yang menunjukkan aksi DynamoDB Streams. Program ini melakukan hal berikut: Program ini melakukan hal berikut:

1. Membuat tabel DynamoDB dengan aliran diaktifkan.
2. Menjelaskan pengaturan aliran untuk tabel ini.
3. Memodifikasi data dalam tabel.
4. Menjelaskan pecahan di aliran.
5. Membaca catatan aliran dari serpihan.
6. Membersihkan.

Saat Anda menjalankan program, Anda akan melihat output seperti berikut.

```
Issuing CreateTable request for TestTableForStreams
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:
01234567890...,},}
  Shard iterator: EjYFEkX2a26eVTWe...
    ApproximateCreationDateTime: Tue Mar 20 09:50:00 PDT 2018,Keys:
    {Id={N: 1,}},NewImage: {Message={S: New item!,}, Id={N: 1,}},SequenceNumber:
    1000000000003218256368,SizeBytes: 24,StreamViewType: NEW_AND_OLD_IMAGES
      {ApproximateCreationDateTime: Tue Mar 20 09:50:00 PDT 2018,Keys: {Id={N:
    1,}},NewImage: {Message={S: This item has changed,}, Id={N: 1,}},OldImage:
    {Message={S: New item!,}, Id={N: 1,}},SequenceNumber: 2000000000003218256412,SizeBytes:
    56,StreamViewType: NEW_AND_OLD_IMAGES}
        {ApproximateCreationDateTime: Tue Mar 20 09:50:00 PDT 2018,Keys: {Id={N:
    1,}},OldImage: {Message={S: This item has changed,}, Id={N: 1,}},SequenceNumber:
    3000000000003218256413,SizeBytes: 36,StreamViewType: NEW_AND_OLD_IMAGES}
      ...
    Deleting the table...
  Demo complete
```

Example

```
package com.amazon.codesamples;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreams;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreamsClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeAction;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.AttributeValueUpdate;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeStreamRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeStreamResult;
import com.amazonaws.services.dynamodbv2.model.DescribeTableResult;
import com.amazonaws.services.dynamodbv2.model.GetRecordsRequest;
import com.amazonaws.services.dynamodbv2.model.GetRecordsResult;
import com.amazonaws.services.dynamodbv2.model.GetShardIteratorRequest;
import com.amazonaws.services.dynamodbv2.model.GetShardIteratorResult;
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.Record;
import com.amazonaws.services.dynamodbv2.model.Shard;
import com.amazonaws.services.dynamodbv2.model.ShardIteratorType;
import com.amazonaws.services.dynamodbv2.model.StreamSpecification;
import com.amazonaws.services.dynamodbv2.model.StreamViewType;
import com.amazonaws.services.dynamodbv2.util.TableUtils;

public class StreamsLowLevelDemo {

    public static void main(String args[]) throws InterruptedException {
```

```
AmazonDynamoDB dynamoDBClient = AmazonDynamoDBClientBuilder
    .standard()
    .withRegion(Regions.US_EAST_2)
    .withCredentials(new
DefaultAWSCredentialsProviderChain())
    .build();

AmazonDynamoDBStreams streamsClient =
AmazonDynamoDBStreamsClientBuilder
    .standard()
    .withRegion(Regions.US_EAST_2)
    .withCredentials(new
DefaultAWSCredentialsProviderChain())
    .build();

// Create a table, with a stream enabled
String tableName = "TestTableForStreams";

ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>(
    Arrays.asList(new AttributeDefinition()
        .withAttributeName("Id")
        .withAttributeType("N")));

ArrayList<KeySchemaElement> keySchema = new ArrayList<>(
    Arrays.asList(new KeySchemaElement()
        .withAttributeName("Id")
        .withKeyType(KeyType.HASH))); //
Partition key

StreamSpecification streamSpecification = new StreamSpecification()
    .withStreamEnabled(true)
    .withStreamViewType(StreamViewType.NEW_AND_OLD_IMAGES);

CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)

.withKeySchema(keySchema).withAttributeDefinitions(attributeDefinitions)
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits(10L)
        .withWriteCapacityUnits(10L))
    .withStreamSpecification(streamSpecification);

System.out.println("Issuing CreateTable request for " + tableName);
dynamoDBClient.createTable(createTableRequest);
```

```
System.out.println("Waiting for " + tableName + " to be created...");

try {
    TableUtils.waitUntilActive(dynamoDBClient, tableName);
} catch (AmazonClientException e) {
    e.printStackTrace();
}

// Print the stream settings for the table
DescribeTableResult describeTableResult =
dynamoDBClient.describeTable(tableName);
String streamArn = describeTableResult.getTable().getLatestStreamArn();
System.out.println("Current stream ARN for " + tableName + ": " +
    describeTableResult.getTable().getLatestStreamArn());
StreamSpecification streamSpec =
describeTableResult.getTable().getStreamSpecification();
System.out.println("Stream enabled: " + streamSpec.getStreamEnabled());
System.out.println("Update view type: " +
streamSpec.getStreamViewType());
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
    Map<String, AttributeValue> item = new HashMap<>();
    item.put("Id", new AttributeValue().withN(i.toString()));
    item.put("Message", new AttributeValue().withS("New item!"));
    dynamoDBClient.putItem(tableName, item);

    // Update the item
    Map<String, AttributeValue> key = new HashMap<>();
    key.put("Id", new AttributeValue().withN(i.toString()));
    Map<String, AttributeValueUpdate> attributeUpdates = new
HashMap<>();
    attributeUpdates.put("Message", new AttributeValueUpdate()
        .withAction(AttributeAction.PUT)
        .withValue(new AttributeValue()
```

```
        .withS("This item has
changed"))));
        dynamoDBClient.updateItem(tableName, key, attributeUpdates);

        // Delete the item
        dynamoDBClient.deleteItem(tableName, key);
    }

    // Get all the shard IDs from the stream. Note that DescribeStream
returns
    // the shard IDs one page at a time.
    String lastEvaluatedShardId = null;

    do {
        DescribeStreamResult describeStreamResult =
streamsClient.describeStream(
            new DescribeStreamRequest()
                .withStreamArn(streamArn)

.withExclusiveStartShardId(lastEvaluatedShardId));
        List<Shard> shards =
describeStreamResult.getStreamDescription().getShards();

        // Process each shard on this page

        for (Shard shard : shards) {
            String shardId = shard.getShardId();
            System.out.println("Shard: " + shard);

            // Get an iterator for the current shard

            GetShardIteratorRequest getShardIteratorRequest = new
GetShardIteratorRequest()
                .withStreamArn(streamArn)
                .withShardId(shardId)

.withShardIteratorType(ShardIteratorType.TRIM_HORIZON);
            GetShardIteratorResult getShardIteratorResult =
streamsClient

.getShardIterator(getShardIteratorRequest);
            String currentShardIter =
getShardIteratorResult.getShardIterator();
```

```

// Shard iterator is not null until the Shard is sealed
(marked as READ_ONLY).
// To prevent running the loop until the Shard is
sealed, which will be on
// average
// 4 hours, we process only the items that were written
into DynamoDB and then
// exit.
int processedRecordCount = 0;
while (currentShardIter != null && processedRecordCount
< maxItemCount) {
    System.out.println("    Shard iterator: " +
currentShardIter.substring(380));

    // Use the shard iterator to read the stream
records

    GetRecordsResult getRecordsResult =
streamsClient
                                .getRecords(new
GetRecordsRequest()

.withShardIterator(currentShardIter));
    List<Record> records =
getRecordsResult.getRecords();
    for (Record record : records) {
        System.out.println("        " +
record.getDynamodb());
    }
    processedRecordCount += records.size();
    currentShardIter =
getRecordsResult.getNextShardIterator();
}
}

// If LastEvaluatedShardId is set, then there is
// at least one more page of shard IDs to retrieve
lastEvaluatedShardId =
describeStreamResult.getStreamDescription().getLastEvaluatedShardId();

} while (lastEvaluatedShardId != null);

// Delete the table
System.out.println("Deleting the table...");

```

```
        dynamoDBClient.deleteTable(tableName);

        System.out.println("Demo complete");
    }
}
```

DynamoDB Streams dan pemicu AWS Lambda

Topik

- [Tutorial #1: Menggunakan filter untuk memproses semua peristiwa dengan Amazon AWS Lambda DynamoDB dan menggunakan AWS CLI](#)
- [Tutorial #2: Menggunakan filter untuk memproses beberapa peristiwa dengan DynamoDB dan Lambda.](#)
- [Praktik terbaik dengan Lambda](#)

Amazon DynamoDB terintegrasi AWS Lambda sehingga Anda dapat membuat pemicu —potongan kode yang secara otomatis merespons peristiwa di DynamoDB Streams. Dengan pemicu, Anda dapat membangun aplikasi yang bereaksi terhadap modifikasi data di tabel DynamoDB.

Jika Anda mengaktifkan DynamoDB Streams pada tabel, Anda dapat mengaitkan aliran Amazon Resource Name (ARN) dengan fungsi yang Anda tulis. AWS Lambda Semua tindakan mutasi pada tabel DynamoDB tersebut kemudian dapat ditangkap sebagai item di aliran. Misalnya, Anda dapat menyetel pemicu sehingga ketika item dalam tabel diubah, rekaman baru segera muncul di aliran tabel tersebut.

Note

Anda dapat memiliki lebih dari dua fungsi Lambda yang berlangganan, tetapi ini dapat mengakibatkan pelambatan baca jika menggunakan lebih dari dua fungsi Lambda ke satu aliran DynamoDB.

Layanan [AWS Lambda](#) ini melakukan polling aliran untuk catatan baru empat kali per detik. Ketika rekaman aliran baru tersedia, fungsi Lambda Anda dipanggil secara sinkron. Anda dapat berlangganan hingga dua fungsi Lambda ke aliran DynamoDB yang sama.

Fungsi Lambda dapat mengirimkan pemberitahuan, memulai alur kerja, atau melakukan banyak tindakan lain yang Anda tentukan. Anda dapat menulis fungsi Lambda dengan mudah menyalin setiap catatan aliran ke penyimpanan persisten, seperti Amazon S3 File Gateway (Amazon S3), dan membuat jejak audit permanen dari aktivitas penulisan di tabel Anda. Selain itu, misalkan Anda mempunyai aplikasi game seluler yang menulis ke tabel GameScores. Setiap kali atribut TopScore dari tabel GameScores diperbarui, catatan stream yang sesuai ditulis ke stream tabel ini. Peristiwa ini kemudian dapat memicu fungsi Lambda yang memposting pesan ucapan selamat di jaringan media sosial. Fungsi ini juga dapat ditulis untuk mengabaikan catatan stream yang tidak memperbarui GameScores, atau yang tidak memodifikasi atribut TopScore.

Jika fungsi Anda mengembalikan kesalahan, Lambda akan mencoba ulang batch tersebut hingga berhasil diproses atau datanya habis masa berlakunya. Anda juga dapat mengonfigurasi Lambda untuk mencoba ulang dengan batch yang lebih kecil, membatasi jumlah percobaan ulang, membuang catatan setelah terlalu lama, dan opsi lainnya.

Sebagai praktik terbaik kinerja, fungsi Lambda harus berumur pendek. Untuk menghindari penundaan pemrosesan yang tidak perlu, logika yang rumit juga tidak boleh dijalankan. Khususnya untuk aliran kecepatan tinggi, lebih baik memicu alur kerja fungsi langkah pasca-pemrosesan asinkron daripada Lambdas yang berjalan lama secara sinkron.

Anda tidak dapat menggunakan pemicu Lambda yang sama di berbagai AWS akun. Baik tabel DynamoDB dan fungsi Lambda harus milik akun yang sama. AWS

Untuk informasi selengkapnya AWS Lambda, lihat [Panduan AWS Lambda Pengembang](#).

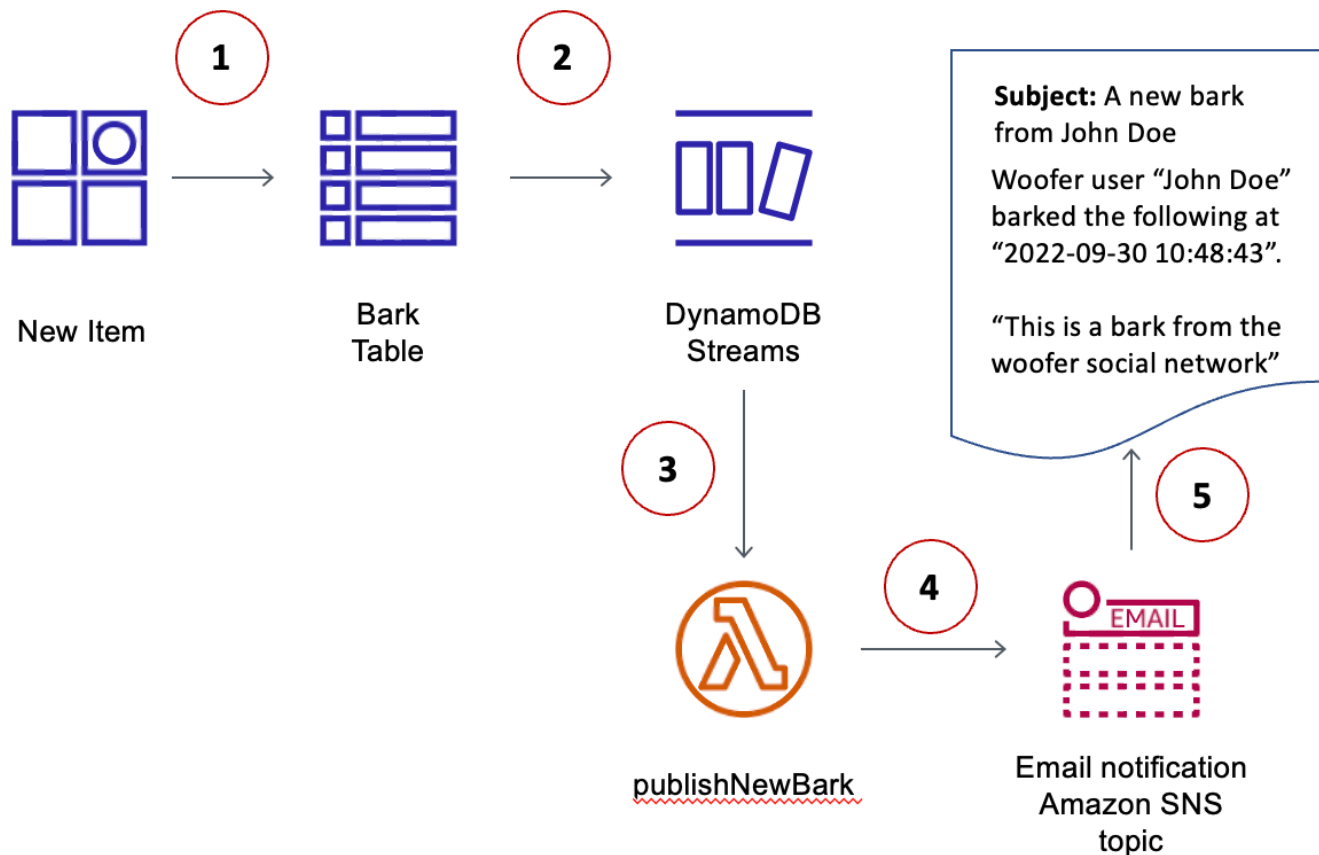
Tutorial #1: Menggunakan filter untuk memproses semua peristiwa dengan Amazon AWS Lambda DynamoDB dan menggunakan AWS CLI

Topik

- [Langkah 1: Buat tabel DynamoDB dengan aliran diaktifkan](#)
- [Langkah 2: Buat peran eksekusi Lambda](#)
- [Langkah 3: Buat Topik Amazon SNS](#)
- [Langkah 4: Buat dan uji fungsi Lambda](#)
- [Langkah 5: Buat dan uji pemicu](#)

Dalam tutorial ini, Anda akan membuat AWS Lambda pemicu untuk memproses aliran dari tabel DynamoDB.

Skenario untuk tutorial ini adalah Woofer, sebuah jejaring sosial sederhana. Pengguna Woofer berkomunikasi menggunakan bark (pesan teks singkat) yang dikirim ke pengguna Woofer lainnya. Diagram berikut menunjukkan komponen dan alur kerja untuk aplikasi ini.



1. Seorang pengguna menulis item ke tabel DynamoDB (BarkTable). Setiap item dalam tabel mewakili bark.
2. Sebuah catatan stream baru ditulis untuk mencerminkan bahwa item baru telah ditambahkan ke BarkTable.
3. Rekaman aliran baru memicu AWS Lambda fungsi (publishNewBark).
4. Jika catatan stream menunjukkan bahwa item baru ditambahkan ke BarkTable, fungsi Lambda membaca data dari catatan stream dan menerbitkan pesan ke topik di Amazon Simple Notification Service (Amazon SNS).
5. Pesan diterima oleh pelanggan untuk topik Amazon SNS. (Dalam tutorial ini, satu-satunya pelanggan adalah alamat email.)

Sebelum Anda Memulai

Tutorial ini menggunakan AWS Command Line Interface AWS CLI. Jika Anda belum melakukannya, ikuti petunjuk di [Panduan Pengguna AWS Command Line Interface](#) untuk menginstal dan mengkonfigurasi AWS CLI.

Langkah 1: Buat tabel DynamoDB dengan aliran diaktifkan

Pada langkah ini, Anda membuat tabel DynamoDB (`BarkTable`) untuk menyimpan semua bark dari pengguna Woofer. Kunci primer terdiri dari `Username` (kunci partisi) dan `Timestamp` (kunci urutan). Kedua atribut ini berjenis string.

`BarkTable` memiliki aliran yang diaktifkan. Kemudian dalam tutorial ini, Anda membuat pemicu dengan mengaitkan AWS Lambda fungsi dengan aliran.

1. Masukkan perintah berikut untuk membuat tabel.

```
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. Dalam output, cari `LatestStreamArn`.

```
...  
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/  
stream/timestamp  
...
```

Buat catatan tentang *region* dan *accountID*, karena Anda membutuhkannya untuk langkah-langkah lain dalam tutorial ini.

Langkah 2: Buat peran eksekusi Lambda

Pada langkah ini, Anda membuat peran AWS Identity and Access Management (IAM) (`WooferLambdaRole`) dan menetapkan izin untuk itu. Peran ini digunakan oleh fungsi Lambda yang Anda buat di [Langkah 4: Buat dan uji fungsi Lambda](#).

Anda juga membuat kebijakan untuk peran. Kebijakan ini berisi semua izin yang dibutuhkan fungsi Lambda pada saat waktu aktif.

1. Buat file yang diberi nama `trust-relationship.json` dengan konten berikut ini.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Masukkan perintah berikut untuk membuat `WoofierLambdaRole`.

```
aws iam create-role --role-name WoofierLambdaRole \
  --path "/service-role/" \
  --assume-role-policy-document file:///trust-relationship.json
```

3. Buat file yang diberi nama `role-policy.json` dengan konten berikut ini. (Ganti *region* dan *accountID* dengan AWS Wilayah dan ID akun Anda.)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:region:accountID:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeStream",

```

```

        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:ListStreams"
    ],
    "Resource": "arn:aws:dynamodb:region:accountID:table/BarkTable/stream/
*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "sns:Publish"
        ],
        "Resource": [
            "*"
        ]
    }
]
}

```

Kebijakan ini memiliki empat pernyataan yang memungkinkan `WoofierLambdaRole` untuk melakukan hal berikut:

- Menjalankan fungsi Lambda (`publishNewBark`). Anda membuat fungsi nanti dalam tutorial ini.
 - Akses CloudWatch Log Amazon. Fungsi Lambda menulis diagnostik ke CloudWatch Log saat runtime.
 - Membaca data dari stream DynamoDB untuk `BarkTable`.
 - Mempublikasikan pesan ke Amazon SNS.
4. Masukkan perintah berikut untuk melampirkan kebijakan ke `WoofierLambdaRole`.

```

aws iam put-role-policy --role-name WoofierLambdaRole \
  --policy-name WoofierLambdaRolePolicy \
  --policy-document file://role-policy.json

```

Langkah 3: Buat Topik Amazon SNS

Pada langkah ini, Anda membuat topik Amazon SNS (`woofierTopic`) dan mendaftarkan langganan alamat email untuk itu. Fungsi Lambda Anda menggunakan topik ini untuk mempublikasikan bark baru dari pengguna `Woofier`.

1. Masukkan perintah berikut untuk membuat topik Amazon SNS.

```
aws sns create-topic --name woofertopic
```

2. Masukkan perintah berikut untuk mendaftarkan langganan alamat email ke woofertopic. (Ganti *region* dan *accountID* dengan Wilayah AWS dan ID akun Anda, dan ganti *example@example.com* dengan alamat email yang valid.)

```
aws sns subscribe \  
  --topic-arn arn:aws:sns:region:accountID:woofertopic \  
  --protocol email \  
  --notification-endpoint example@example.com
```

3. Amazon SNS mengirimkan pesan konfirmasi ke alamat email Anda. Pilih tautan Konfirmasi langganan di pesan tersebut untuk menyelesaikan proses berlangganan.

Langkah 4: Buat dan uji fungsi Lambda

Pada langkah ini, Anda membuat AWS Lambda fungsi (`publishNewBark`) untuk memproses catatan aliran dari `BarkTable`.

Fungsi `publishNewBark` hanya memproses hanya peristiwa stream yang sesuai dengan item baru di `BarkTable`. Fungsi membaca data dari peristiwa tersebut, dan kemudian memanggil Amazon SNS untuk mempublikasikannya.

1. Buat file yang diberi nama `publishNewBark.js` dengan konten berikut ini. Ganti *region* dan *accountID* dengan AWS Wilayah dan ID akun Anda.

```
'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: 'Woofers user ' + who + ' barked the following at ' + when
+ ':\n\n ' + what,
  TopicArn: 'arn:aws:sns:region:accountID:woofersTopic'
};
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. Buat file zip untuk menampung `publishNewBark.js`. Jika Anda memiliki utilitas baris perintah zip, Anda dapat memasukkan perintah berikut untuk melakukan hal ini.

```
zip publishNewBark.zip publishNewBark.js
```

3. Ketika Anda membuat fungsi Lambda, Anda menentukan Amazon Resource Name (ARN) untuk `WoofersLambdaRole`, yang Anda buat di [Langkah 2: Buat peran eksekusi Lambda](#). Masukkan perintah berikut untuk mengambil ARN ini.

```
aws iam get-role --role-name WoofersLambdaRole
```

Dalam output, cari ARN untuk `WoofersLambdaRole`.

```
...
"Arn": "arn:aws:iam::region:role/service-role/WoofersLambdaRole"
...
```

Masukkan perintah berikut untuk membuat fungsi Lambda. Ganti `roleARN` dengan ARN untuk `WoofersLambdaRole`.

```
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. Sekarang uji `publishNewBark` untuk memverifikasi bahwa ia bekerja. Untuk melakukannya, Anda memberikan input yang menyerupai catatan nyata dari DynamoDB Streams.

Buat file yang diberi nama `payload.json` dengan konten berikut ini. Ganti *region* dan *accountID* dengan ID akun Wilayah AWS dan Anda.

```
{  
  "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 Woofers 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"
}
]
```

Masukkan perintah berikut untuk menguji fungsi `publishNewBark`.

```
aws lambda invoke --function-name publishNewBark --payload file://payload.json --
cli-binary-format raw-in-base64-out output.txt
```

Jika tes berhasil, Anda akan melihat output sebagai berikut.

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
```

Selain itu, file `output.txt` akan berisi teks berikut.

```
"Successfully processed 1 records."
```

Anda juga akan menerima pesan email baru dalam beberapa menit.

Note

AWS Lambda menulis informasi diagnostik ke Amazon CloudWatch Logs. Jika Anda mengalami kesalahan dengan fungsi Lambda Anda, Anda dapat menggunakan diagnostik ini untuk tujuan pemecahan masalah:

1. Buka CloudWatch konsol di <https://console.aws.amazon.com/cloudwatch/>.
2. Pilih Log di panel navigasi.
3. Pilih grup log berikut ini: `/aws/lambda/publishNewBark`

4. Pilih stream log terbaru untuk melihat output (dan kesalahan) dari fungsi.

Langkah 5: Buat dan uji pemicu

Dalam [Langkah 4: Buat dan uji fungsi Lambda](#), Anda menguji fungsi Lambda untuk memastikan bahwa itu berjalan dengan benar. Pada langkah ini, Anda membuat pemicu dengan mengaitkan fungsi Lambda (`publishNewBark`) dengan sumber peristiwa (aliran `BarkTable`).

1. Saat Anda membuat pemicu, Anda perlu menentukan ARN untuk stream `BarkTable`. Masukkan perintah berikut untuk mengambil ARN ini.

```
aws dynamodb describe-table --table-name BarkTable
```

Dalam output, cari `LatestStreamArn`.

```
...
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

2. Masukkan perintah berikut untuk membuat pemicu. Ganti *streamARN* dengan ARN stream yang sebenarnya.

```
aws lambda create-event-source-mapping \
  --region region \
  --function-name publishNewBark \
  --event-source streamARN \
  --batch-size 1 \
  --starting-position TRIM_HORIZON
```

3. Uji pemicu. Masukkan perintah berikut untuk menambahkan item ke `BarkTable`.

```
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"}
```

Anda akan menerima pesan email baru dalam beberapa menit.

4. Buka konsol DynamoDB dan tambahkan beberapa item lagi ke `BarkTable`. Anda harus menentukan nilai untuk atribut `Username` dan `Timestamp`. (Anda juga harus menentukan nilai untuk `Message`, meskipun tidak diperlukan.) Anda akan menerima pesan email baru untuk setiap item yang ditambahkan ke `BarkTable`.

Fungsi Lambda memproses hanya item baru yang Anda tambahkan ke `BarkTable`. Jika Anda memperbarui atau menghapus item dalam tabel, fungsi tidak melakukan apa pun.

Note

AWS Lambda menulis informasi diagnostik ke Amazon CloudWatch Logs. Jika Anda mengalami kesalahan dengan fungsi Lambda Anda, Anda dapat menggunakan diagnostik ini untuk tujuan pemecahan masalah.

1. Buka CloudWatch konsol di <https://console.aws.amazon.com/cloudwatch/>.
2. Pilih Log di panel navigasi.
3. Pilih grup log berikut ini: `/aws/lambda/publishNewBark`
4. Pilih stream log terbaru untuk melihat output (dan kesalahan) dari fungsi.

Tutorial #2: Menggunakan filter untuk memproses beberapa peristiwa dengan DynamoDB dan Lambda.

Topik

- [Menyatukan semuanya - AWS CloudFormation](#)
- [Menyatukan semuanya - CDK](#)

Dalam tutorial ini, Anda akan membuat AWS Lambda pemicu untuk memproses hanya beberapa peristiwa dalam aliran dari tabel DynamoDB.

Dengan [pemfilteran peristiwa Lambda](#), Anda dapat menggunakan ekspresi filter untuk mengontrol peristiwa mana yang dikirim Lambda ke fungsi Anda untuk diproses. Anda dapat mengonfigurasi hingga 5 filter berbeda per aliran DynamoDB. Jika Anda menggunakan jendela batching, Lambda menerapkan kriteria filter untuk setiap acara baru untuk melihat apakah itu harus disertakan dalam batch saat ini.

Filter diterapkan melalui struktur yang disebut `FilterCriteria`. 3 atribut utama `FilterCriteria` adalah `metadata properties`, `data properties`, dan `filter patterns`.

Berikut adalah contoh struktur dari acara DynamoDB Streams:

```
{
  "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": "7000000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-StreamsSampleDDBTable-LU0I6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

`metadata properties` adalah bidang objek peristiwa. Dalam kasus DynamoDB Streams, `metadata properties` adalah bidang seperti `dynamodb` atau `eventName`.

`data properties` adalah bidang badan peristiwa. Untuk memfilter `data properties`, pastikan untuk memasukkannya ke `FilterCriteria` dalam kunci yang tepat. Untuk sumber peristiwa DynamoDB, kunci data adalah `NewImage` atau `OldImage`.

Akhirnya, aturan filter akan menentukan ekspresi filter yang ingin Anda terapkan ke properti tertentu. Berikut ini adalah beberapa contohnya:

Operator perbandingan	Contoh	Sintaks aturan (Sebagian)
Null	Jenis Produk adalah null	<pre>{ "product_type": { "S": null } }</pre>
Kosong	Nama produk kosong	<pre>{ "product_name": { "S": [""] } }</pre>
Setara	Negara bagian sama dengan Florida	<pre>{ "state": { "S": ["FL"] } }</pre>
Dan	Negara bagian produk sama dengan Florida dan kategori produk Cokelat	<pre>{ "state": { "S": ["FL"] } , "category ": { "S": ["CHOCOLAT E"] } }</pre>
Atau	Negara bagian produk adalah Florida atau California	<pre>{ "state": { "S": ["FL","CA"] } }</pre>
Bukan	Negara bagian produk bukan Florida	<pre>{"state": {"S": [{"anything-but": ["FL"]}]]}}</pre>
Exists	Produk Rumahan ada	<pre>{"homemade": {"S": [{"exists": true}]]}}</pre>
Tidak ada	Produk Rumahan tidak ada	<pre>{"homemade": {"S": [{"exists": false}]]}}</pre>
Dimulai dengan	PK dimulai dengan PERUSAHAAN	<pre>{"PK": {"S": [{"prefix ": "COMPANY"}]]}}</pre>

Anda dapat menentukan hingga 5 pola penyaringan peristiwa untuk fungsi Lambda. Perhatikan bahwa masing-masing dari 5 peristiwa tersebut akan dievaluasi sebagai OR logis. Jadi jika

Anda mengkonfigurasi dua filter bernama `Filter_One` dan `Filter_Two`, fungsi Lambda akan mengeksekusi `Filter_One` OR `Filter_Two`.

Note

Di halaman [pemfilteran acara Lambda](#) ada beberapa opsi untuk memfilter dan membandingkan nilai numerik, namun dalam kasus peristiwa filter DynamoDB itu tidak berlaku karena angka di DynamoDB disimpan sebagai string. Misalnya `"quantity": { "N": "50" }`, kita tahu itu nomor karena properti `"N"`.

Menyatukan semuanya - AWS CloudFormation

Untuk menampilkan fungsionalitas pemfilteran acara dalam praktiknya, berikut adalah contoh CloudFormation template. Templat ini akan menghasilkan tabel DynamoDB Sederhana dengan PK Kunci Partisi dan SK Kunci Urutan dengan Amazon DynamoDB Streams diaktifkan. Ini akan membuat fungsi lambda dan peran Eksekusi Lambda sederhana yang memungkinkan penulisan log ke Amazon Cloudwatch, dan membaca peristiwa dari Amazon DynamoDB Stream. Ini juga akan menambahkan pemetaan sumber peristiwa antara DynamoDB Streams dan fungsi Lambda, sehingga fungsi tersebut dapat dijalankan setiap kali ada kejadian di Amazon DynamoDB Stream.

```
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
```

Setelah Anda menerapkan templat pembentukan cloud ini, Anda dapat memasukkan Item Amazon DynamoDB berikut:

```
{
  "PK": "COMPANY#1000",
  "SK": "PRODUCT#CHOCOLATE#DARK",
  "company_id": "1000",
  "type": "",
  "state": "FL",
  "stores": 5,
  "price": 15,
  "quantity": 50,
  "fabric": "Florida Chocolates"
}
```


Berkat fungsi lambda sederhana yang disertakan sebaris dalam template pembentukan cloud ini, Anda akan melihat peristiwa di grup CloudWatch log Amazon untuk fungsi lambda sebagai berikut:

```
{
  "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": "7000000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-StreamsSampleDDBTable-LU0I6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

Contoh Filter

- Hanya produk yang cocok dengan status tertentu

Contoh ini memodifikasi CloudFormation template untuk menyertakan filter untuk mencocokkan semua produk yang berasal dari Florida, dengan singkatan "FL".

```

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

```

Setelah Anda menerapkan kembali tumpukan, Anda dapat menambahkan item DynamoDB berikut ke tabel. Perhatikan bahwa itu tidak akan muncul di log fungsi Lambda, karena produk dalam contoh ini berasal dari California.

```

{
  "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": ""
}

```

- Hanya item yang dimulai dengan beberapa nilai di PK dan SK

Contoh ini memodifikasi CloudFormation template untuk menyertakan kondisi berikut:

```

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

```

Perhatikan kondisi AND mengharuskan kondisi berada di dalam pola, di mana PK dan SK Kunci berada dalam ekspresi yang sama dipisahkan oleh koma.

Baik mulai dengan beberapa nilai pada PK dan SK atau dari keadaan tertentu.

Contoh ini memodifikasi CloudFormation template untuk menyertakan kondisi berikut:

```

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

```

Perhatikan kondisi OR ditambahkan dengan memperkenalkan pola baru di bagian filter.

Menyatukan semuanya - CDK

Contoh templat pembentukan proyek CDK berikut berjalan melalui fungsionalitas penyaringan acara. Sebelum bekerja dengan proyek CDK ini, Anda perlu [menginstal prasyarat](#) termasuk [menjalankan skrip persiapan](#).

Buat proyek CDK

Pertama buat AWS CDK proyek baru, dengan memanggil `cdk init` dalam direktori kosong.

```

mkdir ddb_filters
cd ddb_filters

```

```
cdk init app --language python
```

Perintah `cdk init` menggunakan nama folder proyek untuk memberi nama berbagai elemen proyek, termasuk kelas, subfolder, dan file. Tanda hubung apa pun dalam nama folder diubah menjadi garis bawah. Nama tersebut harus mengikuti bentuk pengenalan Python. Misalnya, seharusnya tidak dimulai dengan angka atau berisi spasi.

Untuk bekerja dengan proyek baru, aktifkan lingkungan virtualnya. Ini memungkinkan dependensi proyek diinstal secara lokal di folder proyek, bukan secara global.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Note

Anda mungkin mengenali ini sebagai perintah Mac/Linux untuk mengaktifkan lingkungan virtual. Templat Python menyertakan file batch, `source.bat`, yang memungkinkan perintah yang sama untuk digunakan pada Windows. Perintah Windows tradisional `.venv\Scripts\activate.bat` juga berfungsi. Jika Anda menginisialisasi AWS CDK proyek Anda menggunakan AWS CDK Toolkit v1.70.0 atau yang lebih lama, lingkungan virtual Anda ada di direktori, bukan `.env .venv`.

Infrastruktur Dasar

Buka file `./ddb_filters/ddb_filters_stack.py` dengan editor teks pilihan Anda. File ini dibuat secara otomatis saat Anda membuat AWS CDK proyek.

Selanjutnya, tambahkan fungsi `_create_ddb_table` dan `_set_ddb_trigger_function`. Fungsi-fungsi ini akan membuat tabel DynamoDB dengan kunci partisi PK dan mengurutkan kunci SK dalam mode penyediaan mode sesuai permintaan, dengan Amazon DynamoDB Streams diaktifkan secara default untuk menampilkan gambar Baru dan Lama.

Fungsi Lambda akan disimpan di folder `lambda` di bagian file `app.py`. File ini akan dibuat nanti. Ini akan mencakup variabel lingkungan `APP_TABLE_NAME`, yang akan menjadi nama Tabel Amazon DynamoDB yang dibuat oleh tumpukan ini. Dalam fungsi yang sama kami akan memberikan izin baca aliran ke fungsi Lambda. Akhirnya, hal tersebut akan berlangganan DynamoDB Streams sebagai sumber acara untuk fungsi lambda.

Di akhir file dalam metode `__init__`, Anda akan memanggil konstruksi masing-masing untuk menginisialisasi mereka dalam tumpukan. Untuk proyek yang lebih besar yang memerlukan komponen dan layanan tambahan, mungkin yang terbaik adalah mendefinisikan konstruksi ini di luar tumpukan dasar.

```
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)
```

Sekarang kita akan membuat fungsi lambda yang sangat sederhana yang akan mencetak log ke Amazon CloudWatch. Untuk melakukannya, buat folder baru bernama `lambda`.

```
mkdir lambda
touch app.py
```

Menggunakan editor teks favorit Anda, tambahkan konten berikut ke file `app.py`:

```
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)
```

Memastikan Anda berada di folder `/ddb_filters/`, ketikkan perintah berikut untuk membuat aplikasi sampel:

```
cdk deploy
```

Pada titik tertentu Anda akan diminta untuk mengonfirmasi apakah Anda ingin menerapkan solusi tersebut. Terima perubahan dengan mengetik `Y`.

```
#####
# + # ${LambdaHandler/ServiceRole} # arn:${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
```

Setelah perubahan diterapkan, buka AWS konsol Anda dan tambahkan satu item ke tabel Anda.

```
{
  "PK": "COMPANY#1000",
  "SK": "PRODUCT#CHOCOLATE#DARK",
  "company_id": "1000",
  "type": "",
  "state": "FL",
  "stores": 5,
  "price": 15,
  "quantity": 50,
  "fabric": "Florida Chocolates"
}
```

CloudWatch Log sekarang harus berisi semua informasi dari entri ini.

Contoh Filter

- Hanya produk yang cocok dengan status tertentu

Buka file `ddb_filters/ddb_filters/ddb_filters_stack.py`, dan modifikasi untuk menyertakan filter yang cocok dengan semua produk yang setara dengan “FL”. Ini dapat direvisi tepat di bawah `event_subscription` di baris 45.

```
event_subscription.add_property_override(  
    property_path="FilterCriteria",  
    value={  
        "Filters": [  
            {  
                "Pattern": json.dumps(  
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}  
                )  
            },  
        ]  
    },  
)
```

- Hanya item yang dimulai dengan beberapa nilai di PK dan SK

Ubah skrip python untuk menyertakan kondisi berikut:

```
event_subscription.add_property_override(  
    property_path="FilterCriteria",  
    value={  
        "Filters": [  
            {  
                "Pattern": json.dumps(  
                    {  
                        {  
                            "dynamodb": {  
                                "Keys": {  
                                    "PK": {"S": [{"prefix": "COMPANY"}]},  
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},  
                                }  
                            }  
                        }  
                    )  
                },  
            ],  
        },  
)
```



```
    ],
  },
```

- Baik mulai dengan beberapa nilai pada PK dan SK atau dari keadaan tertentu.

Ubah skrip python untuk menyertakan kondisi berikut:

```
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"]}}}
                )
            },
        ]
    },
)
```

Perhatikan bahwa kondisi OR ditambahkan dengan menambahkan lebih banyak elemen ke array Filter.

Pembersihan

Temukan tumpukan filter di dasar direktori kerja Anda, dan jalankan `cdk destroy`. Anda akan diminta untuk mengonfirmasi penghapusan sumber daya:

```
cdk destroy
Are you sure you want to delete: DdbFiltersStack (y/n)? y
```

Praktik terbaik dengan Lambda

AWS Lambda Fungsi berjalan di dalam wadah —lingkungan eksekusi yang diisolasi dari fungsi lain. Ketika Anda menjalankan fungsi untuk pertama kalinya, AWS Lambda membuat wadah baru dan mulai mengeksekusi kode fungsi.

Fungsi Lambda memiliki handler yang dijalankan sekali per permohonan. Handler berisi logika bisnis utama untuk fungsi. Misalnya, fungsi Lambda yang ditampilkan dalam [Langkah 4: Buat dan uji fungsi Lambda](#) memiliki handler yang dapat memproses catatan dalam DynamoDB stream.

Anda juga dapat memberikan kode inisialisasi yang hanya berjalan satu kali—setelah penampung dibuat, tetapi sebelumnya AWS Lambda menjalankan handler untuk pertama kalinya. Fungsi Lambda yang ditampilkan di [Langkah 4: Buat dan uji fungsi Lambda](#) memiliki kode inisialisasi yang mengimpor SDK untuk JavaScript di Node.js, dan membuat klien untuk Amazon SNS. Objek-objek ini hanya boleh didefinisikan sekali, di luar handler.

Setelah fungsi berjalan, AWS Lambda mungkin memilih untuk menggunakan kembali wadah untuk pemanggilan fungsi berikutnya. Dalam kasus ini, handler fungsi Anda mungkin dapat menggunakan kembali sumber daya yang Anda tetapkan dalam kode inisialisasi Anda. (Anda tidak dapat mengontrol berapa lama AWS Lambda akan mempertahankan kontainer, atau apakah kontainer akan digunakan kembali.)

Untuk pemicu DynamoDB AWS Lambda menggunakan, kami merekomendasikan hal berikut:

- AWS klien layanan harus dipakai dalam kode inisialisasi, bukan di handler. Ini memungkinkan AWS Lambda untuk menggunakan kembali koneksi yang ada, selama masa pakai kontainer.
- Secara umum, Anda tidak perlu secara eksplisit mengelola koneksi atau menerapkan penyatuan koneksi karena AWS Lambda mengelola ini untuk Anda.

Konsumen Lambda untuk aliran DynamoDB tidak menjamin persis sekali pengiriman dan dapat menyebabkan duplikat sesekali. Pastikan kode fungsi Lambda Anda idempoten untuk mencegah timbulnya masalah tak terduga karena pemrosesan duplikat.

Untuk informasi selengkapnya, lihat [Praktik terbaik untuk bekerja dengan AWS Lambda fungsi](#) di Panduan AWS Lambda Pengembang.

Menggunakan cadangan Sesuai Permintaan dan DynamoDB Permintaan

Anda dapat menggunakan kemampuan sesuai permintaan DynamoDB untuk membuat pencadangan penuh dari tabel Anda untuk retensi jangka panjang, dan pengarsipan untuk kebutuhan kepatuhan terhadap peraturan. Anda dapat mencadangkan dan memulihkan data tabel Anda kapan saja dengan satu klik di AWS Management Console atau satu panggilan API. Tindakan Backup dan pemulihan dijalankan dengan dampak tidak pada performa atau ketersediaan tabel.

Video berikut akan memberi Anda tampilan pengantar konsep cadangan dan pemulihan.

[Cadangkan dan pulihkan](#)

Tersedia dua opsi untuk membuat dan mengelola cadangan sesuai permintaan DynamoDB:

- AWS Layanan Backup
- DynamoDB

Dengan AWS Backup, Anda dapat mengonfigurasi kebijakan cadangan dan memantau aktivitas untuk AWS sumber daya dan beban kerja lokal di satu tempat. Menggunakan DynamoDB dengan AWS Backup, Anda dapat menyalin cadangan sesuai permintaan Anda ke AWS Region dan Wilayah, tambahkan tag alokasi biaya ke cadangan sesuai permintaan, dan transisi cadangan sesuai permintaan ke penyimpanan dingin dengan biaya lebih rendah. Untuk menggunakan fitur-fitur canggih ini, Anda harus [memilih](#) kepada AWS Backup. Pilihan keikutsertaan berlaku untuk akun tertentu dan AWS Wilayah, jadi Anda mungkin harus ikut serta ke beberapa Wilayah menggunakan akun yang sama. Untuk informasi selengkapnya, lihat [Panduan Developer AWS Backup](#).

Pencadangan sesuai permintaan dan proses pemulihan diskalakan tanpa mengurangi performa atau ketersediaan aplikasi Anda. Ini menggunakan teknologi terdistribusi baru dan unik yang memungkinkan Anda menyelesaikan pencadangan dalam hitungan detik terlepas dari ukuran tabel. Anda dapat membuat cadangan yang konsisten dalam hitungan detik di ribuan partisi tanpa khawatir tentang jadwal atau proses pencadangan yang berjalan lama. Semua cadangan sesuai permintaan dibuat katalog, dapat ditemukan, dan dipertahankan sampai secara eksplisit dihapus.

Selain itu, operasi pencadangan dan pemulihan sesuai permintaan tidak mempengaruhi performa atau latensi API. Pencadangan dipertahankan terlepas dari penghapusan tabel. Untuk informasi selengkapnya, lihat [Menggunakan pencadangan dan pemulihan DynamoDB](#).

Pencadangan sesuai permintaan DynamoDB tersedia tanpa biaya tambahan di luar harga normal yang terkait dengan ukuran penyimpanan cadangan. Pencadangan sesuai permintaan DynamoDB tidak dapat disalin ke akun atau Wilayah yang berbeda. Untuk membuat salinan cadangan di AWS akun dan Wilayah dan untuk fitur-fitur lanjutan lainnya, Anda harus menggunakan AWS Backup. Jika Anda menggunakan AWS Backup fitur Anda akan ditagih untuk mereka dengan AWS Backup. Untuk informasi selengkapnya tentang penetapan harga dan ketersediaan Wilayah AWS, lihat [Harga Amazon DynamoDB](#).

Topik

- [Menggunakan AWS Backup dengan DynamoDB](#)
- [Menggunakan pencadangan dan pemulihan DynamoDB](#)

Menggunakan AWS Backup dengan DynamoDB

Amazon DynamoDB dapat membantu Anda memenuhi kepatuhan peraturan dan persyaratan kelangsungan bisnis melalui fitur cadangan yang disempurnakan AWS Backup. AWS Backup adalah layanan perlindungan data terkelola sepenuhnya yang memudahkan untuk memusatkan dan mengotomatisasi backup AWS layanan, di awan, dan di tempat. Dengan menggunakan layanan ini, Anda dapat mengonfigurasi kebijakan cadangan dan memantau aktivitas untuk AWS sumber daya di satu tempat. Untuk menggunakan AWS Backup, Anda harus afirmatif [Keikutsertaan](#). Pilihan opt-in berlaku untuk akun tertentu dan AWS Wilayah, jadi Anda mungkin harus memilih untuk beberapa Wilayah menggunakan akun yang sama. Untuk informasi selengkapnya, lihat [AWS Backup Developer](#).

Amazon DynamoDB terintegrasi secara native dengan AWS Backup. Anda dapat menggunakan AWS Backup untuk menjadwalkan, menyalin, menandai, dan siklus hidup backup on-demand DynamoDB Anda secara otomatis. Anda dapat terus melihat dan memulihkan cadangan ini dari konsol DynamoDB. Anda dapat menggunakan konsol DynamoDB, API, dan AWS Antarmuka Baris Perintah (AWS CLI) untuk mengaktifkan backup otomatis untuk tabel DynamoDB Anda.

Note

Setiap backup yang dilakukan melalui DynamoDB akan tetap tidak berubah. Anda masih dapat membuat cadangan melalui alur kerja DynamoDB saat ini.

Fitur cadangan yang disempurnakan tersedia melalui AWS Backup termasuk:

Backup terjadwal- Anda dapat mengatur backup terjadwal secara teratur dari tabel DynamoDB Anda menggunakan paket cadangan.

Salinan lintas akun dan lintas Wilayah- Anda dapat secara otomatis menyalin cadangan Anda ke lemari besi cadangan lain di yang berbedaAWSWilayah atau akun, yang memungkinkan Anda mendukung persyaratan perlindungan data Anda.

Penyimpanan dingin- Anda dapat mengkonfigurasi backup Anda untuk menerapkan aturan siklus hidup untuk menghapus atau transisi backup ke penyimpanan yang lebih dingin. Ini dapat membantu Anda mengoptimalkan biaya cadangan Anda.

Tag- Anda dapat secara otomatis menandai cadangan Anda untuk tujuan penagihan dan alokasi biaya.

Enkripsi— DynamoDB on-demand backup dikelola melaluiAWS Backupsekarang disimpan diAWS Backuplemari besi. Hal ini memungkinkan Anda untuk mengenkripsi dan mengamankan backup Anda dengan menggunakanAWS KMS keyyang independen dari kunci enkripsi tabel DynamoDB Anda.

Backup audit— Anda dapat menggunakanAWS BackupAudit Manager untuk mengaudit kepatuhan AndaAWS Backupkebijakan dan untuk menemukan aktivitas cadangan dan sumber daya yang belum sesuai dengan kontrol yang Anda tetapkan. Anda juga dapat menggunakannya untuk secara otomatis menghasilkan jejak audit laporan harian dan sesuai permintaan untuk tujuan tata kelola cadangan Anda.

Backup aman menggunakan model WORM— Anda dapat menggunakanAWS BackupVault Lock untuk mengaktifkan write-once-read-many Pengaturan (WORM) untuk backup Anda. DenganAWS BackupVault Lock, Anda dapat menambahkan lapisan pertahanan tambahan yang melindungi cadangan dari operasi penghapusan yang tidak disengaja atau berbahaya, perubahan pada periode penyimpanan cadangan, dan pembaruan ke pengaturan siklus hidup. Untuk mempelajari lebih lanjut, lihat[AWS BackupKunci Vault](#).

Fitur backup yang disempurnakan tersedia di semuaAWSWilayah. Untuk mempelajari lebih lanjut tentang fitur-fitur ini, lihat[AWS BackupPanduan Pengembang](#).

Topik

- [Mencadangkan dan memulihkan tabel DynamoDB denganAWS Backup: Cara kerjanya](#)
- [Membuat backup tabel DynamoDB denganAWS Backup](#)
- [Menyalin cadangan tabel DynamoDB denganAWS Backup](#)

- [Memulihkan cadangan tabel DynamoDB dari AWS Backup](#)
- [Menghapus cadangan tabel DynamoDB dengan AWS Backup](#)
- [Catatan penggunaan](#)

Mencadangkan dan memulihkan tabel DynamoDB dengan AWS Backup: Cara kerjanya

Anda dapat menggunakan fitur cadangan sesuai permintaan untuk membuat pencadangan penuh tabel Amazon DynamoDB Anda. Bagian ini memberikan gambaran umum tentang apa yang terjadi selama proses pencadangan dan pemulihan.

Backup

Saat Anda membuat cadangan sesuai permintaan dengan AWS Backup, penanda waktu permintaan dikatalogkan. Cadangan dibuat dengan tidak tersinkronisasi dengan menerapkan semua perubahan sampai waktu permintaan untuk snapshot terakhir tabel lengkap.

Setiap kali Anda membuat cadangan sesuai permintaan, seluruh data tabel dicadangkan. Tidak ada batasan jumlah pencadangan sesuai permintaan yang dapat diambil.

Note

Tidak seperti DynamoDB Backup, backup dibuat dengan AWS Backup tidak seketika.

Saat pencadangan sedang berlangsung, Anda tidak dapat melakukan hal berikut:

- Menjeda atau membatalkan operasi pencadangan.
- Menghapus tabel sumber cadangan.
- Menonaktifkan pencadangan pada tabel jika cadangan untuk tabel sedang berlangsung.

AWS Backup menyediakan jadwal pencadangan otomatis, manajemen retensi, dan manajemen siklus hidup. Ini menghilangkan kebutuhan akan skrip khusus dan proses manual. AWS Backup menjalankan backup dan menghapusnya saat kedaluwarsa. Untuk informasi selengkapnya, lihat [Panduan Developer AWS Backup](#).

Jika Anda menggunakan konsol, pencadangan apa pun yang dibuat menggunakan AWS Backup tercantum pada tab Cadangan dengan Jenis cadangan diatur ke AWS_BACKUP.

 Note


Anda tidak dapat menghapus pencadangan yang ditandai dengan Jenis cadangan dari `AWS_BACKUP` menggunakan konsol DynamoDB. Untuk mengelola cadangan ini, gunakan konsol AWS Backup.

Untuk mempelajari cara melakukan pencadangan, lihat [Membuat backup tabel DynamoDB](#).

Mengembalikan

Anda memulihkan tabel tanpa mengkonsumsi throughput apa pun yang disediakan pada tabel. Anda dapat melakukan pemulihan tabel penuh dari cadangan DynamoDB Anda, atau Anda dapat mengkonfigurasi pengaturan tabel tujuan. Ketika Anda melakukan pemulihan, Anda dapat mengubah pengaturan tabel berikut:

- Indeks sekunder global (GSI)
- Indeks sekunder lokal (LSIs)
- Mode penagihan
- Kapasitas baca dan tulis yang disediakan
- Pengaturan Enkripsi

 Important

Ketika Anda melakukan pemulihan tabel penuh, tabel tujuan diatur dengan unit kapasitas baca yang sama dan unit kapasitas tulis yang dimiliki tabel sumber ketika cadangan diminta. Proses pemulihan juga memulihkan indeks sekunder lokal dan indeks sekunder global.


Anda dapat menyalin cadangan data tabel DynamoDB Anda ke yang berbeda AWS Wilayah dan kemudian mengembalikannya di Wilayah baru itu. Anda dapat menyalin dan kemudian memulihkan cadangan antara AWS Kawasan komersial, AWS Wilayah China, dan AWS GovCloud (AS) Wilayah. Anda hanya membayar data yang Anda salin dari sumber Wilayah dan data yang Anda pulihkan ke tabel baru di Wilayah tujuan.

AWS Backup akan mengembalikan tabel dengan semua indeks asli.

Anda harus secara manual mengatur berikut ini pada tabel yang dipulihkan:

- Kebijakan Auto Scaling
- Kebijakan (IAM) AWS Identity and Access Management
- AmazonCloudWatchmetrik dan alarm
- Tanda
- Pengaturan stream
- Pengaturan Time to Live (TTL)
- Pengaturan perlindungan penghapusan
- Pengaturan Point in Time Recovery (PITR)

Anda hanya dapat memulihkan seluruh data tabel ke tabel baru dari pencadangan. Anda dapat menulis ke tabel yang dipulihkan hanya setelah menjadi aktif.

 Note

AWS Backupmengembalikan tidak merusak. Anda tidak dapat menimpa tabel yang ada selama operasi pemulihan.

Metrik layanan menunjukkan bahwa 95 persen dari pemulihan tabel pelanggan selesai dalam waktu kurang dari satu jam. Namun, waktu pemulihan secara langsung berhubungan dengan konfigurasi tabel Anda (seperti ukuran tabel Anda dan jumlah partisi yang mendasari) dan variabel terkait lainnya. Praktik terbaik saat merencanakan pemulihan bencana adalah secara teratur mendokumentasikan waktu penyelesaian pemulihan rata-rata dan menetapkan bagaimana waktu ini mempengaruhi Tujuan Waktu Pemulihan Anda secara keseluruhan.

Untuk mempelajari cara melakukan pemulihan, lihat [Memulihkan tabel DynamoDB dari cadangan](#).

Anda dapat menggunakan kebijakan IAM untuk kontrol akses. Untuk informasi selengkapnya, lihat [Menggunakan IAM dengan Pencadangan dan Pemulihan DynamoDB](#).

Semua konsol cadangan dan pemulihan dan tindakan API ditangkap dan direkam di AWS CloudTrail untuk pencatatan, pemantauan berkelanjutan, dan audit.

Membuat backup tabel DynamoDB denganAWS Backup

Bagian ini menjelaskan cara mengaktifkanAWS Backup untuk membuat cadangan sesuai permintaan dan terjadwal dari tabel DynamoDB Anda.

Topik

- [MenghidupkanAWS Backup fitur](#)
- [Pencadangan](#)
- [Pencadangan](#)

MenghidupkanAWS Backup fitur

Anda harus mengaktifkanAWS Backup untuk menggunakannya dengan DynamoDB.

Untuk menghidupkanAWS Backup, ikuti langkah-langkah berikut:

1. Masuk keAWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Di jendela Pengaturan Backup, pilih Aktifkan.
4. Layar konfirmasi akan muncul. Pilih Aktifkan fitur.

AWS Backupfitur sekarang tersedia untuk tabel DynamoDB Anda.

Jika Anda memilih untuk menonaktifkanAWS Backup fitur setelah dinyalakan, ikuti langkah-langkah berikut:

1. Masuk keAWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Di jendela Pengaturan Backup, pilih Matikan.
4. Layar konfirmasi akan muncul. Pilih Matikan fitur.

Jika Anda tidak dapat mengaktifkan atau menonaktifkanAWS Backup fitur,AWS admin Anda mungkin perlu melakukan tindakan tersebut.

Pencadangan

Untuk membuat cadangan sesuai permintaan atas tabel DynamoDB, ikuti langkah-langkah berikut:

1. Masuk keAWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.

2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Pilih Buat cadangan.
4. Dari menu tarik-turun yang muncul, pilih Buat cadangan sesuai permintaan.
5. Untuk membuat cadangan yang dikelola AWS Backup dengan penyimpanan hangat dan fitur dasar lainnya, pilih Pengaturan Default. Untuk membuat cadangan yang dapat dialihkan ke penyimpanan dingin, atau untuk membuat cadangan dengan fitur DynamoDB, bukan AWS Backup, pilih Sesuaikan pengaturan.

Jika Anda ingin membuat cadangan ini dengan fitur DynamoDB sebelumnya, pilih Sesuaikan pengaturan dan kemudian pilih Backup dengan DynamoDB.

6. Setelah Anda menyelesaikan pengaturan, pilih Buat cadangan.

Pencadangan

Untuk menjadwalkan cadangan, ikuti langkah-langkah ini.

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Dari menu tarik-turun yang muncul, pilih Jadwalkan pencadangan dengan AWS Backup.
4. Anda menerima cadangan AWS Backup terkait cadangan.

Menyalin cadangan tabel DynamoDB dengan AWS Backup

Anda dapat membuat salinan cadangan saat ini. Anda dapat menyalin backup ke beberapa AWS Akun atau AWS Daerah sesuai permintaan atau secara otomatis sebagai bagian dari rencana cadangan terjadwal. Anda juga dapat mengotomatiskan urutan salinan lintas akun dan lintas wilayah untuk Klien Enkripsi Amazon DynamoDB.

Replikasi lintas wilayah sangat berharga jika Anda memiliki kelangsungan bisnis atau persyaratan kepatuhan untuk menyimpan cadangan jarak minimum dari data produksi Anda.

Cadangan lintas akun berguna untuk menyalin cadangan Anda dengan aman ke satu atau lebih AWS Akun di organisasi Anda untuk alasan operasional atau keamanan. Jika cadangan asli Anda secara tidak sengaja dihapus, Anda dapat menyalin cadangan dari akun tujuannya ke akun sumbernya, dan kemudian memulai pemulihan. Sebelum Anda dapat melakukan ini, Anda harus memiliki dua akun milik Organizations yang sama di layanan Organisasi.

Salinan mewarisi konfigurasi cadangan sumber kecuali Anda menentukan sebaliknya, dengan satu pengecualian: jika Anda menentukan bahwa salinan baru Anda “Tidak Pernah” berakhir. Dengan pengaturan ini, salinan baru masih mewarisi tanggal kedaluwarsa sumbernya. Jika Anda ingin salinan cadangan baru Anda permanen, atur cadangan sumber Anda agar tidak pernah kedaluwarsa, atau tentukan salinan baru Anda untuk kedaluwarsa 100 tahun setelah pembuatannya.

Note

Jika Anda menyalin ke akun lain, Anda harus terlebih dahulu memiliki izin dari akun tersebut.

Untuk menyalin cadangan, lakukan hal berikut:

1. Masuk keAWSManagement Console dan buka konsol DynamoDB di<https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Pilih kotak centang di samping cadangan yang ingin Anda salin.
 - Jika cadangan yang ingin Anda salin berwarna abu-abu, Anda harus mengaktifkannya[fitur-fitur lanjutan denganAWS Backup](#). Kemudian buat cadangan baru. Anda sekarang dapat menyalin cadangan baru ini ke Wilayah dan akun lain, dan menyalin cadangan baru lainnya ke depan.
4. Pilih Salin.
5. Jika Anda ingin menyalin cadangan ke akun lain atau Wilayah, pilih kotak centang di sampingSalin titik pemulihan ke tujuan lain. Kemudian pilih apakah Anda akan menyalin ke Wilayah lain di akun Anda, atau ke akun lain di Wilayah yang berbeda.

Note

Untuk memulihkan cadangan ke Wilayah atau akun lain, Anda harus terlebih dahulu menyalin cadangan ke Wilayah atau akun tersebut.

6. Pilih lemari besi yang diinginkan file akan disalin ke dalam. Anda juga dapat membuat brankas cadangan baru jika diinginkan.
7. Pilih Salin cadangan.

Memulihkan cadangan tabel DynamoDB dari AWS Backup

Bagian ini menjelaskan cara mengembalikan cadangan tabel DynamoDB dari. AWS Backup

Topik

- [Memulihkan tabel DynamoDB dari AWS Backup](#)
- [Memulihkan tabel DynamoDB ke Wilayah atau akun lain](#)

Memulihkan tabel DynamoDB dari AWS Backup

Untuk memulihkan tabel DynamoDB Anda, ikuti langkah-langkah AWS Backup berikut:

1. [Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/](https://console.aws.amazon.com/dynamodb/)
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tab Backup.
4. Centang kotak centang di samping cadangan sebelumnya yang ingin Anda pulihkan.
5. Pilih Pulihkan. Anda akan dibawa ke tabel Pulihkan dari layar cadangan.
6. Masukkan nama untuk tabel yang baru dipulihkan, enkripsi yang dimiliki tabel baru ini, kunci yang ingin Anda gunakan untuk mengenkripsi pemulihan, dan opsi lainnya.
7. Setelah selesai, pilih Pulihkan.

Memulihkan tabel DynamoDB ke Wilayah atau akun lain

Untuk mengembalikan tabel DynamoDB ke Wilayah atau akun lain, Anda harus terlebih dahulu menyalin cadangan ke Wilayah atau akun baru tersebut. Untuk menyalin ke akun lain, akun tersebut harus memberi Anda izin terlebih dahulu. Setelah Anda menyalin cadangan DynamoDB Anda ke Wilayah atau akun baru, cadangan tersebut dapat dipulihkan dengan proses di bagian sebelumnya.

Menghapus cadangan tabel DynamoDB dengan AWS Backup

Bagian ini menjelaskan cara menghapus cadangan tabel DynamoDB dengan AWS Backup

Cadangan DynamoDB yang dibuat melalui fitur AWS Backup disimpan di brankas Backup. AWS

Untuk menghapus cadangan semacam ini, lakukan hal berikut:

1. [Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/](https://console.aws.amazon.com/dynamodb/).
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.

3. Pada layar berikut, pilih Continue to AWS Backup.

Anda akan dibawa ke konsol AWS Backup. [Untuk mempelajari lebih lanjut tentang cara menghapus cadangan di konsol AWS Backup, lihat Menghapus cadangan](#)

Catatan penggunaan

Bagian ini menjelaskan perbedaan teknis antara cadangan sesuai permintaan yang dikelola oleh AWS Backup dan DynamoDB.

AWS Backup memiliki beberapa alur kerja dan perilaku yang berbeda dari DynamoDB. Ini termasuk:

Enkripsi - Cadangan yang dibuat dengan AWS Backup paket disimpan dalam lemari besi terenkripsi dengan kunci yang dikelola oleh AWS Backup layanan. Lemari besi memiliki kebijakan kontrol akses untuk keamanan tambahan.

Backup ARN - File cadangan yang dibuat oleh sekarang AWS Backup akan memiliki AWS Backup ARN, yang dapat memengaruhi model izin pengguna. Nama sumber daya Backup (ARN) akan berubah dari `arn:aws:dynamodb:kearn:aws:backup`.

Menghapus cadangan - Cadangan yang dibuat hanya AWS Backup dapat dihapus dari AWS Backup lemari besi. Anda tidak akan dapat menghapus AWS Backup file dari konsol DynamoDB.

Proses Backup - Tidak seperti backup DynamoDB, backup AWS Backup dibuat dengan tidak instan.

Penagihan - Backup tabel DynamoDB dengan AWS Backup fitur ditagih dari AWS Backup.

Peran IAM - Jika Anda mengelola akses melalui peran IAM, Anda juga perlu mengonfigurasi peran IAM baru dengan izin baru ini:

```
"dynamodb:StartAwsBackupJob",  
"dynamodb:RestoreTableFromAwsBackup"
```

`dynamodb:StartAwsBackupJob` diperlukan untuk cadangan yang sukses dengan AWS Backup fitur, dan `dynamodb:RestoreTableFromAwsBackup` diperlukan untuk memulihkan dari cadangan yang dibuat dengan AWS Backup fitur.

Untuk melihat izin ini dalam kebijakan IAM lengkap, lihat Contoh 8 di [Menggunakan IAM](#).

Menggunakan pencadangan dan pemulihan DynamoDB

Amazon DynamoDB mendukung pencadangan sesuai permintaan yang berdiri sendiri dan memulihkan fitur. Fitur-fitur tersebut tersedia untuk Anda terlepas dari apakah Anda menggunakan AWS Backup.

Anda dapat menggunakan kemampuan sesuai permintaan DynamoDB untuk membuat pencadangan penuh dari tabel Anda untuk retensi dan pengarsipan jangka panjang untuk kebutuhan kepatuhan terhadap peraturan. Anda dapat mencadangkan dan memulihkan data tabel Anda kapan saja dengan satu klik di AWS Management Console atau dengan satu panggilan API. Tindakan pencadangan dan pemulihan dijalankan dengan dampak nol pada performa atau ketersediaan tabel.

Anda dapat membuat pencadangan tabel menggunakan konsol, AWS Antarmuka Baris Perintah (AWS CLI), atau API DynamoDB. Untuk informasi selengkapnya, lihat [Membuat backup tabel DynamoDB](#).

Untuk informasi tentang pemulihan tabel dari pencadangan, lihat [Memulihkan tabel DynamoDB dari cadangan](#).

Mencadangkan dan memulihkan tabel DynamoDB dengan DynamoDB: Cara kerjanya

Anda dapat menggunakan fitur pencadangan on-demand DynamoDB untuk membuat cadangan lengkap tabel Amazon DynamoDB Anda. Fitur ini tersedia secara independen dari AWS cadangan. Bagian ini memberikan gambaran umum tentang apa yang terjadi selama proses pencadangan dan pemulihan DynamoDB.

Backup

Saat Anda membuat cadangan sesuai permintaan dengan DynamoDB, penanda waktu permintaan akan dikatalogkan. Cadangan dibuat dengan tidak tersinkronisasi dengan menerapkan semua perubahan sampai waktu permintaan untuk snapshot terakhir tabel lengkap. Permintaan cadangan DynamoDB diproses secara instan dan tersedia untuk dipulihkan dalam hitungan menit.

Note

Setiap kali Anda membuat cadangan sesuai permintaan, seluruh data tabel dicadangkan. Tidak ada batasan jumlah pencadangan sesuai permintaan yang dapat diambil.

Semua pencadangan di DynamoDB bekerja tanpa mengkonsumsi throughput apa pun yang disediakan pada tabel.

Pencadangan DynamoDB tidak menjamin konsistensi kausal di item; namun, ketidaksimetrisan antara pembaruan dalam cadangan biasanya jauh lebih sedikit dari satu detik.

Saat pencadangan sedang berlangsung, Anda tidak dapat melakukan hal berikut:

- Menjeda atau membatalkan operasi pencadangan.
- Menghapus tabel sumber cadangan.
- Menonaktifkan pencadangan pada tabel jika cadangan untuk tabel sedang berlangsung.

Jika Anda tidak ingin membuat penjadwalan skrip dan pekerjaan pembersihan, Anda dapat menggunakan AWS Backup untuk membuat rencana pencadangan dengan kebijakan jadwal dan retensi untuk tabel DynamoDB Anda. AWS Backup menjalankan pencadangan dan menghapusnya ketika kedaluwarsa. Untuk informasi selengkapnya, lihat [AWS Backup Panduan Developer](#).

Selain AWS Backup, Anda dapat menjadwalkan backup berkala atau masa depan dengan menggunakan AWS Lambda fungsi. Untuk informasi lebih lanjut, lihat posting blog [Solusi tanpa server untuk menjadwalkan cadangan Amazon DynamoDB On-Demand](#).

Jika Anda menggunakan konsol, pencadangan apa pun yang dibuat menggunakan AWS Backup tercantum pada tab Cadangan dengan Jenis cadangan diatur ke AWS.

Note

Anda tidak dapat menghapus pencadangan yang ditandai dengan Jenis cadangan dari AWS menggunakan konsol DynamoDB. Untuk mengelola cadangan ini, gunakan konsol AWS Backup.

Untuk mempelajari cara melakukan pencadangan, lihat [Membuat backup tabel DynamoDB](#).

Mengembalikan

Anda memulihkan tabel tanpa mengkonsumsi throughput apa pun yang disediakan pada tabel. Anda dapat melakukan pemulihan tabel penuh dari cadangan DynamoDB Anda, atau Anda dapat mengkonfigurasi pengaturan tabel tujuan. Ketika Anda melakukan pemulihan, Anda dapat mengubah pengaturan tabel berikut:

- Indeks sekunder global (GSI)
- Indeks sekunder lokal (LSIs)
- Mode penagihan
- Kapasitas baca dan tulis yang disediakan
- Pengaturan Enkripsi

 Important

Ketika Anda melakukan pemulihan tabel penuh, tabel tujuan diatur dengan unit kapasitas baca dan unit kapasitas tulis yang sama seperti tabel sumber, seperti yang tercatat pada saat cadangan diminta. Proses pemulihan juga memulihkan indeks sekunder lokal dan indeks sekunder global.


Anda juga dapat memulihkan data tabel DynamoDB Anda di Wilayah AWS sehingga tabel yang dipulihkan dibuat di Wilayah yang berbeda dari tempat cadangan berada. Anda dapat melakukan pengembalian lintas wilayah antara AWS Kawasan komersial, AWS Wilayah China, dan AWS GovCloud(AS) Wilayah. Anda hanya membayar untuk data yang Anda transfer keluar dari Wilayah sumber dan untuk memulihkan ke tabel baru di Wilayah tujuan.

Pemulihan dapat lebih cepat dan lebih hemat biaya jika Anda memilih untuk mengecualikan beberapa atau semua indeks sekunder dari pembuatan pada tabel yang baru dipulihkan.

Anda harus secara manual mengatur berikut ini pada tabel yang dipulihkan:

- Kebijakan Auto Scaling
- Kebijakan (IAM) AWS Identity and Access Management
- AmazonCloudWatchmetrik dan alarm
- Tanda
- Pengaturan stream
- Pengaturan Time to Live (TTL)
- Pengaturan perlindungan penghapusan
- Pengaturan Point in Time Recovery (PITR)

Anda hanya dapat memulihkan seluruh data tabel ke tabel baru dari pencadangan. Anda dapat menulis ke tabel yang dipulihkan hanya setelah menjadi aktif.

 Note

Anda tidak dapat menimpa tabel yang ada selama operasi pemulihan.

Metrik layanan menunjukkan bahwa 95 persen dari pemulihan tabel pelanggan selesai dalam waktu kurang dari satu jam. Namun, waktu pemulihan secara langsung berhubungan dengan konfigurasi tabel Anda (seperti ukuran tabel Anda dan jumlah partisi yang mendasari) dan variabel terkait lainnya. Praktik terbaik saat merencanakan pemulihan bencana adalah secara teratur mendokumentasikan waktu penyelesaian pemulihan rata-rata dan menetapkan bagaimana waktu ini mempengaruhi Tujuan Waktu Pemulihan Anda secara keseluruhan.

Untuk mempelajari cara melakukan pemulihan, lihat [Memulihkan tabel DynamoDB dari cadangan](#).

Anda dapat menggunakan kebijakan IAM untuk kontrol akses. Untuk informasi selengkapnya, lihat [Menggunakan IAM dengan Pencadangan dan Pemulihan DynamoDB](#).

Semua konsol cadangan dan pemulihan dan tindakan API ditangkap dan direkam di AWS CloudTrail untuk pencatatan, pemantauan berkelanjutan, dan audit.

Membuat backup tabel DynamoDB

Bagian ini menjelaskan cara menggunakan konsol Amazon DynamoDB atau AWS Command Line Interface untuk membuat cadangan tabel.

Topik

- [Membuat cadangan tabel \(konsol\)](#)
- [Membuat cadangan tabel \(AWS CLI\)](#)

Membuat cadangan tabel (konsol)

Ikuti langkah-langkah ini untuk membuat cadangan bernama `MusicBackup` untuk tabel `Music` yang sudah ada menggunakan AWS Management Console.

Untuk membuat cadangan tabel

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Anda dapat membuat cadangan dengan melakukan salah satu hal berikut:
 - Pada tab Cadangan dari tabel Music, pilih Buat cadangan.
 - Di panel navigasi pada sisi kiri konsol, pilih Cadangan. Kemudian pilih Buat cadangan.
3. Pastikan bahwa Music adalah nama tabel, dan masukkan **MusicBackup** untuk nama cadangan. Lalu, pilih Buat untuk membuat cadangan.

Create backup

Backup settings [Info](#)

Source table

Backup name

This will be used to identify your backup.

Between 3 and 255 characters in length. Only A-Z, a-z, 0-9, underscore characters, hyphens, and periods are allowed.

[Cancel](#) [Create backup](#)

Note

Jika Anda membuat cadangan menggunakan bagian Cadangan di panel navigasi, tabel tidak dipilih sebelumnya untuk Anda. Anda harus secara manual memilih nama tabel sumber untuk cadangan.

Sementara cadangan sedang dibuat, status cadangan diatur ke Membuat. Setelah pencadangan selesai, status cadangan berubah menjadi Tersedia.

On-demand backups (1) [Info](#) ↻ Restore Delete Create backup

🔍 Find backups by ARN or name

< 1 > ⚙️

<input type="checkbox"/>	Name	Status	Creatio...	ARN
<input type="checkbox"/>	MusicBackup	🟢 Available	August 23...	arn:aws:dynamodb:us-w

Membuat cadangan tabel (AWS CLI)

Ikuti langkah-langkah ini untuk membuat cadangan untuk tabel Music yang ada menggunakan AWS CLI.

Untuk membuat cadangan tabel

- Buat cadangan dengan nama MusicBackup untuk tabel Music.

```
aws dynamodb create-backup --table-name Music \  
--backup-name MusicBackup
```

Sementara cadangan sedang dibuat, status cadangan diatur ke CREATING.

```
{  
  "BackupDetails": {  
    "BackupName": "MusicBackup",  
    "BackupArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/  
backup/01489602797149-73d8d5bc",  
    "BackupStatus": "CREATING",  
    "BackupCreationDateTime": 1489602797.149  
  }  
}
```

Setelah pencadangan selesai, BackupStatus harus berubah ke AVAILABLE. Untuk mengonfirmasi hal ini, gunakan perintah describe-backup. Anda bisa mendapatkan nilai input backup-arn dari output dari langkah sebelumnya atau dengan menggunakan perintah list-backups.

```
aws dynamodb describe-backup \  
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/01489173575360-  
b308cd7d
```

Untuk melacak cadangan Anda, Anda dapat menggunakan perintah `list-backups`. Ini membuat daftar semua cadangan yang ada di `CREATING` atau status `AVAILABLE`.

```
aws dynamodb list-backups
```

Perintah `list-backups` dan perintah `describe-backup` untuk memeriksa informasi tentang tabel sumber dari cadangan.

Memulihkan tabel DynamoDB dari cadangan

Bagian ini menjelaskan cara memulihkan tabel dari pencadangan menggunakan konsol Amazon DynamoDB atau AWS Command Line Interface (AWS CLI).

Note

Jika ingin menggunakan AWS CLI, Anda harus mengonfigurasinya terlebih dahulu. Untuk informasi selengkapnya, lihat [Mengakses DynamoDB](#).

Topik

- [Memulihkan tabel dari cadangan \(konsol\)](#)
- [Memulihkan tabel dari cadangan \(AWS CLI\)](#)

Memulihkan tabel dari cadangan (konsol)

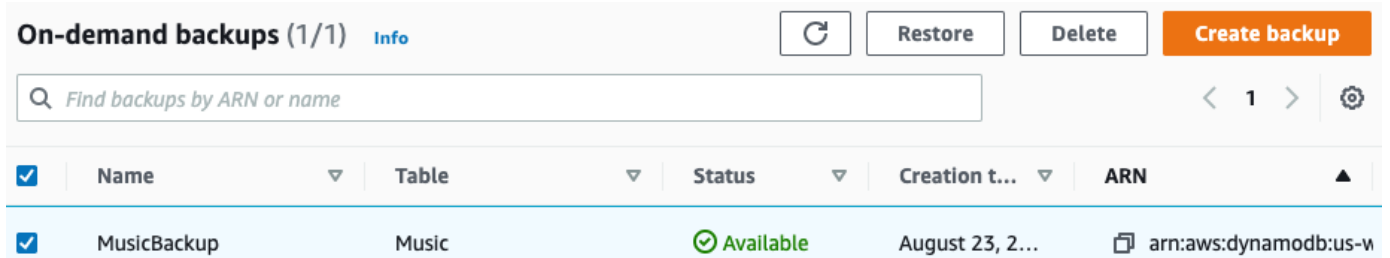
Prosedur berikut menunjukkan cara memulihkan tabel `Music` dengan menggunakan file `MusicBackup` yang dibuat di tutorial [Membuat backup tabel DynamoDB](#).

Note

Prosedur ini mengasumsikan bahwa tabel `Music` tidak ada lagi sebelum memulihkannya menggunakan file `MusicBackup`.

Untuk memulihkan tabel dari cadangan

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Dalam daftar cadangan, pilih MusicBackup.



The screenshot shows the 'On-demand backups' section in the AWS Management Console. At the top, there are buttons for 'Restore', 'Delete', and 'Create backup'. Below these is a search bar with the placeholder text 'Find backups by ARN or name'. A table lists the backup details:

<input checked="" type="checkbox"/>	Name	Table	Status	Creation t...	ARN
<input checked="" type="checkbox"/>	MusicBackup	Music	Available	August 23, 2...	arn:aws:dynamodb:us-w

4. Pilih Pemulihan.
5. Masukkan **Music** sebagai nama tabel baru. Konfirmasikan nama cadangan dan detail cadangan lainnya. Kemudian pilih Pulihkan tabel untuk memulai proses pemulihan.

Note

Anda dapat memulihkan tabel ke Wilayah AWS yang sama atau ke Wilayah yang berbeda dari tempat cadangan berada. Anda juga dapat mengecualikan indeks sekunder dari yang dibuat pada tabel yang baru dipulihkan. Selain itu, Anda dapat menentukan mode enkripsi yang berbeda.

Tabel yang dipulihkan dari cadangan selalu dibuat menggunakan kelas tabel DynamoDB Standard.

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.

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.

i 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

Tabel yang sedang dipulihkan ditampilkan dengan status `Membuat`. Setelah proses pemulihan selesai, status `Music` tabel berubah menjadi `Aktif`.

Memulihkan tabel dari cadangan (AWS CLI)

Ikuti langkah-langkah ini untuk menggunakan AWS CLI untuk memulihkan tabel `Music` menggunakan `MusicBackup` yang dibuat di tutorial [Membuat backup tabel DynamoDB](#).

Untuk memulihkan tabel dari cadangan

1. Konfirmasikan pencadangan yang ingin Anda pulihkan dengan menggunakan perintah `list-backups`. Contoh ini menggunakan `MusicBackup`.

```
aws dynamodb list-backups
```

Untuk mendapatkan detail tambahan untuk cadangan, gunakan perintah `describe-backup`. Anda bisa mendapatkan input `backup-arn` dari langkah sebelumnya.

```
aws dynamodb describe-backup \  
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/  
backup/01489173575360-b308cd7d
```

2. Pulihkan tabel dari cadangan. Dalam hal ini, `MusicBackup` memulihkan tabel `Music` ke Wilayah AWS yang sama.

```
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. Memulihkan tabel dari cadangan dengan pengaturan tabel kustom. Dalam hal ini, `MusicBackup` memulihkan tabel `Music` dan menentukan mode enkripsi untuk tabel yang dipulihkan.

Note

Parameter `sse-specification-override` mengambil nilai yang sama seperti yang digunakan parameter `sse-specification-override` dalam perintah `CreateTable`.

Untuk mempelajari informasi lebih lanjut, lihat [Mengelola tabel yang dienkrpsi di DynamoDB](#).

```
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
```

Anda dapat memulihkan tabel ke Wilayah AWS yang berbeda dari tempat cadangan berada.

Note


- Parameter `sse-specification-override` wajib untuk pemulihan lintas-wilayah tetapi opsional untuk pemulihan di Wilayah yang sama dengan tabel sumber.
- Saat melakukan pemulihan lintas wilayah dari baris perintah, Anda harus mengatur default Wilayah AWS ke Wilayah tujuan yang diinginkan. Untuk mempelajari lebih lanjut, lihat [Opsi baris perintah](#) di dalam [AWS Command Line Interface Panduan Pengguna](#).

```
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
```

Anda dapat menimpa modus penagihan dan throughput yang disediakan untuk tabel yang dipulihkan.


```
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
```


Anda dapat mengecualikan beberapa atau semua indeks sekunder dari yang dibuat pada tabel yang baru dipulihkan.

 Note

Pemulihan dapat lebih cepat dan lebih hemat biaya jika Anda mengecualikan beberapa atau semua indeks sekunder dari pembuatan pada tabel yang dipulihkan.

```
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

Indeks sekunder yang disediakan harus sesuai indeks yang ada. Anda tidak dapat membuat indeks baru pada saat pemulihan.

Anda dapat menggunakan kombinasi dari penimpanan yang berbeda. Sebagai contoh, Anda dapat menggunakan indeks sekunder global tunggal dan mengubah throughput yang disediakan pada saat yang sama, seperti berikut.

```
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
```

Untuk memverifikasi pemulihan, gunakan perintah `describe-table` untuk menggambarkan tabel `Music`.

```
aws dynamodb describe-table --table-name Music
```

Tabel yang sedang dipulihkan dari pencadangan ditampilkan dengan status `Membuat`. Setelah proses pemulihan selesai, status `Music` tabel berubah menjadi `Aktif`.

Important

Saat pemulihan sedang berlangsung, jangan mengubah atau menghapus kebijakan IAM role Anda; jika tidak, perilaku tak terduga dapat terjadi. Sebagai contoh, misalkan Anda menghapus izin tulis untuk tabel sementara tabel yang sedang dipulihkan. Dalam kasus ini, operasi `RestoreTableFromBackup` yang mendasar tidak akan dapat menulis data mana pun dipulihkan ke tabel.

Setelah operasi pemulihan selesai, Anda dapat mengubah atau menghapus kebijakan IAM role Anda.

Kebijakan IAM yang melibatkan [pembatasan IP sumber](#) untuk mengakses tabel pemulihan target harus memiliki set kunci [aws:ViaAWSService](#) ke `false` untuk memastikan bahwa pembatasan hanya berlaku untuk permintaan yang dibuat langsung oleh utama. Jika tidak, pemulihan akan dibatalkan.

Jika cadangan Anda dienkrpsi dengan Kunci yang dikelola AWS atau kunci terkelola pelanggan, jangan nonaktifkan atau hapus kunci saat pemulihan sedang berlangsung, atau pemulihan akan gagal.

Setelah operasi pemulihan selesai, Anda dapat mengubah kunci enkripsi untuk tabel yang dipulihkan dan menonaktifkan atau menghapus kunci lama.

Menghapus cadangan tabel DynamoDB

Bagian ini menjelaskan cara menggunakan AWS Management Console atau AWS Command Line Interface (AWS CLI) untuk menghapus cadangan tabel Amazon DynamoDB.

Note

Jika ingin menggunakan AWS CLI, Anda harus mengonfigurasinya terlebih dahulu. Untuk informasi selengkapnya, lihat [Menggunakan AWS CLI](#).

Topik

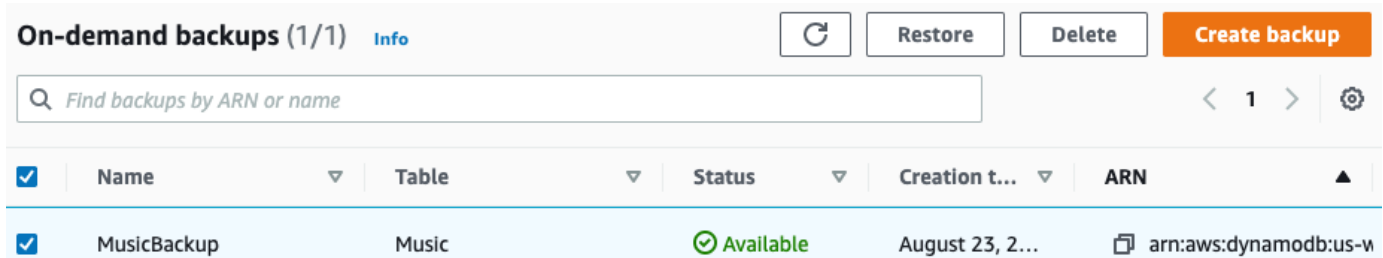
- [Menghapus cadangan tabel \(konsol\)](#)
- [Menghapus cadangan tabel \(AWS CLI\)](#)

Menghapus cadangan tabel (konsol)

Prosedur berikut menunjukkan cara menggunakan konsol untuk menghapus MusicBackup yang dibuat di tutorial [Membuat backup tabel DynamoDB](#).

Untuk menghapus pencadangan

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Dalam daftar cadangan, pilih MusicBackup.



4. Pilih Delete (Hapus). Mengonfirmasi bahwa Anda ingin menghapus cadangan dengan mengetik **delete** dan mengklik Hapus.

Menghapus cadangan tabel (AWS CLI)

Contoh berikut menghapus cadangan untuk tabel yang ada Music tabel menggunakan AWS CLI.

```
aws dynamodb delete-backup \  
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/  
backup/01489602797149-73d8d5bc
```

Menggunakan IAM dengan Pencadangan dan Pemulihan DynamoDB

Anda dapat menggunakan AWS Identity and Access Management (IAM) untuk membatasi tindakan pemulihan dan pencadangan Amazon DynamoDB untuk beberapa sumber daya. API `CreateBackup` dan `RestoreTableFromBackup` beroperasi pada basis per-tabel.

Untuk informasi selengkapnya tentang menggunakan kebijakan IAM dalam DynamoDB, lihat [Kebijakan berbasis identitas untuk DynamoDB](#).

Berikut ini adalah contoh kebijakan IAM yang dapat Anda gunakan untuk mengonfigurasi fungsionalitas cadangan dan pemulihan tertentu di DynamoDB.

Contoh 1: Mengizinkan CreateBackup dan RestoreTableFromBackup Tindakan

Kebijakan IAM berikut memberikan izin untuk mengizinkan tindakan DynamoDB CreateBackup dan RestoreTableFromBackup pada semua tabel:

```
{
  "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

RestoreTableFromBackup Izin DynamoDB diperlukan pada cadangan sumber, dan izin baca dan tulis DynamoDB pada tabel target diperlukan untuk mengembalikan fungsionalitas. RestoreTableToPointInTime Izin DynamoDB diperlukan pada tabel sumber, dan izin baca dan tulis DynamoDB pada tabel target diperlukan untuk mengembalikan fungsionalitas.

Contoh 2: Mengizinkan CreateBackup dan Menolak RestoreTableFromBackup

Kebijakan IAM berikut memberikan izin untuk tindakan CreateBackup dan menolak tindakan RestoreTableFromBackup:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["dynamodb:CreateBackup"],
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": ["dynamodb:RestoreTableFromBackup"],
      "Resource": "*"
    }
  ]
}
```

Contoh 3: Izinkan ListBackups dan tolak CreateBackup dan RestoreTableFromBackup

Kebijakan IAM berikut memberikan izin untuk tindakan ListBackups dan menolak tindakan CreateBackup dan RestoreTableFromBackup:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["dynamodb:ListBackups"],
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": [
        "dynamodb:CreateBackup",
        "dynamodb:RestoreTableFromBackup"
      ],
      "Resource": "*"
    }
  ]
}
```

```
]
}
```

Contoh 4: Mengizinkan ListBackups dan Menolak DeleteBackup

Kebijakan IAM berikut memberikan izin untuk tindakan ListBackups dan menolak tindakan DeleteBackup:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["dynamodb:ListBackups"],
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": ["dynamodb>DeleteBackup"],
      "Resource": "*"
    }
  ]
}
```

Contoh 5: Izinkan RestoreTableFromBackup dan DescribeBackup untuk semua sumber daya dan DeleteBackup tolak cadangan tertentu

Kebijakan IAM berikut memberikan izin untuk tindakan RestoreTableFromBackup dan DescribeBackup dan menolak tindakan DeleteBackup untuk sumber daya cadangan tertentu:

```
{
  "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

RestoreTableFromBackup Izin DynamoDB diperlukan pada cadangan sumber, dan izin baca dan tulis DynamoDB pada tabel target diperlukan untuk mengembalikan fungsionalitas. RestoreTableToPointInTime Izin DynamoDB diperlukan pada tabel sumber, dan izin baca dan tulis DynamoDB pada tabel target diperlukan untuk mengembalikan fungsionalitas.

Contoh 6: Mengizinkan CreateBackup untuk tabel tertentu

Kebijakan IAM berikut memberikan izin untuk tindakan CreateBackup hanya pada tabel Movies:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",

```

```
        "Action": ["dynamodb:CreateBackup"],
        "Resource": [
            "arn:aws:dynamodb:us-east-1:123456789012:table/Movies"
        ]
    }
]
```

Contoh 7: Mengizinkan ListBackups

Kebijakan IAM berikut memberikan izin untuk tindakan ListBackups berikut:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["dynamodb:ListBackups"],
      "Resource": "*"
    }
  ]
}
```

Important

Anda tidak dapat memberikan izin untuk tindakan ListBackups di tabel tertentu.

Contoh 8: Izinkan akses keAWS Backup fitur

Anda akan memerlukan izin API untukStartAwsBackupJob tindakan untuk cadangan sukses dengan fitur-fitur canggih, dandynamodb:RestoreTableFromAwsBackup tindakan untuk berhasil memulihkan cadangan itu.

Kebijakan IAM berikut memberikanAWS Backup izin untuk memicu cadangan dengan fitur-fitur canggih dan pemulihan. Perhatikan juga bahwa jika tabel dienkrpsi, kebijakan akan memerlukan akses ke [kunciAWS KMS](#).

```
{
  "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:account-id:table/Books"
    },
    {
      "Sid": "AllowRestoreFromAwsBackup",
      "Effect": "Allow",
      "Action": ["dynamodb:RestoreTableFromAwsBackup"],
      "Resource": "*"
    },
  ],
}

```

Contoh 9: Tolak RestoreTableToPointInTime untuk Tabel Sumber Tertentu

Kebijakan IAM berikut menolak izin untuk `RestoreTableToPointInTime` tindakan untuk tabel sumber tertentu:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "dynamodb:RestoreTableToPointInTime"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music"
    }
  ]
}

```

Contoh 10: Tolak RestoreTableFromBackup untuk semua Cadangan untuk Tabel Sumber Tertentu

Kebijakan IAM berikut menolak izin untuk `RestoreTableToPointInTime` tindakan untuk semua cadangan untuk tabel sumber tertentu:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "dynamodb:RestoreTableFromBackup"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/*"
    }
  ]
}
```

Point-in-time Pemulihan P untuk DynamoDB

Anda dapat membuat cadangan sesuai permintaan dari tabel Amazon DynamoDB Anda, atau Anda dapat mengaktifkan pencadangan berkelanjutan menggunakan pemulihan point-in-time. Untuk informasi selengkapnya tentang pencadangan sesuai permintaan, lihat [Menggunakan cadangan Sesuai Permintaan dan DynamoDB Permintaan](#).

Point-in-time Pemulihan P membantu melindungi tabel DynamoDB Anda dari operasi tulis atau hapus yang tidak disengaja. Dengan point-in-time pemulihan, Anda tidak perlu khawatir tentang membuat, memelihara, atau menjadwalkan cadangan sesuai permintaan. Misalnya, anggaplah skrip pengujian tidak sengaja menulis ke tabel DynamoDB produksi. Dengan point-in-time pemulihan, Anda dapat mengembalikan tabel itu ke titik waktu mana pun selama 35 hari terakhir. Setelah Anda mengaktifkan point-in-time pemulihan, Anda dapat mengembalikan ke titik waktu dari lima menit sebelum waktu saat ini hingga 35 hari yang lalu. DynamoDB memelihara cadangan tambahan tabel Anda.

Selain itu, point-in-time operasi tidak memengaruhi kinerja atau latensi API. Untuk informasi selengkapnya, lihat [Point-in-time recovery: Cara kerjanya](#).

Anda dapat mengembalikan tabel DynamoDB ke titik waktu menggunakan AWS Management Console, AWS CLI (), AWS Command Line Interface atau DynamoDB API. Proses point-in-time pemulihan mengembalikan ke tabel baru. Untuk informasi selengkapnya, lihat [Memulihkan tabel DynamoDB ke titik waktu](#).

Video berikut akan memberi Anda tampilan pengantar konsep pencadangan dan pemulihan dan berbicara lebih banyak tentang point-in-time pemulihan.

[Pencadangan dan pemulihan](#)

Untuk informasi selengkapnya tentang ketersediaan dan harga AWS Wilayah, lihat harga [Amazon DynamoDB](#).

Topik

- [Point-in-time recovery: Cara kerjanya](#)
- [Sebelum Anda mulai menggunakan point-in-time pemulihan](#)
- [Memulihkan tabel DynamoDB ke titik waktu](#)

Point-in-time recovery: Cara kerjanya

Amazon point-in-time DynamoDB recovery (PITR) menyediakan backup otomatis data tabel DynamoDB Anda. Bagian ini memberikan gambaran umum tentang cara kerja proses di DynamoDB.

Mengaktifkan pemulihan point-in-time

Anda dapat mengaktifkan point-in-time pemulihan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau DynamoDB API. Saat diaktifkan, point-in-time pemulihan menyediakan pencadangan berkelanjutan hingga Anda mematakannya secara eksplisit. Untuk informasi selengkapnya, lihat [Memulihkan tabel DynamoDB ke titik waktu](#).

Setelah Anda mengaktifkan point-in-time pemulihan, Anda dapat mengembalikan ke titik waktu mana pun di dalam `EarliestRestorableDateTime` dan `LatestRestorableDateTime`. `LatestRestorableDateTime` biasanya lima menit sebelum waktu saat ini.

Note

Proses point-in-time pemulihan selalu kembali ke tabel baru.

Memulihkan tabel menggunakan point-in-time pemulihan

Untuk `EarliestRestorableDateTime`, Anda dapat memulihkan tabel Anda ke titik waktu mana pun selama 35 hari terakhir. Periode penyimpanan bersifat tetap selama 35 hari (5 minggu kalender) dan tidak dapat diubah. Sejumlah pengguna dapat menjalankan hingga 50 pemulihan secara bersamaan (semua jenis pemulihan) di akun tertentu.

⚠ Important

Jika Anda menonaktifkan point-in-time pemulihan dan kemudian mengaktifkannya kembali di atas meja, Anda mengatur ulang waktu mulai di mana Anda dapat memulihkan tabel itu. Akibatnya, Anda hanya dapat segera memulihkan tabel tersebut menggunakan `LatestRestorableDateTime`.

Saat Anda memulihkan menggunakan point-in-time pemulihan, DynamoDB mengembalikan data tabel Anda ke status berdasarkan tanggal dan waktu `day:hour:minute:second ()` yang dipilih ke tabel baru.

Anda memulihkan tabel tanpa menggunakan throughput apa pun yang disediakan pada tabel. Anda dapat melakukan pemulihan tabel lengkap menggunakan point-in-time pemulihan, atau Anda dapat mengonfigurasi pengaturan tabel tujuan. Anda dapat mengubah pengaturan tabel berikut pada tabel yang dipulihkan:

- Indeks sekunder global (GSI)
- Indeks sekunder lokal (LSI)
- Mode penagihan
- Kapasitas baca dan tulis yang disediakan
- Pengaturan enkripsi

⚠ Important

Saat Anda melakukan pemulihan tabel penuh, tabel tujuan diatur dengan unit kapasitas baca dan tulis yang disediakan yang sama seperti yang dimiliki tabel sumber saat cadangan diminta. Misalnya, throughput tabel yang disediakan baru-baru ini diturunkan menjadi 50 unit kapasitas baca dan 50 unit kapasitas tulis. Anda kemudian memulihkan status tabel ke tiga minggu lalu, saat throughput yang disediakan ditetapkan ke 100 unit kapasitas baca dan 100 unit kapasitas tulis. Dalam hal ini, DynamoDB memulihkan data tabel Anda ke titik waktu tersebut dengan throughput yang disediakan sejak saat itu (100 unit kapasitas baca dan 100 unit kapasitas tulis).

Anda juga dapat memulihkan data tabel DynamoDB AWS di seluruh Wilayah sehingga tabel yang dipulihkan dibuat di Wilayah yang berbeda dari tempat tabel sumber berada. Anda dapat melakukan pemulihan lintas wilayah antara Wilayah AWS komersial, Wilayah AWS China, dan Wilayah AWS GovCloud (AS). Anda hanya membayar untuk data yang Anda transfer keluar dari Wilayah sumber dan untuk memulihkan ke tabel baru di Wilayah tujuan.

Note

Pemulihan lintas wilayah tidak didukung jika sumber atau wilayah tujuan adalah Asia Pasifik (Hong Kong) atau Timur Tengah (Bahrain).

Pemulihan bisa lebih cepat dan hemat biaya jika Anda mengecualikan beberapa atau semua indeks agar tidak dibuat pada tabel yang dipulihkan.

Anda harus mengatur hal berikut secara manual pada tabel yang dipulihkan:

- Kebijakan penskalaan otomatis
- AWS Identity and Access Management Kebijakan (IAM)
- CloudWatch Metrik dan alarm Amazon
- Tag
- Pengaturan aliran
- Pengaturan Waktu untuk Tayang (TTL)
- Pengaturan oint-in-time pemulihan P

Waktu yang Anda perlukan untuk memulihkan tabel bervariasi berdasarkan beberapa faktor. Waktu point-in-time pemulihan tidak selalu berkorelasi langsung dengan ukuran tabel. Untuk informasi selengkapnya, lihat [Mengembalikan](#).

Menghapus tabel dengan point-in-time pemulihan diaktifkan

Saat Anda menghapus tabel yang mengaktifkan point-in-time pemulihan, DynamoDB secara otomatis membuat snapshot cadangan yang disebut cadangan sistem dan mempertahankannya selama 35 hari (tanpa biaya tambahan). Anda dapat menggunakan cadangan sistem untuk memulihkan tabel yang dihapus ke keadaan sebelum penghapusan. Semua cadangan sistem mengikuti konvensi penamaan standar `table-name$DeletedTableBackup`.

Note

Setelah tabel dengan point-in-time pemulihan diaktifkan dihapus, Anda dapat menggunakan pemulihan sistem untuk mengembalikan tabel itu ke satu titik waktu: saat sebelum penghapusan. Anda tidak memiliki kemampuan untuk memulihkan tabel yang dihapus ke titik waktu lain dalam 35 hari terakhir.

Sebelum Anda mulai menggunakan point-in-time pemulihan

Sebelum Anda mengaktifkan point-in-time pemulihan (PITR) pada tabel Amazon DynamoDB, pertimbangkan hal berikut:

- Jika Anda menonaktifkan point-in-time pemulihan dan kemudian mengaktifkannya kembali di atas meja, Anda mengatur ulang waktu mulai di mana Anda dapat memulihkan tabel itu. Akibatnya, Anda hanya dapat segera memulihkan tabel tersebut menggunakan `LatestRestorableDateTime`.
- Anda dapat mengaktifkan point-in-time pemulihan pada setiap replika lokal tabel global. Saat Anda memulihkan tabel, cadangan akan dikembalikan ke tabel independen yang bukan bagian dari tabel global. Jika Anda menggunakan [Tabel Global versi 2019.11.21 \(Saat ini\)](#) dari tabel global, Anda dapat membuat tabel global baru dari tabel yang dipulihkan. Untuk informasi selengkapnya, lihat [Tabel global: Cara kerjanya](#).
- Anda juga dapat memulihkan data tabel DynamoDB di seluruh Wilayah AWS sehingga tabel yang dipulihkan dibuat di Wilayah yang berbeda dari tempat tabel sumber berada. Anda dapat melakukan pemulihan lintas wilayah antara Wilayah AWS komersial, Wilayah AWS China, dan Wilayah AWS GovCloud (AS). Anda hanya membayar untuk data yang Anda transfer keluar dari Wilayah sumber dan untuk memulihkan ke tabel baru di Wilayah tujuan.
- AWS CloudTrail mencatat semua tindakan konsol dan API untuk point-in-time pemulihan guna mengaktifkan pencatatan, pemantauan berkelanjutan, dan audit. Lihat informasi yang lebih lengkap di [Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail](#).

Memulihkan tabel DynamoDB ke titik waktu

Amazon DynamoDB point-in-time pemulihan (PITR) menyediakan pencadangan berkelanjutan data tabel DynamoDB Anda. Anda dapat memulihkan tabel ke titik waktu menggunakan konsol

DynamoDB atau AWS Command Line Interface (AWS CLI). Parameter point-in-time memulihkan ke tabel baru.

Jika ingin menggunakan AWS CLI, Anda harus mengonfigurasinya terlebih dahulu. Untuk informasi selengkapnya, lihat [Mengakses DynamoDB](#).

Topik

- [Memulihkan tabel DynamoDB ke titik waktu \(konsol\)](#)
- [Memulihkan tabel ke titik waktu \(AWS CLI\)](#)

Memulihkan tabel DynamoDB ke titik waktu (konsol)

Contoh berikut menunjukkan cara menggunakan konsol DynamoDB untuk memulihkan tabel yang ada bernama `Music` ke sebuah titik waktu.

Note

Prosedur ini mengasumsikan bahwa Anda telah mengaktifkan point-in-time pemulihan Untuk mengaktifkannya untuk `Music` meja, di `Cadangan tab`, di `Point-in-time pemulihan (PITR)` bagian, pilih `ditlalu centang kotak` di samping `Aktifkan point-in-time-recovery`.

Untuk memulihkan tabel ke titik waktu

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi pada sisi kiri konsol, pilih `Tabel`.
3. Dalam daftar tabel, pilih tabel `Music`.
4. Pada `Cadangan tab` dari `Music` tabel, di `Point-in-time pemulihan (PITR)` bagian, pilih `Pemulihan`.
5. Untuk nama tabel baru, masukkan **`MusicMinutesAgo`**.

Note

Anda dapat memulihkan tabel ke Wilayah AWS yang sama atau ke Wilayah yang berbeda dari tempat tabel sumber berada. Anda juga dapat mengecualikan indeks sekunder dari yang dibuat pada tabel yang dipulihkan. Selain itu, Anda dapat menentukan mode enkripsi yang berbeda.

6. Untuk mengkonfirmasi waktu yang dapat dipulihkan, atur memulihkan tanggal dan waktu kePaling Awal. Lalu pilihPemulihanuntuk memulai proses pemulihan.

Tabel yang sedang dipulihkan ditampilkan dengan status Memulihkan. Setelah proses pemulihan selesai, status MusicMinutesAgo tabel berubah menjadi Aktif.

Memulihkan tabel ke titik waktu (AWS CLI)

Prosedur berikut menunjukkan cara menggunakan AWS CLI untuk memulihkan tabel yang ada bernama Music ke titik waktu.

Note

Prosedur ini mengasumsikan bahwa Anda telah mengaktifkan point-in-time pemulihan Untuk mengaktifkannya untuk tabel Music, jalankan perintah berikut.

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

Untuk memulihkan tabel ke titik waktu

1. Konfirmasi bahwa point-in-time pemulihan diaktifkan untukMusictabel dengan menggunakandescribe-continuous-backupsperintah.

```
aws dynamodb describe-continuous-backups \  
  --table-name Music
```

Pencadangan berkelanjutan (otomatis diaktifkan pada pembuatan tabel) dan point-in-timepemulihan diaktifkan.

```
{  
  "ContinuousBackupsDescription": {  
    "PointInTimeRecoveryDescription": {  
      "PointInTimeRecoveryStatus": "ENABLED",  
      "EarliestRestorableDateTime": 1519257118.0,  
      "LatestRestorableDateTime": 1520018653.01
```



```
    },  
    "ContinuousBackupsStatus": "ENABLED"  
  }  
}
```

2. Memulihkan tabel ke titik waktu. Dalam hal ini, tabel Music dipulihkan ke LatestRestorableDateTime (~5 menit lalu) ke Wilayah AWS yang sama.

```
aws dynamodb restore-table-to-point-in-time \  
  --source-table-name Music \  
  --target-table-name MusicMinutesAgo \  
  --use-latest-restorable-time
```

Note

Anda juga dapat memulihkan ke titik waktu tertentu. Untuk melakukannya, jalankan perintah menggunakan argumen `--restore-date-time`, dan tentukan stempel waktu. Anda dapat menentukan titik waktu apa pun selama 35 hari terakhir. Sebagai contoh, perintah berikut memulihkan tabel ke EarliestRestorableDateTime.

```
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
```

Menentukan argumen `--no-use-latest-restorable-time` adalah opsional ketika memulihkan ke titik waktu tertentu.

3. Memulihkan tabel ke titik waktu dengan pengaturan tabel kustom. Dalam hal ini, tabel Music dipulihkan ke LatestRestorableDateTime (~5 menit lalu).

Anda dapat menentukan mode enkripsi yang berbeda untuk tabel dipulihkan, sebagai berikut.

Note

Parameter `sse-specification-override` mengambil nilai yang sama seperti yang digunakan parameter `sse-specification-override` dalam perintah `CreateTable`.

Untuk mempelajari informasi lebih lanjut, lihat [Mengelola tabel yang dienkrpsi di DynamoDB](#).

```
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
```

Anda dapat memulihkan tabel ke Wilayah AWS dari tempat tabel sumber berada.

Note

- Parameter `sse-specification-override` wajib untuk pemulihan lintas-wilayah tetapi opsional untuk pemulihan ke Wilayah yang sama dengan tabel sumber.
- Parameter `source-table-arn` harus disediakan untuk pemulihan lintas-wilayah.
- Saat melakukan pemulihan lintas wilayah dari baris perintah, Anda harus mengatur default Wilayah AWS ke Wilayah tujuan yang diinginkan. Untuk mempelajari lebih lanjut, lihat [Opsional baris perintah](#) di dalam [AWS Command Line Interface Panduan Pengguna](#).

```
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
```

Anda dapat menimpa modus penagihan dan throughput yang disediakan untuk tabel yang dipulihkan.

```
aws dynamodb restore-table-to-point-in-time \  
  --source-table-name Music \  
  --target-table-name MusicMinutesAgo \  
  --use-latest-restorable-time
```

```
--use-latest-restorable-time \  
--billing-mode-override PAY_PER_REQUEST
```

Anda dapat mengecualikan beberapa atau semua indeks sekunder dari yang dibuat pada tabel yang baru dipulihkan.

Note

Pemulihan dapat lebih cepat dan lebih hemat biaya jika Anda mengecualikan beberapa atau semua indeks sekunder dari pembuatan pada tabel yang baru dipulihkan.

```
aws dynamodb restore-table-to-point-in-time \  
--source-table-name Music \  
--target-table-name MusicMinutesAgo \  
--use-latest-restorable-time \  
--global-secondary-index-override '[]'
```

Anda dapat menggunakan kombinasi dari penimpanan yang berbeda. Sebagai contoh, Anda dapat menggunakan indeks sekunder global tunggal dan mengubah throughput yang disediakan pada saat yang sama, seperti berikut.

```
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=KEYS}" \  
\   
--sse-specification-override Enabled=true,SSEType=KMS \  
--use-latest-restorable-time
```

Untuk memverifikasi pemulihan, gunakan perintah `describe-table` untuk menggambarkan tabel `MusicEarliestRestorableDateTime`.

```
aws dynamodb describe-table --table-name MusicEarliestRestorableDateTime
```

Tabel yang sedang dipulihkan ditampilkan dengan status Membuat dan pemulihan dalam proses sebagai benar. Setelah proses pemulihan selesai, status `MusicEarliestRestorableDateTime` tabel berubah menjadi Aktif.

 **Important**

Ketika pemulihan sedang berlangsung, jangan memodifikasi atau menghapus kebijakan (IAM) AWS Identity and Access Management yang memberikan izin entitas IAM (misalnya, pengguna, grup, atau peran) untuk melakukan pemulihan. Jika tidak, perilaku tak terduga dapat terjadi. Sebagai contoh, misalkan Anda menghapus izin tulis untuk tabel sementara tabel yang sedang dipulihkan. Dalam kasus ini, operasi `RestoreTableToPointInTime` yang mendasar tidak dapat menulis data mana pun yang dipulihkan ke tabel. Kebijakan IAM yang melibatkan pembatasan IP sumber untuk mengakses tabel pemulihan target juga dapat menyebabkan masalah.

Anda dapat mengubah atau menghapus izin hanya setelah operasi pemulihan selesai.

Akselerasi dalam memori dengan DynamoDB Accelerator (DAX)

Amazon DynamoDB dirancang untuk skala dan performa. Dalam kebanyakan kasus, waktu respons DynamoDB dapat diukur dalam milidetik satu digit. Namun, ada kasus penggunaan tertentu yang memerlukan waktu respons dalam mikrodetik. Untuk kasus penggunaan ini, DynamoDB Accelerator (DAX) memberikan waktu respons yang cepat untuk mengakses data yang akhirnya konsisten.

DAX adalah layanan caching yang kompatibel dengan DynamoDB yang memungkinkan Anda mendapatkan manfaat dari performa dalam memori yang cepat untuk aplikasi yang menuntut. DAX membahas tiga skenario inti:

1. Sebagai cache dalam memori, DAX mengurangi waktu respons dari beban kerja bacaan akhir konsisten dengan urutan besarnya dari milidetik satu digit menjadi mikrodetik.
2. DAX mengurangi kompleksitas operasional dan aplikasi dengan menyediakan layanan terkelola yang kompatibel dengan API DynamoDB. Oleh karena itu, DAX hanya memerlukan perubahan fungsional minimal untuk digunakan dengan aplikasi yang ada.
3. Untuk beban kerja baca berat atau bursty, DAX menyediakan peningkatan throughput dan potensi penghematan biaya operasional dengan mengurangi kebutuhan untuk menyediakan unit kapasitas baca yang berlebihan. Ini amat bermanfaat untuk aplikasi yang memerlukan bacaan berulang untuk kunci individu.

DAX mendukung enkripsi sisi server. Dengan enkripsi saat istirahat, data yang disimpan oleh DAX pada disk akan dienkripsi. DAX membuat perintah tulis data ke disk sebagai bagian dari menyebarkan perubahan dari simpul primer untuk replika baca. Untuk informasi selengkapnya, lihat [Enkripsi DAX saat istirahat](#).

DAX juga mendukung enkripsi dalam transit memastikan bahwa semua permintaan dan tanggapan antara aplikasi dan kluster dienkripsi oleh Keamanan Transportasi Tingkat (TLS), dan koneksi ke kluster dapat dikonfirmasi oleh verifikasi sertifikat kluster x509. Untuk informasi selengkapnya, lihat [Enkripsi DAX dalam transit](#).

Topik

- [Kasus penggunaan untuk DAX](#)
- [Catatan penggunaan DAX](#)

- [DAX: Cara kerjanya](#)
- [Komponen DAX](#)
- [Membuat klaster DAX](#)
- [Model konsistensi DAX dan DynamoDB](#)
- [Mengembangkan dengan klien DynamoDB Accelerator \(DAX\)](#)
- [Mengelola klaster DAX](#)
- [Pemantauan DAX](#)
- [Instans Burstable DAX T3/T2](#)
- [Kontrol akses DAX](#)
- [Enkripsi DAX saat istirahat](#)
- [Enkripsi DAX dalam transit](#)
- [Menggunakan peran IAM tertaut layanan untuk DAX](#)
- [Mengakses DAX di seluruh akun AWS](#)
- [Panduan ukuran cluster DAX](#)
- [Referensi API DAX DAX](#)

Kasus penggunaan untuk DAX

DAX menyediakan akses ke data konsisten dari tabel DynamoDB, dengan latensi mikrodetik. klaster Multi-AZ DAX dapat melayani jutaan permintaan per detik.

DAX sesuai untuk jenis aplikasi berikut:

- Aplikasi yang membutuhkan waktu respons tercepat untuk pembacaan. Beberapa contoh termasuk penawaran waktu nyata, game sosial, dan aplikasi perdagangan. DAX memberikan performa baca dalam memori yang cepat untuk kasus penggunaan ini.
- Aplikasi yang membaca sejumlah kecil item yang lebih sering dibanding yang lain. Misalnya, pertimbangkan sistem perdagangn elektronik yang memiliki penjualan satu hari pada produk populer. Selama penjualan, permintaan untuk produk tersebut (dan data di DynamoDB) akan meningkat tajam, dibandingkan dengan semua produk lainnya. Untuk mengurangi dampak kunci "hot" dan distribusi lalu lintas yang tidak seragam, Anda dapat meniadakan beban aktivitas baca ke cache DAX sampai penjualan satu hari berakhir.
- Aplikasi yang dibaca intensif, namun sensitif juga terkait biaya. Dengan DynamoDB, Anda menetapkan jumlah pembacaan per detik yang dibutuhkan oleh aplikasi Anda. Jika aktivitas baca

meningkat, Anda dapat meningkatkan throughput baca yang ditetapkan tabel Anda (dengan biaya tambahan). Atau, Anda dapat meniadakan beban aktivitas dari aplikasi Anda ke klaster DAX, dan mengurangi jumlah unit kapasitas baca yang Anda butuhkan untuk membeli.

- Aplikasi yang memerlukan pembacaan dengan berulang kali terhadap satu set besar data. Aplikasi semacam itu berpotensi mengalihkan sumber daya basis data dari aplikasi lain. Sebagai contoh, analisis jangka panjang cuaca regional data sementara dapat mengkonsumsi semua kapasitas baca dalam tabel DynamoDB. Situasi ini akan berdampak negatif terhadap aplikasi lain yang perlu mengakses data yang sama. Dengan DAX, analisis cuaca dapat dilakukan terhadap data cache sebagai gantinya.

DAX tidak cocok untuk jenis aplikasi berikut:

- Aplikasi yang memerlukan pembacaan yang sangat konsisten (atau yang tidak dapat mentolerir pembacaan secara konsisten).
- Aplikasi yang tidak memerlukan waktu responns mikrodetik untuk pembacaan, atau yang tidak memerlukan peniadaan beban aktivitas pembacaan berulang dari tabel yang dasar.
- Aplikasi yang menulis secara intensif, atau yang tidak melakukan banyak aktivitas pembacaan.
- Aplikasi yang sudah menggunakan solusi caching yang berbeda dengan DynamoDB, dan menggunakan logika sisi klien mereka sendiri untuk bekerja dengan solusi caching tersebut.

Catatan penggunaan DAX

- Untuk daftar Wilayah AWS di mana DAX tersedia, lihat [Harga Amazon DynamoDB DB](#).
- DAX mendukung aplikasi yang ditulis dalam Go, Java, Node.js, Python, dan NET, menggunakan AWS yang disediakan klien untuk bahasa pemrograman tersebut.
- DAX hanya tersedia untuk platform EC2-VPC.
- Kebijakan peran Layanan klaster DAX harus mengizinkan tindakan `dynamodb:DescribeTable` untuk mempertahankan metadata tentang tabel DynamoDB.
- klaster DAX mempertahankan metadata tentang nama atribut item yang mereka simpan. Metadata tersebut dipertahankan tanpa batas waktu (bahkan setelah item kedaluwarsa atau telah dikeluarkan dari cache). Aplikasi yang menggunakan sejumlah nama atribut tak terbatas, dari waktu ke waktu, dapat menyebabkan kehabisan memori di klaster DAX. Keterbatasan ini hanya berlaku untuk nama atribut tingkat atas, bukan untuk nama atribut bersarang. Contoh nama atribut tingkat atas yang bermasalah termasuk stempel waktu, UUID, dan ID sesi.

Keterbatasan ini hanya berlaku untuk nama atribut, bukan nilai-nilai mereka. Item seperti berikut ini tidak menjadi masalah.

```
{
  "Id": 123,
  "Title": "Bicycle 123",
  "CreationDate": "2017-10-24T01:02:03+00:00"
}
```

Tapi item seperti berikut ini adalah masalah jika ada cukup dari mereka dan mereka masing-masing memiliki stempel waktu yang berbeda.

```
{
  "Id": 123,
  "Title": "Bicycle 123",
  "2017-10-24T01:02:03+00:00": "created"
}
```

DAX: Cara kerjanya

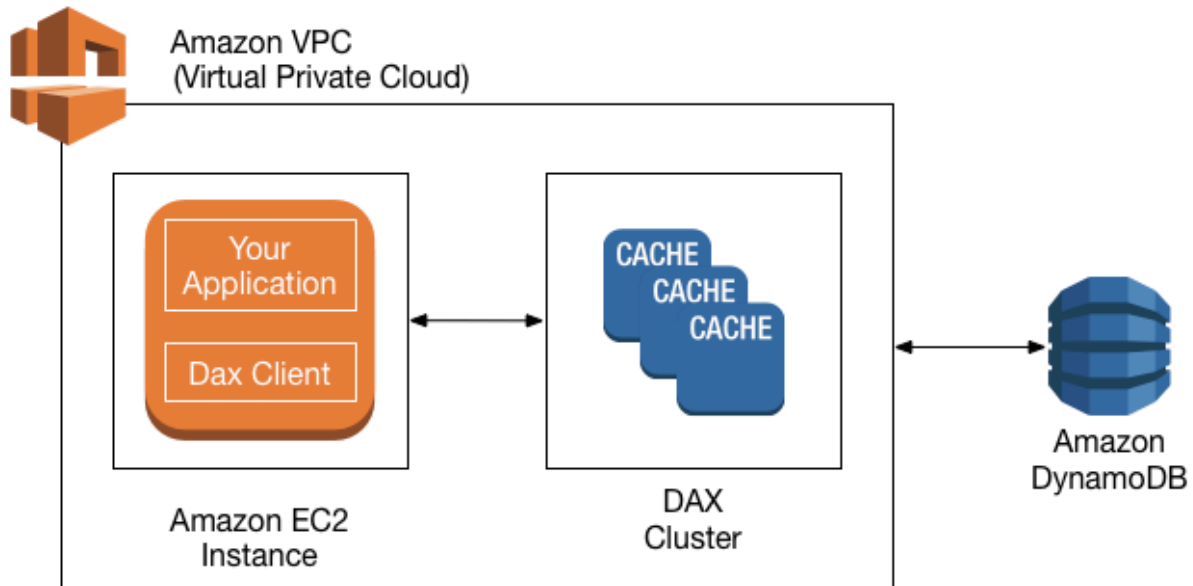
Amazon DynamoDB Accelerator (DAX) dirancang untuk berjalan di lingkungan Amazon Virtual Private Cloud (Amazon VPC). Layanan Amazon VPC mendefinisikan jaringan virtual yang mirip dengan pusat data tradisional. Dengan VPC, Anda memiliki kendali terhadap rentang alamat IP, subnet, tabel rute, gateway jaringan, dan pengaturan keamanan. Anda dapat meluncurkan kluster DAX di jaringan virtual Anda dan kontrol akses ke kluster dengan menggunakan kelompok-kelompok keamanan Amazon VPC.

Note

Jika Anda membuat akun AWS setelah 4 Desember 2013, Anda sudah memiliki VPC default di setiap Wilayah AWS. VPC siap untuk Anda gunakan dengan segera tanpa harus melakukan langkah-langkah konfigurasi tambahan.

Untuk informasi selengkapnya, lihat [VPC default dan subnet default](#) di dalam Panduan Pengguna Amazon VPC.

Diagram berikut menunjukkan gambaran umum tingkat tinggi DAX.



Untuk membuat klaster DAX, Anda menggunakan AWS Management Console. Kecuali Anda menentukan lain, klaster DAX Anda berjalan dalam VPC default Anda. Untuk menjalankan aplikasi Anda, Anda meluncurkan instans Amazon EC2 ke Amazon VPC Anda. Anda kemudian men-deploy aplikasi Anda (dengan klien DAX) pada instans EC2.

Pada saat waktu aktif, klien DAX mengarahkan semua permintaan API DynamoDB aplikasi Anda ke klaster DAX. Jika DAX dapat memproses salah satu permintaan API ini secara langsung, maka itu akan dilakukan. Jika tidak, itu akan melewati permintaan melalui DynamoDB.

Akhirnya, klaster DAX mengembalikan hasil ke aplikasi Anda.

Topik

- [Cara DAX Memproses Permintaan](#)
- [Cache Item](#)
- [Cache kueri](#)

Cara DAX Memproses Permintaan

klaster DAX terdiri atas satu atau lebih simpul. Setiap simpul menjalankan instans sendiri perangkat lunak caching DAX. Salah satu simpul berfungsi sebagai simpul primer untuk klaster. Simpul tambahan (jika ada) berfungsi sebagai replika baca. Untuk informasi selengkapnya, lihat [Simpul](#).

Aplikasi Anda dapat mengakses DAX dengan menentukan titik akhir untuk klaster DAX. Perangkat lunak klien DAX bekerja dengan titik akhir klaster untuk melakukan penyeimbangan beban dan perutean cerdas.

Operasi Baca

DAX dapat menanggapi panggilan API berikut:

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

Jika permintaan menentukan pembacaan yang konsisten (perilaku default), maka akan mencoba membaca item dari DAX:

- Jika DAX memiliki item yang tersedia (Cache hit), DAX mengembalikan item ke aplikasi tanpa mengakses DynamoDB.
- Jika DAX tidak memiliki item yang tersedia (cache terlewatkan), DAX melewati permintaan melalui DynamoDB. Ketika menerima respon dari DynamoDB, DAX mengembalikan hasil ke aplikasi. Tetapi DAX juga menulis hasil ke cache pada simpul primer.

Note

Jika ada replika pembacaan di instans, DAX secara otomatis membuat replika sinkron dengan simpul primer. Untuk informasi selengkapnya, lihat [klaster](#).

Jika permintaan menentukan Pembacaan yang sangat konsisten, DAX melewati permintaan melalui DynamoDB. Hasil dari DynamoDB tidak di-cache di DAX. Sebaliknya, mereka hanya kembali ke aplikasi.

Operasi Penulisan

Operasi API DAX berikut dianggap "write-through":

- `BatchWriteItem`
- `UpdateItem`
- `DeleteItem`
- `PutItem`

Dengan operasi ini, data pertama-tama ditulis ke tabel DynamoDB, dan kemudian ke kluster DAX. Operasi ini hanya dapat berhasil jika data berhasil ditulis di kedua tabel dan DAX.

Operasi lainnya

DAX tidak mengenali operasi DynamoDB untuk mengelola tabel (seperti `CreateTable`, `UpdateTable`, dan sebagainya). Jika aplikasi Anda perlu melakukan operasi ini, maka harus mengakses DynamoDB langsung daripada menggunakan DAX.

Untuk informasi terperinci tentang konsistensi DAX dan DynamoDB, lihat [Model konsistensi DAX dan DynamoDB](#).

Untuk informasi tentang cara kerja transaksi di DAX, lihat [Menggunakan API Transaksional di DynamoDB Accelerator \(DAX\)](#).

Pembatasan Laju Permintaan

Jika jumlah permintaan yang dikirim ke DAX melebihi kapasitas simpul, DAX membatasi tingkat di mana ia menerima permintaan tambahan dengan mengembalikan [ThrottlingException](#). DAX terus mengevaluasi utilisasi CPU Anda untuk menentukan volume permintaan dapat memproses sambil mempertahankan keadaan kluster yang sehat.

Anda dapat memantau [ThrottledRequestCount metris](#) yang diterbitkan DAX ke Amazon CloudWatch. Jika Anda melihat pengecualian ini secara teratur, Anda harus mempertimbangkan [meningkatkan skala kluster](#).

Cache Item

DAX mempertahankan cache item untuk menyimpan hasil dari `GetItem` dan operasi `BatchGetItem`. Item dalam cache mewakili data konsisten dari DynamoDB, dan disimpan oleh nilai-nilai kunci utama mereka.

Ketika aplikasi mengirimkan `GetItem` atau `BatchGetItem` permintaan, DAX mencoba untuk membaca item langsung dari cache item menggunakan nilai-nilai kunci yang ditentukan. Jika item yang ditemukan (cache hit), DAX mengembalikan mereka ke aplikasi segera. Jika item tidak ditemukan (cache miss), DAX mengirimkan permintaan ke DynamoDB. DynamoDB memproses permintaan menggunakan akhirnya konsisten membaca dan mengembalikan item ke DAX. DAX menyimpannya dalam cache item dan kemudian mengembalikannya ke aplikasi.

Cache item memiliki pengaturan Waktu untuk Tayang (TTL), yaitu 5 menit secara default. DAX memberikan stempel waktu untuk setiap item yang menulis ke cache item. Item kedaluwarsa jika tetap dalam cache lebih lama dari pengaturan TTL. Jika Anda mengeluarkan `GetItem` pada item kedaluwarsa, ini dianggap sebagai cache miss, dan DAX mengirimkan `GetItem` Permintaan ke DynamoDB.

Note

Anda dapat menentukan pengaturan TTL untuk item cache ketika Anda membuat sebuah klaster DAX baru. Untuk informasi selengkapnya, lihat [Mengelola klaster DAX](#).

DAX juga setidaknya mempertahankan daftar yang baru-baru ini digunakan (LRU) untuk cache item. Daftar LRU melacak saat item pertama kali ditulis ke cache, dan saat item terakhir dibaca dari cache. Jika cache item penuh, DAX menghapus item lama (walaupun belum kedaluwarsa) untuk membuat ruang untuk item baru. algoritme LRU selalu diaktifkan untuk cache item dan tidak dapat dikonfigurasi pengguna.

Jika Anda menentukan nol sebagai `CacheItemPengaturan TTL`, item di dalam cache item hanya akan di-refresh karena penghapusan LRU atau [“menulis-melalui”](#) Operasi.

Untuk informasi selengkapnya tentang konsistensi item cache di DAX, lihat [perilaku cache item DAX](#).

Cache kueri

DAX mempertahankan cache kueri untuk menyimpan hasil dari `Query` dan operasi `Scan`. Item dalam cache ini mewakili hasil set dari kueri dan pemindaian pada tabel DynamoDB. Hasil set ini disimpan oleh nilai parameter mereka.

Ketika aplikasi mengirimkan `Query` atau `Scan`, DAX mencoba untuk membaca hasil pencocokan ditetapkan dari cache kueri menggunakan nilai parameter tertentu. Jika hasil set ditemukan (cache hit), DAX mengembalikannya ke aplikasi segera. Jika hasil set tidak ditemukan (cache miss),

DAX mengirimkan permintaan ke DynamoDB. DynamoDB memproses permintaan menggunakan pembacaan konsisten dan mengembalikan set hasil ke DAX. DAX menyimpannya dalam cache kueri dan kemudian mengembalikannya ke aplikasi.

Note

Anda dapat menentukan pengaturan TTL untuk cache permintaan saat Anda membuat kluster DAX baru. Untuk informasi selengkapnya, lihat [Mengelola kluster DAX](#).

DAX juga mempertahankan daftar LRU untuk cache kueri. Daftar trek ketika set hasil pertama kali ditulis ke cache, dan ketika hasilnya terakhir dibaca dari cache. Jika cache kueri penuh, DAX menghapus set lama (walaupun belum kedaluwarsa) untuk membuat ruang untuk set baru. algoritme LRU selalu diaktifkan untuk cache kueri, dan tidak dapat dikonfigurasi pengguna.

Jika Anda menentukan nol sebagai cache kueri Pengaturan TTL, respon kueri tidak akan di-cache.

Untuk informasi selengkapnya tentang konsistensi cache kueri di DAX, lihat [perilaku cache kueri DAX](#).

Komponen DAX

Sebuah kluster Amazon DynamoDB Accelerator (DAX) terdiri dari komponen infrastruktur AWS. Bagian ini menjelaskan komponen-komponen ini dan cara mereka bekerja sama.

Topik

- [Simpul](#)
- [klaster](#)
- [Wilayah dan Availability Zone](#)
- [Grup parameter](#)
- [Grup keamanan](#)
- [klaster ARN](#)
- [Titik akhir klaster](#)
- [Titik akhir simpul](#)
- [Grup subnet](#)

- [Kejadian](#)
- [Jendela pemeliharaan](#)

Simpul

Simpul adalah blok bangunan terkecil dari kluster DAX. Setiap simpul menjalankan sebuah instans dari perangkat lunak DAX, dan mempertahankan replika tunggal data cache.

Anda dapat menskala kluster DAX Anda dengan salah satu dari dua cara berikut:

- Dengan menambahkan lebih banyak simpul ke kluster. Hal ini meningkatkan throughput pembacaan keseluruhan kluster.
- Dengan menggunakan jenis simpul yang lebih besar. Jenis simpul yang lebih besar memberikan kapasitas lebih dan dapat meningkatkan throughput. (Anda harus membuat sebuah kluster baru dengan jenis simpul baru.)

Setiap simpul dalam sebuah kluster adalah tipe simpul yang sama dan menjalankan perangkat lunak caching DAX yang sama. Untuk daftar jenis simpul yang tersedia, lihat [Harga Amazon DynamoDB](#).

kluster

kluster adalah pengelompokan logis dari satu atau lebih simpul yang dikelola DAX sebagai satu unit. Salah satu simpul dalam instans ditetapkan sebagai simpul primer, dan simpul lainnya (jika ada) adalah replika baca.

simpul primer bertanggung jawab untuk hal berikut:

- Memenuhi permintaan aplikasi untuk data cache.
- Menangani operasi penulisan untuk DynamoDB.
- Menghapus data dari cache sesuai dengan kebijakan pengosongan kluster.

Ketika perubahan dibuat untuk data cache pada simpul primer, DAX me-deloter untuk semua simpul replika baca menggunakan log replikasi. Setelah konfirmasi diterima dari semua replika baca, DynamoDB menghapus log replikasi dari node utama.

replika baca bertanggung jawab untuk hal berikut:

- Memenuhi permintaan aplikasi untuk data cache.
- Menghapus data dari cache sesuai dengan kebijakan pengosongan klaster.

Namun, tidak seperti simpul primer, replika baca tidak menulis ke DynamoDB.

replika baca melayani dua tujuan tambahan:

- Skalabilitas. Jika Anda memiliki sejumlah besar aplikasi klien yang perlu mengakses DAX secara bersamaan, Anda dapat menambahkan lebih banyak replika untuk skala pembacaan. DAX menyebar beban secara merata di semua simpul di klaster. (Cara lain untuk meningkatkan throughput adalah dengan menggunakan jenis simpul cache yang lebih besar.)
- Ketersediaan tinggi. Jika simpul primer mengalami kegagalan, DAX secara otomatis gagal melakukan replika baca dan menunjuknya sebagai primer baru. Jika simpul replika gagal, simpul lain dalam instans DAX masih dapat melayani permintaan sampai simpul yang mengalami kegagalan dapat dipulihkan. Untuk toleransi kesalahan maksimum, Anda harus men-deploy replika baca di Availability Zone terpisah. Konfigurasi ini memastikan bahwa klaster DAX Anda dapat terus berfungsi, bahkan jika seluruh Availability Zone menjadi tidak tersedia.

Klaster DAX dapat mendukung hingga 11 simpul per klaster (simpul primer ditambah maksimum 10 replika baca).

Important

Untuk penggunaan produksi, kami sangat menyarankan menggunakan DAX dengan setidaknya tiga simpul, di mana setiap simpul ditempatkan di Availability Zone yang berbeda. Tiga simpul diperlukan untuk klaster DAX menjadi toleran terhadap kesalahan. Satu klaster DAX dapat digunakan dengan satu atau dua simpul untuk developeran atau beban kerja pengujian. Klaster dengan satu dan dua simpul tidak toleran terhadap kesalahan, dan kami tidak merekomendasikan penggunaan kurang dari tiga simpul untuk penggunaan produksi. Jika satu atau dua simpul klaster mengalami kesalahan perangkat lunak atau perangkat keras, klaster dapat menjadi tidak tersedia atau kehilangan data cache.

Wilayah dan Availability Zone

Sebuah klaster DAX di Wilayah AWS hanya dapat berinteraksi dengan tabel DynamoDB yang berada di Wilayah yang sama. Untuk alasan ini, pastikan bahwa Anda meluncurkan klaster DAX

Anda di Wilayah yang benar. Jika Anda memiliki tabel DynamoDB di Wilayah lain, Anda juga harus meluncurkan klaster DAX di Wilayah tersebut.

Setiap Wilayah dirancang untuk diisolasi dari Wilayah lainnya. Di dalam setiap Wilayah adalah beberapa Availability Zone. Dengan meluncurkan simpul Anda di Availability Zone yang berbeda, Anda dapat mencapai toleransi kesalahan terbesar sebisa mungkin.

Important

Jangan menempatkan semua simpul klaster Anda di Availability Zone tunggal. Dalam konfigurasi ini, klaster DAX Anda menjadi tidak tersedia jika ada kegagalan Availability Zone. Untuk penggunaan produksi, kami sangat menyarankan menggunakan DAX dengan setidaknya tiga simpul, di mana setiap simpul ditempatkan di Availability Zone yang berbeda. Tiga simpul diperlukan untuk klaster DAX menjadi toleran terhadap kesalahan. Satu klaster DAX dapat digunakan dengan satu atau dua simpul untuk developeran atau beban kerja pengujian. Klaster dengan satu dan dua simpul tidak toleran terhadap kesalahan, dan kami tidak merekomendasikan penggunaan kurang dari tiga simpul untuk penggunaan produksi. Jika satu atau dua simpul klaster mengalami kesalahan perangkat lunak atau perangkat keras, klaster dapat menjadi tidak tersedia atau kehilangan data cache.

Grup parameter

Grup parameter digunakan untuk mengelola pengaturan waktu aktif untuk klaster DAX. DAX memiliki beberapa parameter yang dapat Anda gunakan untuk mengoptimalkan performa (seperti mendefinisikan kebijakan TTL untuk data cache). Sebuah grup parameter adalah set bernama parameter yang dapat Anda terapkan untuk klaster. Anda dapat dengan demikian memastikan bahwa semua simpul dalam klaster yang dikonfigurasi dengan cara yang persis sama.

Grup keamanan

klaster DAX berjalan di lingkungan Amazon Virtual Private Cloud (Amazon VPC). Lingkungan ini adalah jaringan virtual yang didedikasikan untuk akun AWS Anda dan diisolasi dari VPC lainnya. grup keamanan bertindak sebagai firewall virtual untuk VPC Anda guna mengontrol lalu lintas masuk dan keluar.

Ketika Anda meluncurkan sebuah klaster di VPC Anda, Anda menambahkan masuk ke grup keamanan Anda untuk mengizinkan lalu lintas jaringan masuk. Aturan masuknya menentukan

protokol (TCP) dan nomor port (8111) untuk klaster Anda. Setelah Anda menambahkan aturan ini ke grup keamanan Anda, aplikasi yang berjalan dalam VPC Anda dapat mengakses klaster DAX.

klaster ARN

Setiap klaster DAX ditugaskan Amazon Resource Name (ARN). Format ARN adalah sebagai berikut.

```
arn:aws:dax:region:accountID:cache/clusterName
```

Anda menggunakan klaster ARN dalam kebijakan IAM untuk menentukan izin untuk operasi API DAX. Untuk informasi selengkapnya, lihat [Kontrol akses DAX](#).

Titik akhir klaster

Setiap klaster DAX menyediakan klaster titik akhir untuk digunakan oleh aplikasi Anda. Dengan mengakses klaster menggunakan titik akhir, aplikasi Anda tidak perlu tahu hostname dan nomor port simpul individu dalam klaster. Aplikasi Anda secara otomatis "knows" semua simpul di klaster, bahkan jika Anda menambahkan atau menghapus replika baca.

Berikut ini adalah contoh dari titik akhir klaster di wilayah us-east-1 yang tidak dikonfigurasi untuk menggunakan enkripsi transit.

```
dax://my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com
```

Berikut ini adalah contoh dari titik akhir klaster di wilayah yang sama yang dikonfigurasi untuk menggunakan enkripsi dalam transit.

```
daxs://my-encrypted-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com
```

Titik akhir simpul

Masing-masing simpul individu dalam sebuah klaster DAX memiliki sendiri hostname dan nomor port. Berikut ini adalah contoh titik akhir simpul.

```
myDAXcluster-a.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111
```

Aplikasi Anda dapat mengakses simpul langsung dengan menggunakan titik akhirnya. Namun, kami merekomendasikan bahwa Anda memperlakukan klaster DAX sebagai satu unit dan mengaksesnya menggunakan titik akhir klaster sebagai gantinya. Klaster titik akhir menginsulasi aplikasi Anda dari harus mempertahankan daftar simpul dan menjaga daftar terbaru ketika Anda menambahkan atau menghapus simpul dari klaster.

Grup subnet

Akses ke simpul klaster DAX dibatasi untuk aplikasi yang berjalan di instans Amazon EC2 dalam lingkungan Amazon VPC. Anda dapat menggunakan grup subnet untuk memberikan akses klaster dari instans Amazon EC2 berjalan pada subnet tertentu. Grup subnet adalah kumpulan subnet (biasanya private) yang dapat Anda tetapkan untuk klaster Anda agar berjalan di lingkungan Amazon VPC.

Ketika Anda membuat sebuah klaster DAX, Anda harus menentukan grup subnet. DAX menggunakan grup subnet tersebut untuk memilih subnet dan alamat IP dalam subnet tersebut untuk mengasosiasikan dengan simpul Anda.

Kejadian

DAX mencatat peristiwa penting dalam klaster Anda, seperti kegagalan untuk menambahkan simpul, berhasil dalam menambahkan simpul, atau perubahan pada grup keamanan. Dengan memantau peristiwa penting, Anda dapat mengetahui keadaan atau kondisi klaster Anda dan dapat mengambil tindakan korektif sebelum terjadinya peristiwa. Anda dapat mengakses peristiwa-peristiwa ini menggunakan AWS Management Console atau `DescribeEvents` dalam API manajemen DAX.

Anda juga dapat meminta pemberitahuan yang dikirim ke topik Amazon Simple Notification Service (Amazon SNS). Kemudian Anda akan tahu segera ketika sebuah peristiwa terjadi di klaster DAX Anda.

Jendela pemeliharaan

Setiap klaster memiliki waktu pemeliharaan mingguan di mana setiap perubahan sistem diterapkan. Jika Anda tidak menentukan waktu pemeliharaan yang diinginkan saat Anda membuat atau mengubah klaster cache, maka DAX menetapkan jendela pemeliharaan 60 menit pada hari yang dipilih secara acak dalam seminggu.

Jendela pemeliharaan selama 60 menit dipilih secara acak dari blok waktu 8 jam per Wilayah AWS. Daftar tabel berikut mencantumkan blok waktu untuk setiap Wilayah tempat jendela pemeliharaan default ditetapkan.

Kode Wilayah	Nama wilayah	Jendela pemeliharaan
ap-northeast-1	Wilayah Asia Pacific (Tokyo)	13:00–21:00 UTC

Kode Wilayah	Nama wilayah	Jendela pemeliharaan
ap-southeast-1	Wilayah Asia Pacific (Singapore)	14:00–22:00 UTC
ap-southeast-2	Wilayah Asia Pacific (Sydney)	12:00–20:00 UTC
ap-south-1	Wilayah Asia Pacific (Mumbai)	17:30–1:30 UTC
cn-northwest-1	Wilayah China (Ningxia)	23:00–07:00 UTC
cn-north-1	Wilayah China (Beijing)	14:00–22:00 UTC
eu-central-1	Wilayah Eropa (Frankfurt)	23:00–07:00 UTC
eu-west-1	Wilayah Eropa (Irlandia)	22:00–06:00 UTC
eu-west-2	Wilayah Eropa (London)	23:00–07:00 UTC
eu-west-3	Wilayah Eropa (Paris)	23:00–07:00 UTC
sa-east-1	Wilayah South America (São Paulo)	01:00–09:00 UTC
us-east-1	Wilayah US East (N. Virginia)	03:00–11:00 UTC
us-east-2	Wilayah US East (Ohio)	23:00–07:00 UTC
us-west-1	Wilayah Barat AS (N. California)	06:00–14:00 UTC
us-west-2	Wilayah US West (Oregon)	06:00–14:00 UTC

Jendela pemeliharaan harus berada pada waktu penggunaan terendah dan mungkin memerlukan ubahan dari waktu ke waktu. Anda dapat menentukan rentang waktu hingga durasi 24 jam selama kegiatan pemeliharaan yang Anda minta harus terjadi.

Membuat kluster DAX

Bagian ini memandu Anda melalui setup pertama kali dan penggunaan Amazon DynamoDB Accelerator (DAX) di lingkungan default Amazon Virtual Private Cloud (Amazon VPC). Anda dapat membuat kluster DAX pertama Anda menggunakan AWS Command Line Interface (AWS CLI) atau AWS Management Console.

Setelah Anda membuat kluster DAX Anda, Anda dapat mengaksesnya dari instans Amazon EC2 yang berjalan di VPC yang sama. Anda kemudian dapat menggunakan kluster DAX Anda dengan program aplikasi. Untuk informasi selengkapnya, lihat [Mengembangkan dengan klien DynamoDB Accelerator \(DAX\)](#).

Topik

- [Membuat peran layanan IAM untuk DAX untuk mengakses DynamoDB](#)
- [Membuat sebuah grup DAX menggunakan AWS CLI](#)
- [Membuat kluster DAX kluster menggunakan AWS Management Console](#)

Membuat peran layanan IAM untuk DAX untuk mengakses DynamoDB

Agar kluster DAX Anda mengakses tabel DynamoDB atas nama Anda, Anda harus membuat peran layanan. Peran layanan adalah AWS Identity and Access Management (IAM) role yang mengotorisasi layanan AWS untuk bertindak atas nama Anda. Peran layanan memungkinkan DAX untuk mengakses tabel DynamoDB Anda, seolah-olah Anda sendiri yang mengakses tabel tersebut. Anda harus membuat peran layanan sebelum Anda dapat membuat kluster DAX.

Jika Anda menggunakan konsol, alur kerja untuk membuat pemeriksaan kluster dalam memeriksa kehadiran peran layanan DAX yang sudah ada sebelumnya. Jika tidak ditemukan, konsol membuat peran layanan baru untuk Anda. Untuk informasi selengkapnya, lihat [the section called “Langkah 2: Buat kluster DAX kluster DAX”](#).

Jika Anda menggunakan AWS CLI, Anda harus menentukan peran layanan DAX yang telah Anda buat sebelumnya. Jika tidak, Anda harus membuat peran layanan baru sebelumnya. Untuk informasi selengkapnya, lihat [Langkah 1: Membuat peran layanan IAM untuk DAX untuk DAX untuk mengakses DynamoDB menggunakan AWS CLI](#).

Izin yang diperlukan untuk membuat peran layanan

Kebijakan yang dikelola AWS AdministratorAccess menyediakan semua izin yang diperlukan untuk membuat kluster DAX dan peran layanan. Jika pengguna Anda telah AdministratorAccess terlampir, tidak ada tindakan lebih lanjut yang diperlukan.

Jika tidak, Anda harus menambahkan izin berikut untuk kebijakan IAM Anda sehingga pengguna Anda dapat membuat peran layanan:

- iam:CreateRole
- iam:CreatePolicy
- iam:AttachRolePolicy
- iam:PassRole

Melampirkan izin ini ke pengguna yang mencoba untuk melakukan tindakan.

Note

Izin iam:CreateRole, iam:CreatePolicy, iam:AttachRolePolicy, dan iam:PassRole tidak termasuk dalam AWS kebijakan yang dikelola untuk DynamoDB. Ini adalah dengan desain karena izin ini memberikan kemungkinan eskalasi hak istimewa: Artinya, pengguna dapat menggunakan izin ini untuk membuat kebijakan administrator baru dan kemudian melampirkan kebijakan tersebut untuk peran yang ada. Untuk alasan ini, Anda (administrator kluster DAX Anda) harus secara eksplisit menambahkan izin ini ke kebijakan Anda.

Pemecahan Masalah

Jika kebijakan pengguna Anda kehilangan izin iam:CreateRole, iam:CreatePolicy, dan iam:AttachPolicy, Anda akan mendapatkan pesan kesalahan. Tabel berikut mencantumkan pesan ini dan menjelaskan cara memperbaiki masalah.

Jika Anda melihat pesan kesalahan ini...	Lakukan hal berikut:
User: arn:aws:iam:: <i>accountID</i> :user/ <i>userName</i> is not authorized to perform: iam:CreateRole on	Tambahkan iam:CreateRole ke kebijakan pengguna Anda.

<p>Jika Anda melihat pesan kesalahan ini...</p> <pre>resource: arn:aws:iam:: <i>accountID</i> :role/service-role/ <i>roleName</i></pre>	<p>Lakukan hal berikut:</p>
<pre>User: arn:aws:iam:: <i>accountID</i> :user/<i>userName</i> is not authorize d to perform: iam:CreatePolicy on resource: policy <i>policyName</i></pre>	<p>Tambahkan iam:CreatePolicy ke kebijakan pengguna Anda.</p>
<pre>User: arn:aws:iam:: <i>accountID</i> :user/<i>userName</i> is not authorized to perform: iam:AttachRolePolicy on resource: role <i>daxServiceRole</i></pre>	<p>Tambahkan iam:AttachRolePolicy ke kebijakan pengguna Anda.</p>

Untuk informasi selengkapnya tentang kebijakan IAM yang diperlukan untuk administrasi kluster DAX, lihat [Kontrol akses DAX](#).

Membuat sebuah grup DAX menggunakan AWS CLI

Bagian ini menjelaskan cara membuat kluster Amazon DynamoDB Accelerator (DAX) menggunakan AWS Command Line Interface (AWS CLI). Jika Anda belum melakukannya, Anda harus menginstal dan mengonfigurasi AWS CLI. Untuk melakukannya, lihat petunjuk berikut di AWS Command Line Interface Panduan Pengguna:

- [Menginstal AWS CLI](#)
- [Mengkonfigurasi AWS CLI](#)

Important

Untuk mengelola kluster DAX menggunakan AWS CLI, pasang atau tingkatkan ke versi 1.11.110 atau lebih tinggi.

Semua contoh AWS CLI menggunakan Wilayah us-west-2 dan ID akun fiktif.

Topik

- [Langkah 1: Membuat peran layanan IAM untuk DAX untuk mengakses DynamoDB menggunakan AWS CLI](#)
- [Langkah 2: Membuat sebuah grup subnet](#)
- [Langkah 3: Membuat sebuah grup DAX menggunakan AWS CLI](#)
- [Langkah 4: Konfigurasi aturan masuk grup keamanan menggunakan AWS CLI](#)

Langkah 1: Membuat peran layanan IAM untuk DAX untuk mengakses DynamoDB menggunakan AWS CLI

Sebelum Anda dapat membuat kluster Amazon DynamoDB Accelerator (DAX), Anda harus membuat peran layanan untuk itu. Peran layanan adalah AWS Identity and Access Management (IAM) role yang mengotorisasi layanan AWS untuk bertindak atas nama Anda. Peran layanan memungkinkan DAX untuk mengakses tabel DynamoDB Anda, seolah-olah Anda sendiri yang mengakses tabel tersebut.

Pada langkah ini, Anda membuat kebijakan IAM dan kemudian melampirkan kebijakan tersebut ke IAM role. Hal ini memungkinkan Anda untuk menetapkan peran untuk kluster DAX sehingga dapat melakukan operasi DynamoDB atas nama Anda.

Untuk membuat peran layanan IAM untuk DAX

1. Buat file bernama `service-trust-relationship.json` dengan konten berikut.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "dax.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Buat peran layanan.

```
aws iam create-role \
```

```
--role-name DAXServiceRoleForDynamoDBAccess \  
--assume-role-policy-document file://service-trust-relationship.json
```

3. Buat file bernama `service-role-policy.json` dengan konten berikut.

```
{  
  "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:accountID:"  
      ]  
    }  
  ]  
}
```

Ganti *ID akun* dengan ID akun AWS. Untuk menemukan AWS ID akun Anda, di sudut kanan atas konsol, pilih ID masuk Anda. ID akun AWS Anda muncul di menu tarik-turun.

Pada Amazon Resource Name (ARN) yang ada di contoh, *ID akun* harus berupa angka 12 digit. Jangan gunakan tanda hubung atau tanda baca lainnya.

4. Buat kebijakan IAM untuk peran layanan.

```
aws iam create-policy \  
  --policy-name DAXServicePolicyForDynamoDBAccess \  
  --policy-document file://service-role-policy.json
```

Dalam output, perhatikan ARN untuk kebijakan yang Anda buat, seperti dalam contoh berikut.


```
arn:aws:iam::123456789012:policy/DAXServicePolicyForDynamoDBAccess
```

5. Lampirkan kebijakan untuk peran layanan. Ganti *arn* di kode berikut dengan peran ARN yang sebenarnya dari langkah sebelumnya.

```
aws iam attach-role-policy \  
  --role-name DAXServiceRoleForDynamoDBAccess \  
  --policy-arn arn
```

Selanjutnya, Anda menentukan grup subnet untuk VPC default Anda. Grup subnet adalah kumpulan satu subnet atau lebih di VPC Anda. Lihat [Langkah 2: Membuat sebuah grup subnet](#).

Langkah 2: Membuat sebuah grup subnet

Ikuti prosedur ini untuk membuat grup subnet untuk kluster Amazon DynamoDB Accelerator (DAX) Anda menggunakan AWS Command Line Interface (AWS CLI).

Note

Jika Anda sudah membuat grup subnet untuk VPC default Anda, Anda dapat melewati langkah ini.

DAX dirancang untuk dijalankan di lingkungan Amazon Virtual Private Cloud (Amazon VPC). Jika Anda membuat akun AWS setelah 4 Desember 2013, Anda sudah memiliki VPC default di setiap Wilayah AWS. Untuk informasi selengkapnya, lihat [VPC default dan subnet default](#) di Panduan Pengguna Amazon VPC.

Untuk membuat grup subnet

1. Untuk menentukan pengenal VPC default Anda, masukkan perintah berikut.

```
aws ec2 describe-vpcs
```

Dalam output, perhatikan pengenal untuk VPC default Anda, seperti dalam contoh berikut.

```
vpc-12345678
```

2. Menentukan ID subnet yang terkait dengan VPC default Anda. Ganti *IDvpc* dengan ID VPC Anda yang sebenarnya, misalnya, vpc-12345678.

```
aws ec2 describe-subnets \  
  --filters "Name=vpc-id,Values=vpcID" \  
  --query "Subnets[*].SubnetId"
```

Di output, catat pengenal subnet, misalnya, subnet-11111111.

3. Buat grup subnet. Pastikan bahwa Anda menentukan setidaknya satu ID subnet di parameter `--subnet-ids`.

```
aws dax create-subnet-group \  
  --subnet-group-name my-subnet-group \  
  --subnet-ids subnet-11111111 subnet-22222222 subnet-33333333 subnet-44444444
```

Untuk membuat klaster, lihat [Langkah 3: Membuat sebuah grup DAX menggunakan AWS CLI](#).

Langkah 3: Membuat sebuah grup DAX menggunakan AWS CLI

Ikuti prosedur ini untuk menggunakan AWS Command Line Interface (AWS CLI) untuk membuat klaster Amazon DynamoDB Accelerator (DAX) di Amazon VPC default.

Untuk membuat klaster DAX

1. Dapatkan Amazon Resource Name (ARN) untuk peran layanan Anda.

```
aws iam get-role \  
  --role-name DAXServiceRoleForDynamoDBAccess \  
  --query "Role.Arn" --output text
```

Dalam output, perhatikan peran layanan ARN, seperti dalam contoh berikut.

```
arn:aws:iam::123456789012:role/DAXServiceRoleForDynamoDBAccess
```

2. Buat klaster DAX. Ganti *roleARN* dengan ARN dari langkah sebelumnya.

```
aws dax create-cluster \  
  --cluster-name mydaxcluster \  
  --node-type dax.r4.large \  
  --replication-factor 3 \  
  --role-arn roleARN
```

```
--iam-role-arn roleARN \  
--subnet-group my-subnet-group \  
--sse-specification Enabled=true \  
--region us-west-2
```

Semua simpul dalam kluster adalah tipe `dax.r4.large` (`--node-type`). Ada tiga simpul (`--replication-factor`) satu simpul primer dan dua replika.

Note

Sejak `sudo` dan `grep` adalah kata kunci cadangan, Anda tidak dapat membuat kluster DAX dengan kata-kata ini dalam nama kluster. Misalnya, `sudo` dan `sudocluster` adalah nama kluster yang tidak valid.

Untuk melihat status kluster, masukkan perintah berikut.

```
aws dax describe-clusters
```

Status ditampilkan dalam output, misalnya, `"Status": "creating"`.

Note

Membuat kluster memerlukan waktu beberapa menit. Ketika kluster siap, statusnya berubah menjadi `available`. Sementara itu, lanjutkan ke [Langkah 4: Konfigurasi aturan masuk grup keamanan menggunakan AWS CLI](#) dan ikuti arahan di sana.

Langkah 4: Konfigurasi aturan masuk grup keamanan menggunakan AWS CLI

Simpul di kluster Amazon DynamoDB Accelerator (DAX) Anda menggunakan grup keamanan default untuk Amazon VPC Anda. Untuk grup keamanan default, Anda harus mengotorisasi lalu lintas masuk pada TCP port 8111 untuk kluster terenkripsi atau port 9111 untuk kluster terenkripsi. Hal ini memungkinkan instans Amazon EC2 di Amazon VPC Anda untuk mengakses kluster DAX Anda.

Note

Jika Anda meluncurkan klaster DAX Anda dengan grup keamanan yang berbeda (selain default), Anda harus melakukan prosedur ini untuk grup itu sebagai gantinya.

Untuk mengkonfigurasi aturan inbound grup keamanan

1. Masukkan perintah berikut untuk mendapatkan ID grup keamanan default. Ganti *vpcID* dengan ID VPC Anda yang sebenarnya (dari [Langkah 2: Membuat sebuah grup subnet](#)).

```
aws ec2 describe-security-groups \
  --filters Name=vpc-id,Values=vpcID Name=group-name,Values=default \
  --query "SecurityGroups[*].{GroupName:GroupName,GroupId:GroupId}"
```

Pada output, catat pengenal grup keamanan, misalnya, sg-01234567.

2. Kemudian masukkan perintah berikut. Ganti *sgID* dengan pengidentifikasi grup keamanan yang sebenarnya. Gunakan port 8111 untuk klaster yang tidak terenkripsi dan 9111 untuk klaster terenkripsi.

```
aws ec2 authorize-security-group-ingress \
  --group-id sgID --protocol tcp --port 8111
```

Membuat klaster DAX klaster menggunakan AWS Management Console

Bagian ini menjelaskan cara membuat klaster Amazon DynamoDB Accelerator (DAX) menggunakan AWS Management Console.

Topik

- [Langkah 1: Buat grup subnet menggunakan grup subnet AWS Management Console](#)
- [Langkah 2: Buat klaster DAX klaster DAX AWS Management Console](#)
- [Langkah 3: Mengkonfigurasi aturan masuk grup keamanan menggunakan aturan masuk grup keamanan AWS Management Console](#)

Langkah 1: Buat grup subnet menggunakan grup subnet AWS Management Console

Ikuti prosedur ini untuk membuat grup subnet untuk klaster Amazon DynamoDB Accelerator (DAX) Anda menggunakan AWS Management Console.

Note

Jika Anda sudah membuat grup subnet untuk VPC default Anda, Anda dapat melewati langkah ini.

DAX dirancang untuk dijalankan di lingkungan Amazon Virtual Private Cloud (Amazon VPC). Jika Anda membuat akun AWS setelah 4 Desember 2013, Anda sudah memiliki VPC default di setiap Wilayah AWS. Untuk informasi selengkapnya, lihat [VPC default dan subnet default](#) di Panduan Pengguna Amazon VPC.

Sebagai bagian dari proses pembuatan klaster DAX, Anda harus menentukan grup subnet. Grup subnet adalah kumpulan satu subnet atau lebih di VPC Anda. Ketika Anda membuat klaster DAX Anda, simpul disebar ke subnet dalam grup subnet.

Untuk membuat grup subnet

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Dalam panel navigasi, di bawah DAX, pilih Grup subnet.
3. Pilih Buat grup subnet.
4. Di jendela buat grup subnet, lakukan hal berikut:
 - a. Nama — Masukkan nama pendek untuk grup subnet.
 - b. Deskripsi — Masukkan deskripsi untuk grup subnet.
 - c. ID VPC — Pilih pengenal untuk lingkungan Amazon VPC Anda.
 - d. Subnet — Pilih satu subnet atau lebih dari daftar.

Note

Subnet didistribusikan di antara beberapa Availability Zone. Jika Anda berencana untuk membuat mult simpul klaster DAX (simpul primer dan satu replika pembacaan atau lebih), kami sarankan Anda memilih beberapa subnet ID. Kemudian DAX dapat men-deploy simpul klaster ke beberapa Availability Zone. Jika Availability Zone

menjadi tidak tersedia, DAX secara otomatis gagal atas Availability Zone yang masih bertahan. kluster DAX Anda akan terus berfungsi tanpa gangguan.

Saat pengaturan sesuai keinginan Anda, pilih Buat grup subnet.

Untuk membuat kluster, lihat [Langkah 2: Buat kluster DAX kluster DAXAWS Management Console](#).

Langkah 2: Buat kluster DAX kluster DAXAWS Management Console

Ikuti prosedur ini untuk membuat sebuah kluster Amazon DynamoDB Accelerator (DAX) di default Amazon VPC Anda.

Untuk membuat kluster DAX

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi, di bawah DAX, pilih kluster.
3. Pilih Buat kluster.
4. Di jendela Buat kluster, lakukan hal berikut:
 - a. Nama kluster — Masukkan nama pendek untuk kluster DAX Anda.

Note


Sejak `sudo` dan `grep` adalah kata kunci cadangan, Anda tidak dapat membuat kluster DAX dengan kata-kata ini dalam nama kluster. Misalnya, `sudo` dan `sudocluster` adalah nama kluster yang tidak valid.

- b. Deskripsi kluster — Masukkan deskripsi untuk kluster.
- c. Jenis simpul—Pilih tipe simpul untuk semua simpul di kluster.
- d. Ukuran kluster — Pilih jumlah simpul di kluster. Sebuah kluster terdiri dari satu simpul primer dan hingga sembilan replika pembacaan.

Note

Jika Anda ingin membuat kluster simpul tunggal, pilih 1. kluster Anda akan terdiri dari satu simpul primer.

Jika Anda ingin membuat kluster multi simpul, pilih nomor antara 3 (satu sebagai yang utama dan dua replika pembacaan) dan 10 (satu yang utama dan sembilan replika pembacaan).

 Important

Untuk penggunaan produksi, kami sangat menyarankan menggunakan DAX dengan setidaknya tiga simpul, di mana setiap simpul ditempatkan di Availability Zone yang berbeda. Tiga simpul diperlukan untuk kluster DAX menjadi toleran terhadap kesalahan.

Satu kluster DAX dapat digunakan dengan satu atau dua simpul untuk developeran atau beban kerja pengujian. Kluster dengan satu dan dua simpul tidak toleran terhadap kesalahan, dan kami tidak merekomendasikan penggunaan kurang dari tiga simpul untuk penggunaan produksi. Jika satu atau dua simpul kluster mengalami kesalahan perangkat lunak atau perangkat keras, kluster dapat menjadi tidak tersedia atau kehilangan data cache.

- e. Pilih Selanjutnya.
- f. Grup subnet—PilihPilih yang adadan pilih grup subnet yang Anda buat di [Langkah 1: Buat grup subnet menggunakan grup subnetAWS Management Console](#).
- g. Kontrol akses—Pilihdefaultgrup keamanan.
- h. Zona Ketersediaan (AZ)—PilihOtomatis.
- i. Pilih berikutnya.
- j. Peran layanan IAM untuk akses DynamoDB — Pilih Buat baru, dan masukkan informasi berikut:
 - Nama IAM role — Masukkan nama untuk IAM role, misalnya, DAXServiceRole. Konsol menciptakan IAM role baru, dan kluster DAX Anda mengasumsikan peran ini pada saat waktu proses.
 - Pilih kotak di sampingBuat kebijakan.
 - Kebijakan IAM role — Pilih Baca/Tulis. Hal ini memungkinkan kluster DAX untuk melakukan operasi pembacaan dan penulisan di DynamoDB.
 - Nama kebijakan IAM baru—Bidang ini akan terisi saat Anda memasukkan nama peran IAM. Anda juga dapat memasukkan nama untuk kebijakan IAM,

misalnya, `DAXServicePolicy`. Konsol membuat kebijakan IAM baru dan menempel kebijakan untuk IAM role.

- Akses ke tabel DynamoDB Tabel DynamoDB Accelerator—Pilih Semua tabel.
- k. Enkripsi—Pilih Aktifkan enkripsi saat istirahat dan Aktifkan enkripsi saat transit Untuk informasi selengkapnya, lihat [Enkripsi DAX saat istirahat](#) dan [Enkripsi DAX dalam transit](#).

Peran layanan terpisah untuk DAX dalam mengakses Amazon EC2 juga diperlukan. DAX secara otomatis membuat peran layanan ini untuk Anda. Untuk informasi selengkapnya, lihat [Menggunakan peran terkait layanan untuk DAX](#).

5. Ketika pengaturan seperti yang Anda inginkan, pilih Berikutnya.
6. Grup parameter—Pilih Pilih yang ada.
7. Jendela pemeliharaan—Pilih Tidak ada preferensi jika Anda tidak memiliki preferensi saat upgrade perangkat lunak diterapkan, atau pilih Tentukan jendela waktu dan menyediakan Hari kerja, Waktu (UTC) dan Mulai dalam (jam) pilihan untuk menjadwalkan jendela pemeliharaan Anda.
8. Tag—Pilih Tambahkan tag baru untuk memasukkan pasangan kunci/nilai untuk tujuan penandaan.
9. Pilih Selanjutnya.

Pada Memeriksa dan membuat layar, Anda dapat meninjau semua pengaturan. Jika Anda siap untuk membuat kluster, pilih Buat kluster.

Pada layar kluster, kluster DAX Anda akan terdaftar dengan status Membuat.

Note

Membuat kluster memerlukan waktu beberapa menit. Ketika kluster siap, statusnya berubah menjadi Tersedia.

Sementara itu, lanjutkan ke [Langkah 3: Mengkonfigurasi aturan masuk grup keamanan menggunakan aturan masuk grup keamanan AWS Management Console](#) dan ikuti arahan di sana.

Langkah 3: Mengkonfigurasi aturan masuk grup keamanan menggunakan aturan masuk grup keamanan AWS Management Console

kluster Amazon DynamoDB Accelerator (DAX) berkomunikasi melalui TCP port 8111 (untuk kluster tidak terenkripsi) atau 9111 (untuk kluster terenkripsi), sehingga Anda harus mengotorisasi lalu lintas masuk pada port tersebut. Hal ini memungkinkan instans Amazon EC2 di Amazon VPC Anda untuk mengakses kluster DAX Anda.

Note

Jika Anda meluncurkan kluster DAX Anda dengan grup keamanan yang berbeda (selain default), Anda harus melakukan prosedur ini untuk grup itu sebagai gantinya.

Untuk mengkonfigurasi aturan inbound grup keamanan

1. Buka konsol Amazon EC2 di <https://console.aws.amazon.com/ec2/>.
2. Di panel navigasi, pilih Grup Keamanan.
3. Pilih grup keamanan default. Pada menu Tindakan, pilih Edit aturan inbound.
4. Pilih Tambahkan Aturan, dan masukkan informasi berikut:
 - Rentang Port — Masukkan 8111 (jika kluster Anda tidak terenkripsi) atau 9111 (jika kluster Anda dienkripsi).
 - Sumber-Biarkan ini sebagai Kustom, dan pilih bidang pencarian di sebelah kanan. Menu drop-down akan ditampilkan. Pilih pengidentifikasi untuk grup keamanan default.
5. Pilih Simpan aturan untuk menyimpan perubahan Anda.
6. Untuk memperbarui nama di konsol, buka Nama properti dan pilih Sunting pilihan yang ditampilkan.

Model konsistensi DAX dan DynamoDB

Amazon DynamoDB Accelerator (DAX) adalah layanan caching write-through yang dirancang untuk menyederhanakan proses menambahkan cache ke tabel DynamoDB. Karena DAX beroperasi secara terpisah dari DynamoDB, penting bahwa Anda harus memahami konsistensi model DAX dan DynamoDB untuk memastikan bahwa aplikasi Anda berfungsi seperti yang Anda harapkan.

Dalam banyak kasus penggunaan, cara aplikasi Anda menggunakan DAX mempengaruhi konsistensi data dalam kluster DAX, dan konsistensi data antara DAX dan DynamoDB.

Topik

- [Konsistensi di antara node cluster DAX](#)
- [perilaku cache item DAX](#)
- [perilaku cache kueri DAX](#)
- [Bacaan yang sangat konsisten dan transaksional](#)
- [Caching negatif](#)
- [Strategi untuk Penulis](#)

Konsistensi di antara node cluster DAX

Untuk mencapai ketersediaan tinggi untuk aplikasi Anda, kami sarankan Anda menyediakan kluster DAX Anda dengan setidaknya tiga simpul. Kemudian tempatkan simpul tersebut di beberapa Availability Zone dalam Wilayah.

Ketika kluster DAX Anda sedang dijalankan, kluster itu mereplikasi data di antara semua simpul dalam kluster (dengan asumsi bahwa Anda menetapkan lebih dari satu simpul). Pertimbangkan sebuah aplikasi yang melakukan `UpdateItem` dengan berhasil menggunakan DAX. Tindakan ini menyebabkan cache item di simpul primer untuk dimodifikasi dengan nilai baru. Nilai itu kemudian direplikasi ke semua simpul lain dalam kluster. Replikasi ini akhirnya konsisten dan biasanya membutuhkan waktu kurang dari satu detik untuk menyelesaikan.

Dalam skenario ini, mungkin untuk dua klien dalam membaca kunci yang sama dari kluster DAX yang sama tetapi menerima nilai yang berbeda, tergantung pada simpul yang diakses setiap klien. Simpul semua konsisten ketika update telah sepenuhnya direplikasi di seluruh semua simpul dalam kluster. (Perilaku ini mirip dengan sifat konsisten DynamoDB.)

Jika Anda sedang membangun sebuah aplikasi yang menggunakan DAX, aplikasi itu harus dirancang sehingga dapat mentolerir data konsisten.

perilaku cache item DAX

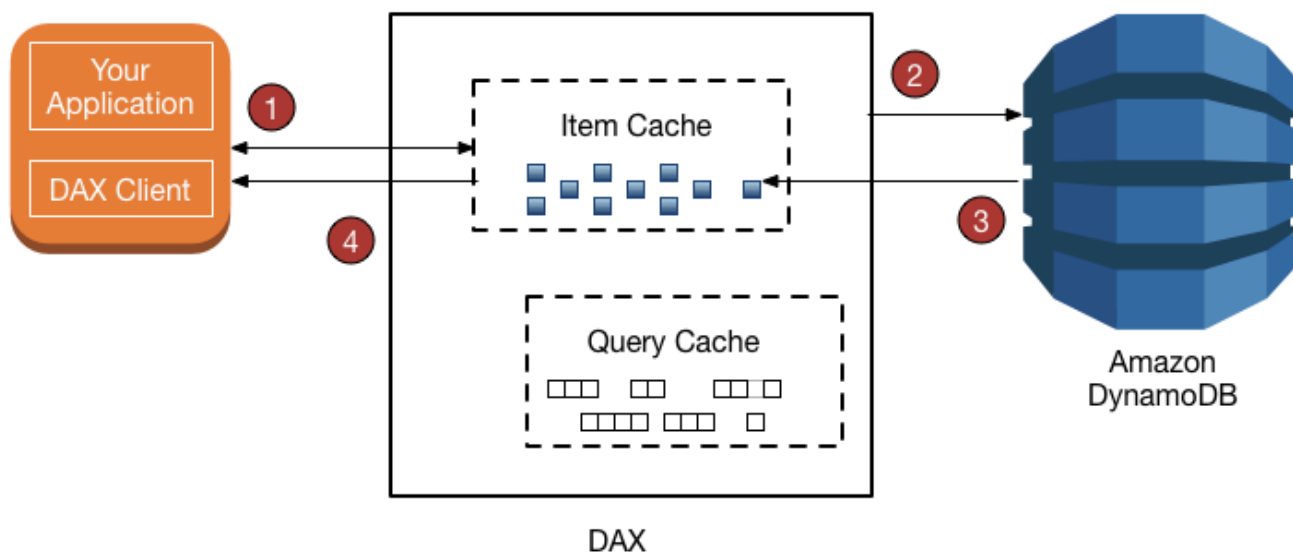
Setiap kluster DAX memiliki dua cache yang berbeda, cache item dan cache kueri. Untuk informasi selengkapnya, lihat [DAX: Cara kerjanya](#).

Bagian ini membahas implikasi konsistensi dari pembacaan dari dan menulis ke cache item DAX.

Konsistensi Pembacaan

Dengan DynamoDB, operasi `GetItem` melakukan bacaan akhir konsisten secara default. Misalkan Anda menggunakan `UpdateItem` dengan klien DynamoDB. Jika Anda kemudian mencoba untuk membaca item yang sama segera setelah itu, Anda mungkin melihat data yang muncul sebelum pembaruan. Hal ini disebabkan penundaan propagasi di semua lokasi penyimpanan DynamoDB. Konsistensi biasanya tercapai dalam hitungan detik. Jadi jika Anda mencoba kembali membaca, Anda mungkin akan melihat item yang diperbarui.

Saat Anda menggunakan `GetItem` dengan klien DAX, operasi (dalam kasus ini, bacaan akhir konsisten) berjalan seperti yang ditunjukkan berikut ini.



1. Klien DAX mengeluarkan permintaan `GetItem`. DAX mencoba untuk membaca item yang diminta dari cache item. Jika item dalam cache (cache hit), DAX mengembalikannya ke aplikasi.
2. Jika item tidak tersedia (cache miss), DAX melakukan operasi `GetItem` konsisten terhadap DynamoDB.
3. DynamoDB mengembalikan item yang diminta, dan DAX menyimpannya dalam cache item.
4. DAX mengembalikan item ke aplikasi.
5. (Tidak ditampilkan) Jika kluster DAX berisi lebih dari satu simpul, item direplikasi ke semua simpul lain dalam kluster.

Item tetap dalam cache item DAX, tunduk pada pengaturan Waktu untuk Tayang (TTL) dan algoritme yang paling baru-baru ini digunakan (LRU) untuk cache. Untuk informasi selengkapnya, lihat [DAX: Cara kerjanya](#).

Namun, selama periode ini, DAX tidak membaca ulang item dari DynamoDB. Jika orang lain memperbarui item menggunakan klien DynamoDB, melewati DAX seluruhnya, permintaan `GetItem` menggunakan klien DAX menghasilkan hasil yang berbeda dari permintaan `GetItem` menggunakan klien DynamoDB. Dalam skenario ini, DAX dan DynamoDB memegang nilai-nilai yang tidak konsisten untuk bukti kunci yang sama sampai TTL untuk DAX item kedaluwarsa.

Jika aplikasi memodifikasi data dalam tabel DynamoDB dasar, melewati DAX, aplikasi perlu mengantisipasi dan mentolerir inkonsistensi data yang mungkin terjadi.

Note

Selain itu `GetItem`, klien DAX juga mendukung permintaan `BatchGetItem`. `BatchGetItem` pada dasarnya adalah pembungkus sekitar satu permintaan atau lebih `GetItem`, sehingga DAX memperlakukan masing-masing sebagai operasi `GetItem` individu.

Konsistensi Penulisan

DAX adalah cache write-through, yang menyederhanakan proses penjagaan cache item DAX agar konsisten dengan tabel DynamoDB dasar.

Klien DAX mendukung operasi API penulisan yang sama seperti DynamoDB (`PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem`, dan `TransactWriteItems`). Ketika Anda menggunakan operasi ini dengan klien DAX, item diubah dalam DAX dan DynamoDB. DAX memperbarui item dalam cache item, terlepas dari nilai TTL untuk item ini.

Misalnya, anggaplah Anda membuat permintaan `GetItem` dari klien DAX untuk membaca item dari tabel `ProductCatalog`. (Kunci partisinya adalah `Id`, dan tidak ada kunci lain seperti itu.) Anda mengambil item yang `Id` adalah `101`. Nilai `QuantityOnHand` untuk item itu adalah `42`. DAX menyimpan item dalam cache item dengan TTL tertentu. Untuk contoh ini, TTL diasumsikan 10 menit. Lalu, 3 menit kemudian, aplikasi lain menggunakan klien DAX untuk memperbarui item yang sama sehingga nilai `QuantityOnHand` sekarang adalah `41`. Dengan asumsi bahwa item tidak diperbarui lagi, pembacaan item yang sama berikutnya dalam 10 menit berikutnya akan mengembalikan nilai cache untuk `QuantityOnHand` (`41`).

Cara DAX Memproses Penulisan

DAX ditujukan untuk aplikasi yang memerlukan pembacaan performa tinggi. Sebagai cache write-through, DAX melewati penulisan Anda melalui DynamoDB secara serentak, kemudian secara otomatis dan secara tidak langsung mereplikasi pembaruan hasil untuk cache item Anda di semua simpul dalam kluster. Anda tidak perlu mengelola logika pembatalan cache karena DAX menanganinya untuk Anda.

DAX mendukung operasi penulisan berikut: `PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem`, dan `TransactWriteItems`.

Ketika Anda mengirim permintaan `PutItem`, `UpdateItem`, `DeleteItem`, atau `BatchWriteItem` untuk DAX, hal berikut akan terjadi:

- DAX mengirimkan permintaan ke DynamoDB.
- DynamoDB membalas DAX, mengkonfirmasi bahwa penulisan berhasil.
- DAX menulis item ke cache item.
- DAX mengembalikan status berhasil ke peminta.

Ketika Anda mengirim permintaan `TransactWriteItems` ke DAX, hal berikut akan terjadi:

- DAX mengirimkan permintaan ke DynamoDB.
- DynamoDB membalas DAX, mengkonfirmasi bahwa transaksi selesai.
- DAX mengembalikan status berhasil ke peminta.
- Di latar belakang, DAX membuat permintaan `TransactGetItems` untuk setiap item dalam permintaan `TransactWriteItems` untuk menyimpan item dalam cache item. `TransactGetItems` digunakan untuk memastikan [isolasi yang dapat diserialisasi](#).

Jika penulisan ke DynamoDB gagal dengan alasan apapun, termasuk throttling, item tidak di-cache di DAX. Pengecualian untuk kegagalan disampaikan ke peminta. Hal ini memastikan bahwa data tidak ditulis ke cache DAX kecuali ditulis terlebih dahulu dengan berhasil ke DynamoDB.

Note

Setiap penulisan ke DAX akan mengubah status cache item. Namun, menulis ke cache item tidak mempengaruhi cache kueri. (cache item DAX dan cache kueri melayani tujuan yang berbeda, dan beroperasi secara independen dari satu sama lain.)

perilaku cache kueri DAX

DAX meng-cache hasil dari Query dan Scan permintaan dalam cache kueri. Namun, hasil ini tidak mempengaruhi cache item sama sekali. Saat aplikasi Anda meminta Query atau Scan dengan DAX, set hasil disimpan dalam cache kueri, tidak dalam cache item. Anda tidak dapat melakukan "pemanasan" cache item dengan melakukan operasi Scan karena cache item dan cache kueri entitas adalah terpisah.

Konsistensi query-update-query

Pembaruan untuk cache item, atau tabel DynamoDB dasar, tidak membatalkan atau merubah hasil yang disimpan dalam cache kueri.

Sebagai gambaran, pertimbangkan skenario berikut: Aplikasi bekerja dengan tabel DocumentRevisions, yang memiliki DocId sebagai kunci partisi dan RevisionNumber sebagai kunci.

1. Klien meminta Query untuk DocId 101, untuk semua item yang RevisionNumber lebih besar dari atau sama dengan 5. DAX menyimpan set hasil dalam cache kueri dan mengembalikan set hasil untuk pengguna.
2. Klien mengeluarkan permintaan PutItem untuk DocId 101 dengan RevisionNumber nilai 20.
3. Klien mengeluarkan Query yang sama seperti yang dijelaskan pada langkah 1 (DocId 101 dan RevisionNumber \geq 5).

Dalam skenario ini, set hasil yang di-cache untuk Query dikeluarkan pada langkah 3 adalah identik dengan hasil yang ditetapkan yang di-cache di langkah 1. Alasannya adalah bahwa DAX tidak membatalkan Query atau Scan set hasil berdasarkan pembaruan untuk masing-masing item. Operasi PutItem dari langkah 2 hanya tercermin dalam cache kueri DAX ketika TTL untuk Query kedaluwarsa.

Aplikasi Anda harus mempertimbangkan nilai TTL untuk cache kueri dan berapa lama aplikasi Anda dapat mentolerir hasil yang tidak konsisten antara cache kueri dan cache item.

Bacaan yang sangat konsisten dan transaksional

Untuk melakukan `GetItem`, `BatchGetItem`, `Query`, atau permintaan `Scan` yang sangat konsisten, Anda perlu mengatur parameter `ConsistentRead` agar tepat. DAX melewati pembacaan yang sangat konsisten ke DynamoDB. Ketika menerima respon dari DynamoDB, DAX mengembalikan hasil ke klien, tetapi bukan meng-cache hasil. DAX tidak dapat menyediakan pembacaan yang sangat konsisten dengan sendirinya karena itu tidak digabungkan dengan erat ke DynamoDB. Untuk alasan ini, setiap pembacaan berikutnya dari DAX pembacaannya harus konsisten. Dan setiap pembacaannya berikutnya akan sangat konsisten untuk melewati DynamoDB.

DAX menangani permintaan `TransactGetItems` dengan cara yang sama dalam menangani pembacaan yang sangat konsisten. DAX melewati semua permintaan `TransactGetItems` ke DynamoDB. Ketika menerima respon dari DynamoDB, DAX mengembalikan hasil ke klien, tetapi bukan meng-cache hasil.

Caching negatif

DAX mendukung entri cache negatif baik di cache item dan di cache kueri. Entri cache negatif terjadi ketika DAX tidak dapat menemukan item yang diminta dalam tabel DynamoDB dasar. Alih-alih menghasilkan kesalahan, DAX meng-cache hasil kosong dan mengembalikan hasil ke pengguna.

Sebagai contoh, misalkan aplikasi mengirimkan permintaan `GetItem` ke kluster DAX, dan tidak ada item yang cocok dalam cache item DAX. Hal ini menyebabkan DAX membaca item yang sesuai dari tabel DynamoDB dasar. Jika item tidak ada di DynamoDB, DAX menyimpan item kosong dalam cache item dan mengembalikan item kosong ke aplikasi. Sekarang anggaplah bahwa aplikasi mengirimkan permintaan `GetItem` lain untuk item yang sama. DAX menemukan item kosong dalam cache item dan mengembalikannya segera ke aplikasi. DAX tidak berunding dengan DynamoDB sama sekali.

Entri cache negatif tetap ada di cache item DAX sampai TTL itemnya kedaluwarsa, LRU-lah yang dipanggil, atau item diubah menggunakan `PutItem`, `UpdateItem`, atau `DeleteItem`.

Cache kueri DAX menangani hasil cache negatif dengan cara yang sama. Jika aplikasi melakukan `Query` atau `Scan`, dan cache kueri DAX tidak berisi hasil cache, DAX mengirimkan permintaan ke DynamoDB. Jika tidak ada item yang cocok dalam set hasil, DAX menyimpan set hasil

kosong di cache kueri dan mengembalikan set hasil kosong ke aplikasi. Permintaan Query atau Scan selanjutnya menghasilkan set hasil (kosong) yang sama, hingga TTL untuk hasil tersebut kedaluwarsa.

Strategi untuk Penulis

Perilaku write-through DAX sesuai untuk banyak pola aplikasi. Namun, ada beberapa pola aplikasi di mana model write-through mungkin tidak sesuai.

Untuk aplikasi yang sensitif terhadap latensi, menulis melalui DAX menimbulkan lompatan jaringan tambahan. Jadi menulis ke DAX adalah sedikit lebih lambat dibanding menulis langsung ke DynamoDB. Jika aplikasi Anda sensitif untuk menulis latensi, Anda dapat mengurangi latensi dengan menulis langsung ke DynamoDB sebagai gantinya. Untuk informasi selengkapnya, lihat [Write-Around](#).

Untuk aplikasi write-intensive (seperti yang melakukan pembebanan data massal), Anda mungkin tidak ingin menulis semua data melalui DAX karena hanya sebagian kecil dari data yang pernah dibaca oleh aplikasi. Ketika Anda menulis sejumlah besar data melalui DAX, maka DAX harus memanggil algoritme LRU untuk membuat ruang di cache untuk item baru untuk dibaca. Hal ini mengurangi efektivitas DAX sebagai cache pembacaan.

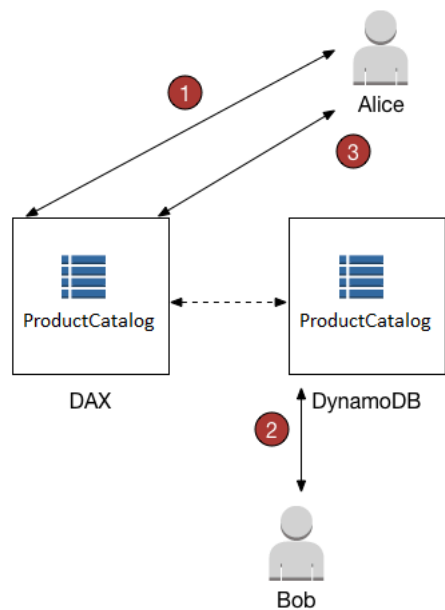
Ketika Anda menulis item ke DAX, status cache item diubah untuk mengakomodasi item baru. (Misalnya, DAX mungkin perlu mengeluarkan data lama dari cache item untuk memberi ruang bagi item baru.) Item baru tetap dalam cache item, tunduk pada algoritme LRU cache dan pengaturan TTL untuk cache. Selama item tetap dalam cache item, DAX tidak membaca ulang item dari DynamoDB.

Tulis-semua (write-through)

Cache item DAX menerapkan kebijakan write-through. Untuk informasi selengkapnya, lihat [Cara DAX Memproses Penulisan](#).

Ketika Anda menulis item, DAX memastikan bahwa item cache disinkronkan dengan item yang ada di DynamoDB. Ini sangat membantu untuk aplikasi yang perlu membaca ulang item segera setelah menulisnya. Namun, jika aplikasi lain menulis langsung ke tabel DynamoDB, item dalam cache item DAX tidak lagi sinkron dengan DynamoDB.

Untuk menggambarkan, pertimbangkan dua pengguna (Alice dan Bob), yang bekerja dengan tabel ProductCatalog. Alice mengakses tabel menggunakan DAX, tapi Bob melewati DAX dan mengakses tabel langsung di DynamoDB.



1. Alice memperbarui item di tabel `ProductCatalog`. DAX meneruskan permintaan untuk DynamoDB, dan pembaruan berhasil. DAX kemudian menulis item ke cache item dan mengembalikan respon yang berhasil ke Alice. Sejak saat itu, sampai item tersebut akhirnya dihapus dari cache, setiap pengguna yang membaca item dari DAX akan melihat item dengan pembaruan Alice.
2. Beberapa waktu kemudian, Bob memperbarui item `ProductCatalog` yang sama yang telah ditulis Alice. Namun, Bob memperbarui item langsung di DynamoDB. DAX tidak secara otomatis menyegarkan item cache dalam menanggapi pembaruan melalui DynamoDB. Oleh karena itu, pengguna DAX tidak melihat pembaruan Bob.
3. Alice membaca item dari DAX lagi. Item ada di cache item, jadi DAX mengembalikannya ke Alice tanpa mengakses tabel DynamoDB.

Dalam skenario ini, Alice dan Bob melihat representasi yang berbeda dari item `ProductCatalog` yang sama. Hal ini terjadi sampai DAX menghapus item dari cache item, atau sampai pengguna lain memperbarui item yang sama lagi menggunakan DAX.

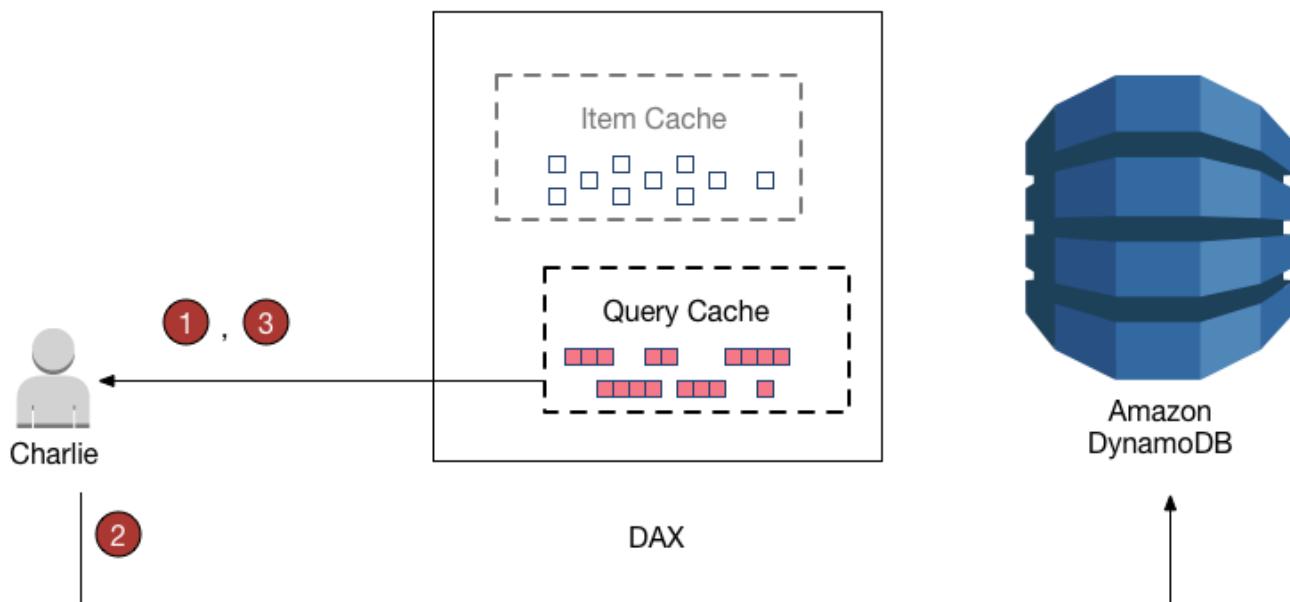
Write-Around

Jika aplikasi Anda perlu menulis data dalam jumlah besar (seperti beban data massal), mungkin masuk akal untuk melewati DAX dan menulis data langsung ke DynamoDB. Strategi seperti write-

around mengurangi latensi penulisan. Namun, cache item tidak tetap sinkron dengan data di DynamoDB.

Jika Anda memutuskan untuk menggunakan strategi write-around, ingat bahwa DAX mengisi cache item setiap kali aplikasi menggunakan klien DAX untuk membaca data. Hal ini dapat menguntungkan dalam beberapa kasus karena memastikan bahwa hanya data yang paling sering dibaca cache (sebagai lawan dari data yang paling sering ditulis).

Sebagai contoh, pertimbangkan pengguna (Charlie) yang ingin bekerja dengan tabel yang berbeda, tabel `GameScores`, menggunakan DAX. Kunci partisi untuk `GameScores` adalah `UserId`, sehingga semua skor Charlie akan memiliki `UserId` yang sama.



1. Charlie ingin mengambil semua skornya, sehingga ia mengirimkan Query ke DAX. Dengan asumsi bahwa permintaan ini belum dikeluarkan sebelumnya, DAX meneruskan kueri untuk DynamoDB untuk diproses. Itu akan menyimpan hasil dalam cache kueri DAX, dan kemudian mengembalikan hasil ke Charlie. Set hasil tetap tersedia dalam cache kueri sampai dihapus.
2. Sekarang anggaplah bahwa Charlie memainkan game Meteor Blasters dan mencapai skor tinggi. Charlie mengirim permintaan UpdateItem ke DynamoDB, memodifikasi item dalam tabel `GameScores`.
3. Akhirnya, Charlie memutuskan untuk menjalankan kembali Query sebelumnya untuk mengambil semua data dari `GameScores`. Charlie tidak melihat skor tinggi untuk Meteor Blasters di daftar

hasil. Hal ini karena hasil kueri berasal dari cache kueri, bukan cache item. Kedua cache independen satu sama lain, sehingga perubahan dalam satu cache tidak mempengaruhi cache lainnya.

DAX tidak menyegarkan set hasil dalam cache kueri dengan data terbaru dari DynamoDB. Setiap set hasil yang ditetapkan dalam cache kueri saat ini sebagai waktu yang Query atau operasi Scan yang dilakukan. Jadi, hasil Query Charlie tidak mencerminkan operasi PutItem. Hal ini terjadi sampai DAX menghapus set hasil dari cache kueri.

Mengembangkan dengan klien DynamoDB Accelerator (DAX)

Untuk menggunakan DAX dari aplikasi, Anda menggunakan klien DAX untuk bahasa pemrograman Anda. Klien DAX dirancang untuk gangguan minimal untuk aplikasi Amazon DynamoDB Anda yang sudah ada dengan hanya beberapa modifikasi kode sederhana yang diperlukan.

Note

Klien DAX untuk berbagai bahasa pemrograman tersedia di situs berikut:

- <http://dax-sdk.s3-website-us-west-2.amazonaws.com>

Bagian ini menunjukkan cara memulai instans Amazon EC2 di default Amazon VPC Anda, terhubung ke instans, dan menjalankan aplikasi sampel. Hal ini juga menyediakan informasi tentang cara mengubah aplikasi yang sudah ada sehingga dapat menggunakan klaster DAX Anda.

Topik

- [Tutorial: Menjalankan aplikasi sampel menggunakan DynamoDB Accelerator \(DAX\)](#)
- [Memodifikasi aplikasi yang ada untuk menggunakan DAX](#)

Tutorial: Menjalankan aplikasi sampel menggunakan DynamoDB Accelerator (DAX)

Tutorial ini menunjukkan cara meluncurkan instans Amazon EC2 di cloud privat virtual (VPC) default, terhubung ke instans, dan menjalankan aplikasi sampel yang menggunakan Amazon DynamoDB Accelerator (DAX).

Note

Untuk menyelesaikan tutorial ini, Anda harus menjalankan kluster DAX di VPC default. Jika Anda belum membuat kluster DAX, lihat petunjuk di [Membuat kluster DAX](#).

Topik

- [Langkah 1: Luncurkan instans Amazon EC2](#)
- [Langkah 2: Buat pengguna dan kebijakan](#)
- [Langkah 3: Konfigurasi instans Amazon EC2](#)
- [Langkah 4: Jalankan aplikasi sampel](#)

Langkah 1: Luncurkan instans Amazon EC2

Ketika kluster Amazon DynamoDB Accelerator (DAX) tersedia, Anda dapat meluncurkan instans Amazon EC2 di Amazon VPC default Anda. Kemudian Anda dapat menginstal dan menjalankan perangkat lunak klien DAX pada instans tersebut.

Cara meluncurkan instans EC2

1. Masuk ke AWS Management Console dan buka konsol Amazon EC2 di <https://console.aws.amazon.com/ec2/>.
2. Pilih Luncurkan Instans dan lakukan hal berikut:

Langkah 1: Pilih Amazon Machine Image (AMI)

1. Di daftar AMI, temukan Amazon Linux AMI dan pilih Pilih.

Langkah 2: Pilih Jenis Instans

1. Dalam daftar jenis instans, pilih t2.micro.
2. Pilih Berikutnya: Konfigurasi Detail Instans.

Langkah 3: Konfigurasi Detail Instans

1. Untuk Jaringan, pilih VPC default Anda.

2. Pilih Berikutnya: Tambahkan Penyimpanan.

Langkah 4: Tambahkan Penyimpanan

1. Lewati langkah ini dengan memilih Berikutnya: Tambahkan Tanda.

Langkah 5: Tambahkan Tanda

1. Lewati langkah ini dengan memilih Berikutnya: Konfigurasi Grup Keamanan.

Langkah 6: Konfigurasi Grup Keamanan

1. Pilih grup keamanan yang ada.
2. Dalam daftar grup keamanan, pilih default. Ini adalah grup keamanan default untuk VPC Anda.
3. Pilih Berikutnya: Tinjau dan Luncurkan.

Langkah 7: Tinjau Peluncuran Instans

1. Pilih Luncurkan.
3. Di jendela Pilih pasangan kunci yang ada atau buat pasangan kunci baru, lakukan salah satu hal berikut:
 - Jika Anda tidak memiliki pasangan kunci Amazon EC2, pilih Buat pasangan kunci baru dan ikuti petunjuknya. Anda diminta untuk mengunduh file kunci privat (file .pem). Anda akan memerlukan file ini nanti ketika masuk ke instans Amazon EC2.
 - Jika Anda sudah memiliki pasangan kunci Amazon EC2 yang ada, buka Pilih pasangan kunci dan pilih pasangan kunci Anda dari daftar. Anda harus sudah memiliki file kunci privat (file .pem) yang tersedia untuk masuk ke instans Amazon EC2 Anda.
4. Setelah mengonfigurasi pasangan kunci, pilih Luncurkan Instans.
5. Di panel navigasi konsol, pilih Dasbor EC2, lalu pilih instans yang Anda luncurkan. Pada panel bawah, di tab Deskripsi, cari DNS Publik untuk instans Anda, misalnya: `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`. Catat nama DNS publik ini karena Anda membutuhkannya untuk [Langkah 3: Konfigurasi instans Amazon EC2](#).

Note

Diperlukan waktu beberapa menit agar instans Amazon EC2 Anda tersedia. Sementara itu, lanjutkan ke [Langkah 2: Buat pengguna dan kebijakan](#) dan ikuti petunjuk di sana.

Langkah 2: Buat pengguna dan kebijakan

Pada langkah ini, Anda membuat pengguna dengan kebijakan yang memberikan akses ke kluster Amazon DynamoDB Accelerator (DAX) dan DynamoDB menggunakan AWS Identity and Access Management. Kemudian, Anda dapat menjalankan aplikasi yang berinteraksi dengan kluster DAX Anda.

Mendaftar Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar Akun AWS

1. Buka <https://portal.aws.amazon.com/billing/signup>.
2. Ikuti petunjuk secara online.

Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

Saat Anda mendaftar Akun AWS, Pengguna root akun AWS akan dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik keamanan, [tetapkan akses administratif ke pengguna administratif](#), dan hanya gunakan pengguna root untuk melakukan [tugas yang memerlukan akses pengguna root](#).

AWS akan mengirimkan email konfirmasi kepada Anda setelah proses pendaftaran selesai. Anda dapat melihat aktivitas akun saat ini dan mengelola akun dengan mengunjungi <https://aws.amazon.com/> dan memilih Akun Saya.

Membuat pengguna administratif

Setelah mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat sebuah pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Mengamankan Pengguna root akun AWS Anda

1. Masuk ke [AWS Management Console](#) sebagai pemilik akun dengan memilih Pengguna root dan memasukkan alamat email Akun AWS Anda. Di halaman berikutnya, masukkan kata sandi Anda.

Untuk bantuan masuk menggunakan pengguna root, lihat [Masuk sebagai pengguna root](#) dalam Panduan Pengguna AWS Sign-In.

2. Aktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

Untuk petunjuknya, silakan lihat [Mengaktifkan perangkat MFA virtual untuk pengguna root Akun AWS Anda \(konsol\)](#) dalam Panduan Pengguna IAM.

Membuat pengguna administratif

1. Aktifkan Pusat Identitas IAM.

Untuk mendapatkan petunjuk, silakan lihat [Mengaktifkan AWS IAM Identity Center](#) di Panduan Pengguna AWS IAM Identity Center.

2. Di Pusat Identitas IAM, berikan akses administratif ke sebuah pengguna administratif.

Untuk mendapatkan tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, silakan lihat [Mengonfigurasi akses pengguna dengan Direktori Pusat Identitas IAM default](#) di Panduan Pengguna AWS IAM Identity Center.

Masuk sebagai pengguna administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email Anda saat Anda membuat pengguna Pusat Identitas IAM.

Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat [Masuk ke portal akses AWS](#) dalam Panduan Pengguna AWS Sign-In.

Untuk memberikan akses, tambahkan izin ke pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

Buat rangkaian izin. Ikuti petunjuk dalam [Buat set izin](#) dalam Panduan Pengguna AWS IAM Identity Center.

- Pengguna yang dikelola di IAM melalui penyedia identitas:

Buat peran untuk federasi identitas. Ikuti petunjuk dalam [Membuat peran untuk penyedia identitas pihak ketiga \(federasi\)](#) di Panduan Pengguna IAM.

- Pengguna IAM:
 - Buat peran yang dapat diambil pengguna Anda. Ikuti petunjuk dalam [Membuat peran untuk pengguna IAM](#) dalam Panduan Pengguna IAM.
 - (Tidak disarankan) Pasang kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk di [Menambahkan izin ke pengguna \(konsol\)](#) dalam Panduan Pengguna IAM.

Cara menggunakan editor kebijakan JSON untuk membuat kebijakan

1. Masuk ke AWS Management Console dan buka konsol IAM di <https://console.aws.amazon.com/iam/>.
2. Pada panel navigasi di sebelah kiri, pilih Kebijakan.

Jika ini pertama kalinya Anda memilih Kebijakan, akan muncul halaman Selamat Datang di Kebijakan Terkelola. Pilih Memulai.

3. Di bagian atas halaman, pilih Buat kebijakan.
4. Di bagian Editor kebijakan, pilih opsi JSON.
5. Masukkan atau tempel dokumen kebijakan JSON. Untuk detail bahasa kebijakan IAM, lihat [Referensi kebijakan JSON IAM](#).
6. Selesaikan peringatan keamanan, kesalahan, atau peringatan umum yang dihasilkan selama [validasi kebijakan](#), lalu pilih Berikutnya.

Note

Anda dapat beralih antara opsi editor Visual dan JSON kapan saja. Namun, jika Anda melakukan perubahan atau memilih Berikutnya di editor Visual, IAM dapat merestrukturisasi kebijakan Anda untuk mengoptimalkannya bagi editor visual. Untuk informasi selengkapnya, lihat [Restrukturisasi kebijakan](#) dalam Panduan Pengguna IAM.

7. (Opsional) Saat membuat atau mengedit kebijakan AWS Management Console, Anda dapat membuat templat kebijakan JSON atau YAML yang dapat digunakan dalam templat AWS CloudFormation.

Untuk melakukannya, di editor Kebijakan pilih Tindakan, lalu pilih Buat CloudFormation templat. Untuk mempelajari selengkapnya tentang AWS CloudFormation, lihat [Referensi jenis sumber daya AWS Identity and Access Management](#) di Panduan Pengguna AWS CloudFormation.

8. Setelah selesai menambahkan izin ke kebijakan, pilih Berikutnya.
9. Pada halaman Tinjau dan buat, masukkan Nama kebijakan dan Deskripsi (opsional) untuk kebijakan yang Anda buat. Tinjau Izin yang ditentukan dalam kebijakan ini untuk melihat izin yang diberikan oleh kebijakan Anda.
10. (Opsional) Tambahkan metadata ke kebijakan dengan melampirkan tanda sebagai pasangan nilai kunci. Untuk informasi selengkapnya tentang penggunaan tanda di IAM, lihat [Menandai sumber daya IAM](#) di Panduan Pengguna IAM.
11. Pilih Buat kebijakan untuk menyimpan kebijakan baru Anda.

Dokumen kebijakan – Salin dan tempel dokumen berikut untuk membuat kebijakan JSON.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "dax:*"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    },
    {
      "Action": [
        "dynamodb:*"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Langkah 3: Konfigurasi instans Amazon EC2

Ketika instans Amazon EC2 Anda tersedia, Anda dapat masuk ke instans dan mempersiapkannya untuk digunakan.

Note

Langkah-langkah berikut menganggap bahwa Anda terhubung ke instans Amazon EC2 dari komputer yang menjalankan Linux. Terkait cara lain untuk terhubung, lihat [Menghubungkan ke Instans Linux Anda](#) di Panduan Pengguna Amazon EC2 untuk Instans Linux.

Cara mengonfigurasi instans EC2

1. Buka konsol Amazon EC2 di <https://console.aws.amazon.com/ec2/>.
2. Gunakan perintah ssh untuk masuk ke instans Amazon EC2 Anda, seperti yang ditampilkan dalam contoh berikut.

```
ssh -i my-keypair.pem ec2-user@public-dns-name
```

Anda perlu menentukan file kunci privat (file `.pem`) dan nama DNS publik instans Anda. (Lihat [Langkah 1: Luncurkan instans Amazon EC2](#)).

ID login adalah `ec2-user`. Tidak diperlukan kata sandi.

3. Setelah Anda masuk ke instans EC2, konfigurasi kredensial AWS seperti yang ditunjukkan berikut ini. Masukkan ID kunci akses dan kunci rahasia AWS Anda (dari [Langkah 2: Buat pengguna dan kebijakan](#)) dan atur nama Wilayah default ke Wilayah Anda saat ini. (Pada contoh berikut, nama Wilayah default adalah `us-west-2`).

```
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]:
```

Setelah meluncurkan dan mengonfigurasi instans Amazon EC2, Anda dapat menguji fungsionalitas DAX menggunakan salah satu aplikasi sampel yang tersedia. Untuk informasi selengkapnya, lihat [Langkah 4: Jalankan aplikasi sampel](#).

Langkah 4: Jalankan aplikasi sampel

Untuk membantu menguji fungsionalitas Amazon DynamoDB Accelerator (DAX), Anda dapat menjalankan salah satu aplikasi sampel yang tersedia pada instans Amazon EC2.

Topik

- [DAX SDK for Go](#)
- [Java dan DAX](#)
- [.NET dan DAX](#)
- [Node.js dan DAX](#)
- [Python dan DAX](#)

DAX SDK for Go

Ikuti prosedur ini untuk menjalankan aplikasi sampel Amazon DynamoDB Accelerator (DAX) SDK for Go pada instans Amazon EC2 Anda.

Cara menjalankan sampel SDK for Go untuk DAX

1. Siapkan SDK for Go pada instans Amazon EC2 Anda:
 - a. Instal bahasa pemrograman Go (GoLang).

```
sudo yum install -y golang
```

- b. Pastikan bahwa Golang sudah diinstal dan berjalan dengan benar.

```
go version
```

Pesan seperti ini akan muncul.

```
go version go1.15.5 linux/amd64
```

Petunjuk yang tersisa bergantung pada dukungan modul, yang merupakan default pada Go versi 1.13.

2. Instal aplikasi Golang sampel.

```
go get github.com/aws-samples/aws-dax-go-sample
```

3. Jalankan program Golang berikut. Program pertama membuat tabel DynamoDB bernama TryDaxGoTable. Program kedua menulis data ke tabel.

```
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. Jalankan program Golang berikut.

```
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
```

Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian GetItem, Query, dan Scan.

5. Pada langkah sebelumnya, Anda menjalankan program terhadap titik akhir DynamoDB. Sekarang, jalankan program lagi, tetapi kali ini operasi GetItem, Query, dan Scan diproses oleh kluster DAX Anda.

Untuk menentukan titik akhir kluster DAX Anda, pilih salah satu dari berikut ini:

- Menggunakan konsol DynamoDB — Pilih kluster DAX Anda. Titik akhir kluster ditampilkan pada konsol, seperti dalam contoh berikut.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- Menggunakan AWS CLI — Masukkan perintah berikut.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

Titik akhir klaster ditampilkan pada output, seperti dalam contoh berikut.

```
{
  "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"
}
```

Sekarang jalankan program lagi, tetapi kali ini tentukan titik akhir klaster sebagai parameter baris perintah.

```
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
```

Periksa sisa output dan perhatikan informasi waktu. Waktu berlalu untuk GetItem, Query, dan Scan harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

6. Jalankan program Golang berikut untuk menghapus TryDaxGoTable.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command delete-table
```

Java dan DAX

DAX SDK for Java 2.x kompatibel dengan [AWS SDK for Java 2.x](#). SDK ini dibuat menggunakan Java 8+ dan mencakup dukungan untuk non-blocking I/O. Untuk informasi tentang penggunaan DAX dengan AWS SDK for Java 1.x, lihat [Menggunakan DAX dengan AWS SDK for Java 1.x](#)

Menggunakan klien sebagai dependensi Maven

Ikuti langkah-langkah berikut untuk menggunakan klien untuk DAX SDK for Java dalam aplikasi Anda sebagai dependensi.

1. Unduh dan instal Apache Maven. Untuk informasi selengkapnya, lihat [Mengunduh Apache Maven](#) dan [Menginstal Apache Maven](#).
2. Tambahkan dependensi Maven klien ke file Project Object Model (POM) aplikasi Anda. Dalam contoh ini, ganti `x.x.x` dengan nomor versi klien sebenarnya.

```
<!--Dependency:-->
<dependencies>
  <dependency>
    <groupId>software.amazon.dax</groupId>
    <artifactId>amazon-dax-client</artifactId>
    <version>x.x.x</version>
  </dependency>
</dependencies>
```

TryDax kode sampel

Setelah menyiapkan ruang kerja dan menambahkan DAX SDK sebagai dependensi, salin [TryDax.java](#) ke proyek Anda.

Jalankan kode menggunakan perintah ini.

```
java -cp classpath TryDax
```

Anda akan melihat output seperti yang berikut ini.

```
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.
```

Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian `GetItem` dan `Query`. Dalam hal ini, Anda menjalankan program terhadap titik akhir DynamoDB. Sekarang Anda akan menjalankan program lagi, kali ini terhadap kluster DAX.

Untuk menentukan titik akhir kluster DAX Anda, pilih salah satu dari berikut ini:

- Pada konsol DynamoDB, pilih kluster DAX Anda. Titik akhir kluster ditampilkan pada konsol, seperti dalam contoh berikut.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- Menggunakan AWS CLI, masukkan perintah berikut:

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

Alamat, port, dan URL titik akhir kluster ditampilkan pada output, seperti dalam contoh berikut.

```
{
  "Address": "my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com",
  "Port": 8111,
  "URL": "dax://my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

Sekarang jalankan program lagi, tetapi kali ini tentukan URL titik akhir kluster sebagai parameter baris perintah.

```
java -cp classpath TryDax dax://my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com
```

Lihat output dan perhatikan informasi waktu. Waktu berlalu untuk `GetItem` dan `Query` harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

Metrik SDK

Dengan DAX SDK for Java 2.x, Anda dapat mengumpulkan metrik tentang klien layanan di aplikasi Anda dan menganalisis output di Amazon CloudWatch. Lihat [Mengaktifkan metrik SDK](#) untuk informasi selengkapnya.

Note

DAX SDK for Java hanya mengumpulkan metrik `ApiCallSuccessful` dan `ApiCallDuration`.

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.waitFor().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 dan DAX

Ikuti langkah-langkah berikut untuk menjalankan sampel .NET pada instans Amazon EC2 Anda.

Note

Tutorial ini menggunakan .NET 6 SDK, tetapi juga akan berfungsi dengan .NET Core SDK. Anda akan diberi tahu cara menjalankan program di Amazon VPC default untuk mengakses kluster Amazon DynamoDB Accelerator (DAX) Anda. Jika ingin, Anda dapat menggunakan AWS Toolkit for Visual Studio untuk menulis aplikasi .NET dan deploy ke VPC Anda. Untuk informasi selengkapnya, lihat [Membuat dan Melakukan Deployment Aplikasi Elastic Beanstalk di .NET Menggunakan AWS Toolkit for Visual Studio](#) di Panduan Developer AWS Elastic Beanstalk.

Cara menjalankan sampel .NET untuk DAX

1. Buka [Halaman Microsoft Downloads](#) dan unduh .NET 6 (atau .NET Core) SDK terbaru untuk Linux. File yang diunduh adalah `dotnet-sdk-N.N.N-linux-x64.tar.gz`.
2. Ekstrak file SDK.

```
mkdir dotnet
tar zxvf dotnet-sdk-N.N.N-linux-x64.tar.gz -C dotnet
```

Ganti *N.N.N* dengan nomor versi .NET SDK sebenarnya (misalnya: `6.0.100`).

3. Verifikasi instalasi.

```
alias dotnet=$HOME/dotnet/dotnet
dotnet --version
```

Tindakan ini akan mencetak nomor versi .NET SDK.

Note

Sebagai ganti nomor versi, Anda mungkin menerima kesalahan berikut:
kesalahan: libunwind.so.8: tidak dapat membuka file objek yang dibagikan: File atau direktori tersebut tidak ada
Untuk mengatasi kesalahan ini, instal paket `libunwind`.

```
sudo yum install -y libunwind
```

Setelah melakukan ini, Anda akan dapat menjalankan perintah `dotnet --version` tanpa kesalahan.

4. Buat proyek .NET baru.

```
dotnet new console -o myApp
```

Ini membutuhkan beberapa menit untuk melakukan one-time-only pengaturan. Setelah selesai, jalankan sampel proyek.

```
dotnet run --project myApp
```

Anda akan menerima pesan berikut: Hello World!

- File `myApp/myApp.csproj` berisi metadata tentang proyek Anda. Untuk menggunakan klien DAX di aplikasi Anda, modifikasi file sehingga terlihat seperti berikut ini.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="AWSSDK.DAX.Client" Version="*" />
  </ItemGroup>
</Project>
```

- Unduh sampel kode sumber program (file `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Setelah unduhan selesai, ekstrak file sumber.

```
unzip TryDax.zip
```

- Sekarang jalankan program sampel satu per satu. Untuk setiap program, salin isinya ke `myApp/Program.cs`, kemudian jalankan proyek `MyApp`.

Jalankan program .NET berikut. Program pertama membuat tabel DynamoDB bernama `TryDaxTable`. Program kedua menulis data ke tabel.

```
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
```

- Berikutnya, jalankan beberapa program untuk melakukan operasi `GetItem`, `Query`, dan `Scan` di kluster DAX Anda. Untuk menentukan titik akhir kluster DAX Anda, pilih salah satu dari berikut ini:

- Menggunakan konsol DynamoDB — Pilih kluster DAX Anda. Titik akhir kluster ditampilkan pada konsol, seperti dalam contoh berikut.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- Menggunakan AWS CLI — Masukkan perintah berikut.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

Titik akhir kluster ditampilkan pada output, seperti dalam contoh berikut.

```
{
  "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"
}
```

Sekarang jalankan program berikut untuk menentukan titik akhir kluster Anda sebagai parameter baris perintah. (Ganti titik akhir sampel dengan titik akhir kluster DAX Anda yang sebenarnya).

```
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
```

Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian `GetItem`, `Query`, dan `Scan`.

9. Jalankan program .NET berikut untuk menghapus `TryDaxTable`.

```
cp TryDax/dotNet/06-DeleteTable.cs myApp/Program.cs
dotnet run --project myApp
```


Untuk informasi selengkapnya tentang program tersebut, lihat bagian berikut:

- [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

Program 01-CreateTable.cs membuat tabel (TryDaxTable). Program .NET yang tersisa di bagian ini bergantung pada tabel ini.

```
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

Program 02-Write-Data.cs menulis data pengujian ke TryDaxTable.

```
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

Program 03-GetItem-Test.cs melakukan operasi GetItem di TryDaxTable.

```
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

Program `04-Query-Test.cs` melakukan operasi Query di `TryDaxTable`.

```
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

Program 05-Scan-Test.cs melakukan operasi Scan di TryDaxTable.

```

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

Program 06-DeleteTable.cs menghapus TryDaxTable. Jalankan program ini setelah Anda selesai melakukan pengujian.

```
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();
        }
    }
}
```

Node.js dan DAX

Ikuti langkah-langkah berikut untuk menjalankan aplikasi sampel Node.js pada instans Amazon EC2 Anda.

Cara menjalankan sampel Node.js untuk DAX

1. Atur Node.js pada instans Amazon EC2 Anda sebagai berikut:

- a. Instal manajer versi simpul (nvm).

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
```

- b. Gunakan nvm untuk menginstal Node.js.

```
nvm install 12.16.3
```

- c. Pastikan bahwa Node.js sudah diinstal dan berjalan dengan benar.

```
node -e "console.log('Running Node.js ' + process.version)"
```

Tindakan ini akan menampilkan pesan berikut.

```
Running Node.js v12.16.3
```

2. Instal klien Node.js DAX menggunakan manajer paket simpul (npm).

```
npm install amazon-dax-client
```

3. Unduh sampel kode sumber program (file .zip).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Setelah unduhan selesai, ekstrak file sumber.

```
unzip TryDax.zip
```

4. Jalankan program Node.js berikut. Program pertama membuat tabel Amazon DynamoDB bernama TryDaxTable. Program kedua menulis data ke tabel.

```
node 01-create-table.js
node 02-write-data.js
```

5. Jalankan program Node.js berikut.

```
node 03-getitem-test.js
node 04-query-test.js
node 05-scan-test.js
```

Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian `GetItem`, `Query`, dan `Scan`.

6. Pada langkah sebelumnya, Anda menjalankan program terhadap titik akhir DynamoDB. Jalankan program lagi, tetapi kali ini, operasi `GetItem`, `Query` dan `Scan` diproses oleh kluster DAX Anda.

Untuk menentukan titik akhir kluster DAX Anda, pilih salah satu dari berikut ini.

- Menggunakan konsol DynamoDB—Pilih kluster DAX Anda. Titik akhir kluster ditampilkan pada konsol, seperti dalam contoh berikut.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- Menggunakan AWS CLI—Masukkan perintah berikut.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

Titik akhir kluster ditampilkan pada output, seperti dalam contoh berikut.

```
{
  "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"
}
```

Sekarang jalankan program lagi, tetapi kali ini tentukan titik akhir kluster sebagai parameter baris perintah.

```
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
```

Periksa sisa output dan perhatikan informasi waktu. Waktu berlalu untuk `GetItem`, `Query`, dan `Scan` harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

7. Jalankan program Node.js berikut untuk menghapus `TryDaxTable`.

```
node 06-delete-table
```

Untuk informasi selengkapnya tentang program tersebut, lihat bagian berikut:

- [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

Program `01-create-table.js` membuat tabel (`TryDaxTable`). Program Node.js yang tersisa di bagian ini bergantung pada tabel ini.

```
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

Program 02-write-data.js menulis data pengujian ke TryDaxTable.

```
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

Program 03-getitem-test.js melakukan operasi GetItem di TryDaxTable.

```
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

Program `04-query-test.js` melakukan operasi Query di `TryDaxTable`.

```
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

Program `05-scan-test.js` melakukan operasi Scan di TryDaxTable.

```
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

Program `06-delete-table.js` menghapus `TryDaxTable`. Jalankan program ini setelah Anda selesai melakukan pengujian.

```
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)
    );
  }
});
```

Python dan DAX

Ikuti prosedur ini untuk menjalankan aplikasi sampel Python pada instans Amazon EC2 Anda.

Cara menjalankan sampel Python untuk DAX

1. Instal klien Python DAX menggunakan utilitas pip.

```
pip install amazon-dax-client
```

2. Unduh sampel kode sumber program (file .zip).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Setelah unduhan selesai, ekstrak file sumber.

```
unzip TryDax.zip
```

3. Jalankan program Python berikut. Program pertama membuat tabel Amazon DynamoDB bernama TryDaxTable. Program kedua menulis data ke tabel.

```
python 01-create-table.py
python 02-write-data.py
```

4. Jalankan program Python berikut.

```
python 03-getitem-test.py
python 04-query-test.py
python 05-scan-test.py
```

Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian GetItem, Query, dan Scan.

5. Pada langkah sebelumnya, Anda menjalankan program terhadap titik akhir DynamoDB. Sekarang jalankan program lagi, tetapi kali ini, operasi GetItem, Query dan Scan diproses oleh klaster DAX Anda.

Untuk menentukan titik akhir klaster DAX Anda, pilih salah satu dari berikut ini:

- Menggunakan konsol DynamoDB — Pilih klaster DAX Anda. Titik akhir klaster ditampilkan pada konsol, seperti dalam contoh berikut.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- Menggunakan AWS CLI — Masukkan perintah berikut.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

Titik akhir klaster ditampilkan pada output, seperti dalam contoh ini.

```
{
  "Address": "my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com",
  "Port": 8111,
  "URL": "dax://my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

Jalankan program lagi, tetapi kali ini tentukan titik akhir kluster sebagai parameter baris perintah.

```
python 03-getitem-test.py dax://my-cluster.16fzcv.dax-clusters.us-
east-1.amazonaws.com
python 04-query-test.py dax://my-cluster.16fzcv.dax-clusters.us-
east-1.amazonaws.com
python 05-scan-test.py dax://my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com
```

Periksa sisa output dan perhatikan informasi waktu. Waktu berlalu untuk `GetItem`, `Query`, dan `Scan` harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

6. Jalankan program Python berikut untuk menghapus `TryDaxTable`.

```
python 06-delete-table.py
```

Untuk informasi selengkapnya tentang program tersebut, lihat bagian berikut:

- [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

Program `01-create-table.py` membuat tabel (`TryDaxTable`). Program Python yang tersisa di bagian ini bergantung pada tabel ini.

```
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"},
        ],
        "ProvisionedThroughput": {"ReadCapacityUnits": 10, "WriteCapacityUnits": 10},
    }
    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

Program 02-write-data.py menulis data pengujian ke TryDaxTable.

```
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

Program 03-getitem-test.py melakukan operasi GetItem di TryDaxTable. Contoh ini diberikan untuk Wilayah eu-west-1.

```
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

Program `04-query-test.py` melakukan operasi Query di `TryDaxTable`.

```

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

Program 05-scan-test.py melakukan operasi Scan di TryDaxTable.

```
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

Program `06-delete-table.py` menghapus `TryDaxTable`. Jalankan program ini setelah Anda selesai menguji fungsionalitas Amazon DynamoDB Accelerator (DAX).

```

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!")
```

Memodifikasi aplikasi yang ada untuk menggunakan DAX

Jika Anda sudah memiliki aplikasi Java yang menggunakan Amazon DynamoDB, yang dapat Anda modifikasi sehingga dapat mengakses kluster DynamoDB Accelerator (DAX) Anda. Anda tidak harus menulis ulang seluruh aplikasi karena klien Java DAX mirip dengan klien level rendah DynamoDB yang disertakan dalam AWS SDK for Java 2.x. Lihat [Bekerja dengan item di DynamoDB](#) untuk detailnya.

Note

Contoh ini menggunakan AWS SDK for Java 2.x. Untuk versi SDK for Java 1.x, lihat [Memodifikasi SDK for Java 1.x yang ada agar menggunakan DAX](#).

Untuk mengubah program Anda, ganti klien DynamoDB dengan klien DAX.

```
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.16fzcv.dax-
clusters.us-east-1.amazonaws.com"
        .region(region)
        .addMetricPublisher(cloudWatchMetricsPub) // optionally enable SDK
metric collection
    .build())
    .build();
```

Anda juga dapat menggunakan pustaka tingkat tinggi yang merupakan bagian dari AWS SDK for Java 2.x, menggantikan klien DynamoDB dengan klien DAX.

```
Region region = Region.US_EAST_1;
```

```
DynamoDbAsyncClient dax = ClusterDaxAsyncClient.builder()
    .overrideConfiguration(Configuration.builder()
        .url(<cluster url>) // for example, "dax://my-cluster.16fzcv.dax-
clusters.us-east-1.amazonaws.com"
        .region(region)
        .build())
    .build();

DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
    .dynamoDbClient(dax)
    .build();
```

Untuk informasi selengkapnya, lihat [Memetakan item dalam tabel DynamoDB](#).

Mengelola kluster DAX

Bagian ini membahas beberapa tugas manajemen umum untuk kluster Amazon DynamoDB Accelerator (DAX).

Topik

- [Izin IAM untuk mengelola kluster DAX](#)
- [Penskalaan kluster DAX](#)
- [Menyesuaikan Pengaturan kluster DAX](#)
- [Mengkonfigurasi Pengaturan TTL](#)
- [Support pemberian tag untuk DAX](#)
- [AWS CloudTrailIntegrasi](#)
- [Menghapus kluster DAX](#)

Izin IAM untuk mengelola kluster DAX

Ketika Anda mengelola kluster DAX menggunakan AWS Management Console atau AWS Command Line Interface (AWS CLI), kami sangat menyarankan agar Anda mempersempit cakupan tindakan yang dapat dilakukan pengguna. Dengan demikian, Anda membantu mengurangi risiko sambil mengikuti prinsip hak istimewa.

Diskusi berikut berfokus pada kontrol akses untuk API manajemen DAX. Untuk informasi selengkapnya, lihat [Akselerator Amazon DynamoDB](#) di dalam Referensi API Amazon DynamoDB.

Note

Untuk informasi lebih rinci tentang mengelola AWS Identity and Access Management izin (IAM), lihat penjelasan berikut ini:

- IAM dan pembuatan klaster DAX: [Membuat klaster DAX](#).
- Operasi bidang data IAM dan DAX: [Kontrol akses DAX](#).

Untuk API manajemen DAX, Anda tidak dapat melingkupi tindakan API ke sumber daya tertentu. Parameter elemen Resource harus diatur ke "*". Hal ini berbeda dari DAX bidang data operasi API, seperti `GetItem`, `Query`, dan `Scan`. Operasi bidang data terpapar melalui klien DAX, dan operasi tersebut dapat dilingkupi untuk sumber daya tertentu.

Untuk menggambarkannya, pertimbangkan dokumen kebijakan IAM berikut.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "dax:*"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
      ]
    }
  ]
}
```

Misalkan maksud dari kebijakan ini adalah untuk mengizinkan panggilan API manajemen DAX untuk klaster `DAXCluster01` dan hanya klaster itu.

Sekarang anggaplah bahwa pengguna melakukan perintah AWS CLI berikut.

```
aws dax describe-clusters
```

Perintah ini gagal dengan pengecualian yang Tidak Diizinkan karena panggilan API `DescribeClusters` dasar tidak dapat dilingkupi ke klaster tertentu. Meskipun kebijakan sintaksis

valid, perintah gagal karena elemen `Resource` harus diatur ke `"*"`. Namun, jika pengguna menjalankan program yang mengirimkan panggilan bidang data DAX (seperti `GetItem` atau `Query`) ke `DAXCluster01`, panggilan tersebut akan berhasil. Hal ini karena API bidang data DAX dapat dilingkupi ke sumber daya tertentu (dalam hal ini, `DAXCluster01`).

Jika Anda ingin menulis kebijakan IAM komprehensif tunggal untuk mencakup API manajemen DAX dan API bidang data DAX, kami sarankan bahwa Anda menyertakan dua pernyataan yang berbeda dalam dokumen kebijakan. Salah satu pernyataan ini harus membahas API bidang data DAX, sementara pernyataan lainnya membahas API manajemen DAX.

Kebijakan berikut menunjukkan sebuah contoh. Perhatikan cara `DAXDataAPIs` dilingkupi ke sumber daya `DAXCluster01`, tetapi sumber daya untuk `DAXManagementAPIs` harus `"*"`. Tindakan yang ditampilkan dalam setiap pernyataan hanya sebagai ilustrasi. Anda dapat menyesuaikan semuanya sesuai kebutuhan untuk aplikasi Anda.

```
{
  "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": [
        "*"
    ]
}
]
```

Penskalaan kluster DAX

Ada dua opsi yang tersedia untuk menskalakan kluster DAX. Pilihan pertama adalah penskalaan horizontal, di mana Anda menambahkan replika pembacaan untuk instans. Pilihan kedua adalah Penskalaan vertikal, di mana Anda memilih jenis simpul yang berbeda. Untuk memperoleh saran tentang cara memilih ukuran kluster dan jenis simpul yang sesuai untuk aplikasi Anda, lihat [Panduan ukuran cluster DAX](#).

Penskalaan Horizontal

Dengan skala horizontal, Anda dapat meningkatkan throughput untuk membaca operasi dengan menambahkan lebih banyak replika pembacaan ke kluster. Satu kluster DAX mendukung hingga 10 replika pembacaan, dan Anda dapat menambahkan atau menghapus replika saat kluster sedang dijalankan.

Contoh AWS CLI berikut menunjukkan cara menambah atau mengurangi jumlah simpul. Argumen `--new-replication-factor` menentukan jumlah total simpul di kluster. Salah satu simpul adalah simpul primer, dan simpul lainnya replika pembacaan.


```
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

Status instans berubah menjadi `modifying` ketika Anda mengubah faktor replikasi. Status berubah menjadi `available` ketika perubahan selesai.

Penskalaan vertikal

Jika Anda memiliki set kerja besar data, aplikasi Anda mungkin mendapatkan manfaat dari menggunakan jenis simpul yang lebih besar. Simpul yang lebih besar dapat mengaktifkan klaster untuk menyimpan lebih banyak data dalam memori, mengurangi cache miss dan meningkatkan performa penerapan secara keseluruhan dari aplikasi. (Semua simpul dalam instans DAX harus dari jenis yang sama.)

Jika klaster DAX Anda memiliki tingkat tinggi penulisan operasi atau cache miss, aplikasi Anda mungkin akan bermanfaat juga dari penggunaan jenis simpul yang lebih besar. Operasi penulisan dan cache miss mengkonsumsi sumber daya pada simpul primer klaster. Oleh karena itu, menggunakan jenis simpul yang lebih besar mungkin akan meningkatkan performa simpul primer dan dengan demikian mengizinkan throughput yang lebih tinggi untuk jenis operasi ini.

Anda tidak dapat mengubah jenis simpul di klaster DAX yang sedang berjalan. Sebagai gantinya, Anda harus membuat klaster baru dengan jenis simpul yang diinginkan. Untuk mengetahui daftar jenis data yang didukung, lihat [Simpul](#).

Anda dapat membuat klaster DAX baru menggunakan AWS Management Console, [AWS CloudFormation](#), AWS CLI, atau [AWS SDK](#). (Untuk AWS CLI, gunakan parameter `--node-type` untuk menentukan jenis simpul.)


Menyesuaikan Pengaturan klaster DAX

Ketika Anda membuat sebuah klaster DAX, pengaturan default berikut digunakan:

- Penghapusan cache otomatis diaktifkan dengan Time to Live (TTL) selama 5 menit
- Tidak ada preferensi untuk Availability Zones
- Tidak ada preferensi untuk jendela pemeliharaan
- Notifikasi dinonaktifkan

Untuk klaster baru, Anda dapat menyesuaikan pengaturan pada saat pembuatan. Untuk melakukannya di AWS Management Console, bersihkan Pengaturan default penggunaan untuk mengubah pengaturan berikut:

- Jaringan dan Keamanan mengizinkan Anda untuk menjalankan simpul klaster DAX individual di Availability Zone yang berbeda di Wilayah AWS. Jika Anda memilih Tidak Ada Preferensi, simpul akan didistribusikan di antara Availability Zone secara otomatis.
- Grup Parameter — Adalah sebuah set bernama parameter yang diterapkan untuk setiap simpul di klaster. Anda dapat menggunakan grup parameter untuk menentukan perilaku TTL cache. Anda dapat mengubah nilai dari setiap parameter yang diberikan dalam grup parameter (kecuali grup parameter default `default.dax.1.0`) kapan pun.
- Jendela pemeliharaan — Adalah jangka waktu mingguan dimana peningkatan dan patch perangkat lunak diterapkan ke simpul dalam instans. Anda dapat memilih hari mulai, waktu mulai, dan durasi jendela pemeliharaan. Jika Anda memilih Tidak Ada Preferensi, jendela pemeliharaan dipilih secara acak dari blok waktu 8 jam per wilayah. Untuk informasi selengkapnya, lihat [Jendela pemeliharaan](#).

 Note

Grup Parameter dan Jendela pemeliharaan juga dapat diubah setiap saat pada klaster berjalan.

Ketika proses pemeliharaan berlangsung, DAX dapat memberitahu Anda menggunakan Amazon Simple Notification Service (Amazon SNS). Untuk mengonfigurasi notifikasi, pilih opsi dari pemilih Topik untuk pemberitahuan SNS. Anda dapat membuat topik Amazon SNS baru, atau menggunakan topik yang sudah ada.

Untuk informasi selengkapnya tentang mengatur dan berlanggan ke topik Amazon SNS, lihat [Memulai dengan Amazon SNS](#) di dalam Panduan Developer Amazon Simple Notification Service.

Mengkonfigurasi Pengaturan TTL

DAX mempertahankan dua cache untuk data yang dibaca dari DynamoDB:

- Cache item — Untuk item yang diambil menggunakan `GetItem` atau `BatchGetItem`.
- Cache kueri — Untuk set hasil diambil menggunakan `Query` atau `Scan`.

Untuk informasi selengkapnya, lihat [Cache Item](#) dan [Cache kueri](#).

Default TTL untuk masing-masing cache ini adalah 5 menit. Jika Anda ingin menggunakan pengaturan TTL yang berbeda, Anda dapat meluncurkan klaster DAX menggunakan grup parameter kustom. Untuk melakukannya di konsol, pilih DAX | grup parameter di panel navigasi.

Anda juga dapat melakukan tugas-tugas ini menggunakan AWS CLI. Contoh berikut menunjukkan cara meluncurkan klaster DAX baru menggunakan grup parameter khusus. Dalam contoh ini, item cache TTL diatur ke 10 menit, dan permintaan cache TTL diatur ke 3 menit.

1. Buat grup parameter baru.

```
aws dax create-parameter-group \  
  --parameter-group-name custom-ttl
```

2. Atur cache item TTL ke 10 menit (600000 milidetik).

```
aws dax update-parameter-group \  
  --parameter-group-name custom-ttl \  
  --parameter-name-values "ParameterName=record-ttl-millis,ParameterValue=600000"
```

3. Mengatur cache permintaan TTL 3 menit (180000 milidetik).

```
aws dax update-parameter-group \  
  --parameter-group-name custom-ttl \  
  --parameter-name-values "ParameterName=query-ttl-millis,ParameterValue=180000"
```

4. Verifikasi bahwa parameter telah ditetapkan dengan benar.

```
aws dax describe-parameters --parameter-group-name custom-ttl \  
  --query "Parameters[*].[ParameterName,Description,ParameterValue]"
```

Anda sekarang dapat meluncurkan klaster DAX baru dengan grup parameter ini.

```
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

Anda tidak dapat mengubah grup parameter yang sedang digunakan oleh instans DAX berjalan.

Support pemberian tag untuk DAX

Banyak layanan AWS, termasuk DynamoDB, support penandaan adalah kemampuan untuk melabeli sumber daya dengan nama yang ditetapkan pengguna. Anda dapat menetapkan tag atau label ke klaster DAX, mengizinkan Anda untuk dengan cepat mengidentifikasi semua AWS yang memiliki tag yang sama, atau untuk mengkategorikan tagihan AWS dengan tag yang Anda tetapkan.

Untuk informasi selengkapnya, lihat [Menambahkan tag dan label ke sumber daya](#).

Menggunakan AWS Management Console

Untuk mengelola tag klaster DAX

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi, di bawah DAX, pilih klaster.
3. Pilih klaster DB yang ingin Anda gunakan.
4. Pilih tab Tag. Anda dapat menambahkan, mencantumkan, mengedit, atau menghapus tag di sini.

Saat pengaturan sesuai keinginan Anda, pilih Terapkan Perubahan.

Menggunakan AWS CLI

Ketika Anda menggunakan AWS CLI untuk mengelola tag klaster DAX, Anda harus terlebih dahulu menentukan Amazon Resource Name (ARN) untuk klaster. Contoh berikut menunjukkan cara menentukan ARN untuk klaster bernama MyDAXCluster.

```
aws dax describe-clusters \  
  --cluster-name MyDAXCluster \  
  --query "Clusters[*].ClusterArn"
```

Dalam output, ARN akan terlihat mirip dengan ini: `arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster`

Contoh berikut menunjukkan cara melabeli klaster.

```
aws dax tag-resource \  
  --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \  
  --tags="Key=ClusterUsage,Value=prod"
```

Daftar semua tag untuk klaster.

```
aws dax list-tags \  
  --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

Untuk menghapus tag, tentukan kuncinya.

```
aws dax untag-resource \  
  --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \  
  --tag-keys ClusterUsage
```

AWS CloudTrailIntegrasi

DAX terintegrasi dengan AWS CloudTrail, memungkinkan Anda untuk audit aktivitas klaster DAX. Anda dapat menggunakan CloudTrail log untuk melihat semua perubahan yang telah dibuat di tingkat klaster. Anda juga dapat melihat perubahan ke komponen klaster seperti simpul, grup subnet, dan grup parameter. Untuk informasi selengkapnya, lihat [Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail](#).

Menghapus kluster DAX

Jika Anda tidak lagi menggunakan kluster DAX, Anda harus menghapusnya untuk menghindari adanya biaya untuk sumber daya yang tidak digunakan.

Anda dapat menghapus kluster DAX menggunakan konsol atau AWS CLI. Berikut adalah contohnya.

```
aws dax delete-cluster --cluster-name mydaxcluster
```

Pemantauan DAX

Pemantauan adalah bagian penting dari pemeliharaan keandalan, ketersediaan, dan solusi Amazon DynamoDB Accelerator (DAX) AWS Anda. Anda harus mengumpulkan data pemantauan dari semua bagian solusi AWS Anda agar dapat dengan lebih mudah melakukan debug jika terjadi kegagalan multi-titik.

Namun sebelum Anda mulai memantau DAX, Anda harus membuat rencana pemantauan yang mencakup jawaban atas pertanyaan berikut:

- Apa sasaran pemantauan Anda?
- Sumber daya apa yang akan Anda pantau?
- Seberapa sering Anda akan memantau sumber daya ini?
- Alat pemantauan apa yang akan Anda gunakan?
- Siapa yang akan melakukan tugas pemantauan?
- Siapa yang harus diberi tahu saat terjadi kesalahan?

Topik

- [Alat pemantauan](#)
- [Pemantauan CloudWatch dengan Amazon](#)
- [Logging operasi DAX menggunakan AWS CloudTrail](#)

Alat pemantauan

AWS menyediakan alat yang dapat Anda gunakan untuk memantau Amazon DynamoDB Accelerator (DAX). Anda dapat mengonfigurasi beberapa alat ini untuk melakukan pemantauan, dan alat lain

memerlukan intervensi manual. Kami menyarankan agar Anda mengotomatiskan tugas pemantauan sebanyak mungkin.

Topik

- [Alat pemantauan otomatis](#)
- [Alat pemantauan manual](#)

Alat pemantauan otomatis

Anda dapat menggunakan alat pemantauan otomatis berikut untuk memantau DAX dan melaporkan saat terjadi kesalahan:

- CloudWatch Alarm Amazon — Lihat metrik tunggal selama periode waktu yang Anda tentukan, dan lakukan satu atau beberapa tindakan berdasarkan nilai metrik relatif terhadap ambang batas tertentu selama sejumlah periode waktu. Tindakan ini adalah pemberitahuan yang dikirim ke topik Amazon Simple Notification Service (Amazon SNS) atau kebijakan Amazon EC2 Auto Scaling. CloudWatch alarm tidak memicu tindakan hanya karena status tertentu; status harus berubah dan dipertahankan selama sejumlah periode tertentu. Untuk informasi selengkapnya, lihat [Pemantauan CloudWatch dengan Amazon](#).
- Amazon CloudWatch Logs — Memantau, menyimpan, dan mengakses file log Anda dari AWS CloudTrail atau sumber lainnya. Untuk informasi selengkapnya, lihat [Memantau File Log](#) di Panduan CloudWatch Pengguna Amazon.
- Amazon CloudWatch Events — Cocokkan peristiwa dan rutekan mereka ke satu fungsi target atau lebih atau aliran untuk membuat perubahan, menangkap informasi status, dan mengambil tindakan korektif. Untuk informasi selengkapnya, lihat [Apa Itu CloudWatch Acara Amazon](#) di Panduan CloudWatch Pengguna Amazon.
- AWS CloudTrail Pemantauan Log — Bagikan berkas CloudTrail log antar akun, pantau berkas log secara waktu nyata dengan mengirimnya ke CloudWatch Log, tulis aplikasi pemrosesan log di Java, dan validasi bahwa berkas log tidak berubah setelah pengiriman oleh CloudTrail. Untuk informasi selengkapnya, lihat [Bekerja dengan File CloudTrail Log](#) di Panduan AWS CloudTrail Pengguna.

Alat pemantauan manual

Bagian penting lainnya dari pemantauan DAX melibatkan pemantauan secara manual atas item-item yang tidak dicakup oleh CloudWatch alarm. DAX, CloudWatch, Trusted Advisor, dan AWS

Management Console dasbor lainnya memberikan at-a-glance pandangan tentang keadaan lingkungan Anda AWS. Sebaiknya Anda juga memeriksa file log di DAX.

- Dasbor DAX menunjukkan:
 - Kesehatan layanan
- Tampilan CloudWatch beranda menunjukkan hal berikut:
 - Alarm dan status saat ini
 - Grafik alarm dan sumber daya
 - Status kesehatan layanan

Selain itu, Anda dapat menggunakan CloudWatch untuk melakukan hal berikut:

- Buat [dasbor yang disesuaikan](#) untuk memantau layanan yang Anda inginkan.
- Data metrik grafik untuk memecahkan masalah dan menemukan tren.
- Cari dan telusuri semua metrik sumber daya AWS Anda.
- Membuat dan mengedit alarm untuk menerima pemberitahuan tentang masalah

Pemantauan CloudWatch dengan Amazon

Anda dapat memantau Amazon DynamoDB Accelerator (DAX) menggunakan CloudWatch Amazon, yang mengumpulkan dan memproses data mentah dari DAX menjadi metrik yang dapat dibaca dan hampir seperti waktu nyata. Statistik ini dicatat untuk jangka waktu dua minggu. Ini berarti bahwa Anda dapat mengakses informasi historis untuk mendapatkan perspektif yang lebih baik tentang performa aplikasi web atau layanan Anda. Secara default, data metrik DAX dikirim ke CloudWatch secara otomatis. Untuk informasi selengkapnya, lihat [Apa yang dimaksud dengan Amazon CloudWatch?](#) di Panduan CloudWatch Pengguna Amazon.

Topik

- [Bagaimana cara menggunakan metrik DAX?](#)
- [Melihat metrik dan dimensi DAX](#)
- [Membuat CloudWatch alarm untuk memantau DAX](#)
- [Pemantauan produksi](#)

Bagaimana cara menggunakan metrik DAX?

Metrik yang dilaporkan oleh DAX memberikan informasi yang dapat Anda analisis dengan cara yang berbeda. Daftar berikut menunjukkan beberapa kegunaan umum untuk metrik. Ini adalah saran untuk memulai, dan bukan daftar komprehensif.

Bagaimana Saya Dapat Melakukannya?	Metrik Terkait
Menentukan jika terjadi kesalahan sistem	Pantau <code>FaultRequestCount</code> untuk menentukan apakah setiap permintaan mengakibatkan kode HTTP 500 (kesalahan server). Hal ini dapat menunjukkan kesalahan layanan internal DAX atau HTTP 500 di tabel dasar SystemErrors .
Menentukan apakah ada kesalahan pengguna yang terjadi	Pantau <code>ErrorRequestCount</code> untuk menentukan apakah setiap permintaan mengakibatkan kode HTTP 400 (kesalahan server). Jika Anda melihat jumlah kesalahan tumbuh, Anda mungkin ingin menyelidiki dan memastikan bahwa Anda mengirim permintaan klien yang benar.
Menentukan jika setiap cache miss terjadi	Memantau <code>ItemCacheMisses</code> untuk menentukan bilangan kali item tidak dijumpai dalam cache, dan <code>QueryCacheMisses</code> dan untuk <code>ScanCacheMisses</code> menentukan jumlah kali kueri atau hasil pemindaian tidak ditemukan dalam cache.
Memantau tingkat cache hit	Gunakan Matematika CloudWatch Metrik untuk menentukan metrik laju cache hit menggunakan ekspresi matematika. Misalnya, untuk cache item, Anda dapat menggunakan ekspresi $m1/SUM([m1, m2])*100$, di mana <code>m1</code> adalah metrik <code>ItemCacheHits</code> dan <code>m2</code> adalah metrik <code>ItemCacheMisses</code> untuk klaster Anda. Untuk cache kueri dan pemindaian, Anda dapat mengikuti pola yang sama menggunakan metrik cache kueri dan pemindaian.

Melihat metrik dan dimensi DAX

Saat Anda berinteraksi dengan Amazon DynamoDB, ia akan mengirimkan metrik dan dimensi ke Amazon CloudWatch. Anda dapat menggunakan prosedur berikut untuk melihat metrik untuk DynamoDB Accelerator (DAX).

Untuk melihat metrik (konsol)

Metrik dikelompokkan terlebih dahulu berdasarkan namespace layanan, kemudian berdasarkan berbagai kombinasi dimensi dalam setiap namespace.

1. Buka CloudWatch konsol di <https://console.aws.amazon.com/cloudwatch/>.
2. Di panel navigasi, pilih Metrik.
3. PILIH namespace DAX.

Untuk melihat metrik (AWS CLI)

- Pada jendela perintah, gunakan perintah berikut.

```
aws cloudwatch list-metrics --namespace "AWS/DAX"
```

Metrik dan dimensi DAX

Bagian berikut berisi metrik dan dimensi yang dikirim DAX. CloudWatch

Metrik DAX

Metrik berikut tersedia dari DAX. DAX mengirimkan metrik CloudWatch hanya jika mereka memiliki nilai yang bukan nol.

Note

CloudWatch menggabungkan metrik DAX berikut pada interval satu menit:

- CPUUtilization
- CacheMemoryUtilization
- NetworkBytesIn
- NetworkBytesOut

- 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

Tidak semua statistik, seperti Average atau Sum, berlaku untuk setiap metrik. Namun, semua nilai ini tersedia melalui konsol DAX, atau dengan menggunakan konsol, AWS CLI, atau AWS SDK untuk semua metrik. CloudWatch Dalam tabel berikut, setiap metrik memiliki daftar statistik valid yang berlaku untuk metrik tersebut.

Metrik	Deskripsi
CPUUtilization	<p>Persentase pemanfaatan CPU simpul atau klaster.</p> <p>Unit: Percent</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average
CacheMemoryUtilization	<p>Persentase memori cache tersedia yang digunakan oleh cache item dan kueri cache pada simpul atau klaster. Data cache mulai dihapus sebelum pemanfaatan memori mencapai 100% (lihat EvictedSize metrik). Jika CacheMemoryUtilization mencapai 100% pada setiap simpul, permintaan penulisan akan di-throttle dan Anda harus mempertimbangkan untuk beralih ke klaster dengan jenis simpul yang lebih besar.</p> <p>Unit: Percent</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average
NetworkBytesIn	<p>Jumlah byte yang digunakan pada semua antarmuka oleh simpul atau klaster.</p> <p>Unit: Bytes</p> <p>Statistik Valid:</p>

Metrik	Deskripsi
	<ul style="list-style-type: none">• Minimum• Maximum• Average
NetworkBytesOut	<p>Jumlah byte yang dikirim pada semua antarmuka jaringan oleh simpul atau klaster. Metrik ini mengidentifikasi volume lalu lintas yang sedang berlangsung dalam hal jumlah byte pada simpul atau klaster tunggal.</p> <p>Unit: Bytes</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average
NetworkPacketsIn	<p>Jumlah paket yang digunakan pada semua antarmuka oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average

Metrik	Deskripsi
NetworkPacketsOut	<p>Jumlah paket yang dikirim pada semua antarmuka jaringan oleh simpul atau klaster. Metrik ini mengidentifikasi volume lalu lintas yang sedang berlangsung dalam hal jumlah paket pada simpul atau klaster tunggal.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average
GetItemRequestCount	<p>Jumlah permintaan GetItem yang ditangani oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
BatchGetItemRequestCount	<p>Jumlah permintaan BatchGetItem yang ditangani oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum
BatchWriteItemRequestCount	<p>Jumlah permintaan BatchWriteItem yang ditangani oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
DeleteItemRequestCount	<p>Jumlah permintaan DeleteItem yang ditangani oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum
PutItemRequestCount	<p>Jumlah permintaan PutItem yang ditangani oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
UpdateItemRequestCount	<p>Jumlah permintaan UpdateItem yang ditangani oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum
TransactWriteItemsCount	<p>Jumlah permintaan TransactWriteItems yang ditangani oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
TransactGetItemsCount	<p>Jumlah permintaan TransactGetItems yang ditangani oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum
ItemCacheHits	<p>Jumlah kali item dikembalikan dari cache oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
ItemCacheMisses	<p>Jumlah kali item tidak ada dalam cache simpul atau klaster, dan harus diambil dari DynamoDB.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum
QueryCacheHits	<p>Jumlah kali hasil kueri dikembalikan dari cache simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
QueryCacheMisses	<p>Jumlah kali hasil kueri tidak ada di cache simpul atau klaster, dan harus diambil dari DynamoDB.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum
ScanCacheHits	<p>Jumlah kali hasil pemindaian dikembalikan dari cache simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
ScanCacheMisses	<p>Jumlah kali hasil pemindaian tidak ada di cache simpul atau klaster, dan harus diambil dari DynamoDB.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum
TotalRequestCount	<p>Jumlah total permintaan ditangani oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
ErrorRequestCount	<p>Jumlah total permintaan yang mengakibatkan kesalahan pengguna dilaporkan oleh simpul atau klaster. Permintaan yang di-throttle oleh simpul atau klaster disertakan.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum
ThrottledRequestCount	<p>Jumlah total permintaan di-throttle oleh simpul atau klaster. Permintaan yang di-throttle oleh DynamoDB tidak disertakan, dan dapat dipantau menggunakan Metrik DynamoDB.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
FaultRequestCount	<p>Jumlah total permintaan yang mengakibatkan kesalahan internal yang dilaporkan oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum
FailedRequestCount	<p>Jumlah total permintaan yang mengakibatkan kesalahan yang dilaporkan oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
QueryRequestCount	<p>Jumlah permintaan kueri ditangani oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum
ScanRequestCount	<p>Jumlah permintaan pemindaian ditangani oleh simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
ClientConnections	<p>Jumlah koneksi simultan yang dibuat oleh klien ke simpul atau klaster.</p> <p>Unit: Count</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum
EstimatedDbSize	<p>Sebuah pendekatan dari jumlah data cache dalam cache item dan cache kueri oleh simpul atau klaster.</p> <p>Unit: Bytes</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average

Metrik	Deskripsi
EvictedSize	<p>Jumlah data yang dihapus oleh simpul atau kluster untuk membuat ruang untuk data yang baru diminta. Jika laju miss naik, dan Anda melihat metrik ini juga tumbuh, itu mungkin berarti bahwa set pekerjaan Anda telah meningkat. Anda harus mempertimbangkan untuk beralih ke kluster dengan jenis simpul yang lebih besar.</p> <p>Unit: Bytes</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• Sum

Metrik	Deskripsi
CPUCreditUsage	<p>Jumlah kredit CPU yang digunakan oleh simpul untuk pemanfaatan CPU. Satu kredit CPU sama dengan satu vCPU yang berjalan pada pemanfaatan 100% selama satu menit atau kombinasi yang setara dari vCPU, pemanfaatan, dan waktu (misalnya, satu vCPU yang berjalan pada pemanfaatan 50% selama dua menit atau dua vCPU yang berjalan pada pemanfaatan 25% selama dua menit).</p> <p>Metrik kredit CPU hanya tersedia dengan frekuensi lima menit. Jika Anda menentukan periode lebih dari lima menit, gunakan statistik Sum, bukan statistik Average.</p> <p>Unit: Credits (vCPU-minutes)</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
CPUCreditBalance	<p>Jumlah kredit CPU yang diperoleh yang masih harus dibayar simpul sejak diluncurkan atau dimulai.</p> <p>Kredit diakumulasi dalam saldo kredit setelah diperoleh, dan dihapus dari saldo kredit saat dibelanjakan. Saldo kredit memiliki batasan maksimum, ditentukan oleh ukuran simpul DAX. Setelah batas tercapai, setiap kredit baru yang diperoleh akan dibuang.</p> <p>Kredit dalam CPUCreditBalance tersedia untuk simpul untuk digunakan hingga melonjak di luar pemanfaatan CPU dasar.</p> <p>Unit: Credits (vCPU-minutes)</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
CPUSurplusCreditBalance	<p>Jumlah kredit surplus yang telah dibelanjakan oleh sebuah simpul DAX ketika nilai CPUCreditBalance nol.</p> <p>Nilai CPUSurplusCreditBalance dibayarkan oleh kredit CPU yang diperoleh. Jika jumlah kredit surplus melebihi jumlah kredit maksimum yang dapat diperoleh simpul dalam jangka waktu 24 jam, kredit surplus yang dibelanjakan di atas jumlah maksimum akan dikenai biaya tambahan.</p> <p>Unit: Credits (vCPU-minutes)</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Metrik	Deskripsi
CPUSurplusCreditsCharged	<p>Jumlah kredit surplus yang dibelanjakan yang tidak dibayarkan oleh kredit CPU yang diperoleh, dan dengan demikian menimbulkan biaya tambahan.</p> <p>Kredit surplus dikenai biaya saat kredit surplus yang dibelanjakan melebihi jumlah kredit maksimum yang dapat diperoleh simpul dalam periode 24 jam. Kredit surplus yang dibelanjakan di atas jumlah maksimum dibebankan pada akhir jam atau saat simpul dihapus.</p> <p>Unit: Credits (vCPU-minutes)</p> <p>Statistik Valid:</p> <ul style="list-style-type: none">• Minimum• Maximum• Average• SampleCount• Sum

Note

Metrik CPUCreditUsage, CPUCreditBalance, CPUSurplusCreditBalance, dan CPUSurplusCreditsCharged hanya tersedia untuk simpul T3.

Dimensi untuk Metrik DAX

Metrik untuk DAX memenuhi syarat oleh nilai-nilai untuk akun, ID klaster, atau IS klaster dan kombinasi ID simpul. Anda dapat menggunakan CloudWatch konsol untuk mengambil data DAX di sepanjang dimensi apa pun di tabel berikut.

Dimensi	Deskripsi
Account	Menyediakan statistik agregat di semua simpul dalam akun.
ClusterId	Membatasi data ke klaster.
ClusterId, NodeId	Membatasi data ke simpul dalam klaster.

Membuat CloudWatch alarm untuk memantau DAX

Anda dapat membuat CloudWatch alarm Amazon yang mengirimkan pesan Amazon Simple Notification Service (Amazon SNS) saat status alarm berubah. Alarm mengawasi satu metrik selama suatu periode waktu yang Anda tentukan. Alarm tersebut juga dapat melakukan satu atau beberapa tindakan berdasarkan nilai metrik relatif terhadap ambang batas selama sejumlah periode waktu. Tindakan ini adalah notifikasi yang dikirim ke topik Amazon SNS atau kebijakan Auto Scaling. Alarm memicu tindakan hanya untuk perubahan status berkelanjutan. CloudWatch alarm tidak memicu tindakan hanya karena alarm tersebut berada dalam status tertentu. Status harus diubah dan dipelihara selama jangka waktu tertentu.

Bagaimana saya bisa diberi tahu tentang cache kueri yang hilang?

1. Buat topik Amazon SNS, `arn:aws:sns:us-west-2:522194210714:QueryMissAlarm`.

Untuk informasi selengkapnya, [lihat Amazon Simple Notification Service](#) di Panduan CloudWatch Pengguna Amazon.

2. Buat alarm.

```
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. Uji alarm.

```
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

Anda dapat menambah atau mengurangi ambang batas ke salah satu yang memungkinkan untuk aplikasi Anda. Anda juga dapat menggunakan [Matematika CloudWatch Metrik](#) untuk menentukan metrik laju cache miss dan mengatur alarm untuk metrik tersebut.

Bagaimana saya bisa diberi tahu jika permintaan menyebabkan kesalahan internal di kluster?

1. Buat topik Amazon SNS, `arn:aws:sns:us-west-2:123456789012:notify-on-system-errors`.

Untuk informasi selengkapnya, [lihat Amazon Simple Notification Service](#) di Panduan CloudWatch Pengguna Amazon.

2. Buat alarm.

```
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. Uji alarm.

```
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
```

Pemantauan produksi

Anda harus menetapkan dasar untuk performa DAX normal di lingkungan Anda, dengan mengukur performa di berbagai waktu dan dengan kondisi beban yang berbeda. Saat Anda memantau DAX, Anda harus mempertimbangkan untuk menyimpan data pemantauan historis. Data yang tersimpan ini memberikan baseline untuk membandingkan data performa saat ini, mengidentifikasi pola performa normal dan anomali performa, dan merancang metode untuk mengatasi masalah.

Untuk menetapkan baseline, setidaknya Anda harus memantau item-item berikut baik saat pengujian beban dan saat produksi.

- Utilisasi CPU dan permintaan throttled, sehingga Anda dapat menentukan apakah Anda mungkin perlu menggunakan jenis simpul yang lebih besar di klaster Anda. Utilisasi CPU klaster Anda tersedia melalui CPUUtilization CloudWatch metrik.
- Latensi operasi (sebagai ukuran di sisi klien) harus tetap konsisten dalam persyaratan latensi aplikasi Anda.
- Tingkat kesalahan harus tetap rendah, seperti yang terlihat dari ErrorRequestCount, FaultRequestCount, dan FailedRequestCount CloudWatch metrik.
- Konsumsi byte jaringan, sehingga Anda dapat menentukan apakah Anda perlu menggunakan lebih banyak simpul atau jenis simpul yang lebih besar di klaster Anda. NetworkBytesIn dan

[NetworkBytesOut metrik tersedia di CloudWatch, dan Anda harus membandingkannya dengan bandwidth dasar instans yang tersedia seperti yang didokumentasikan di sini.](#)

Note

Bandwidth baseline yang tersedia yang didokumentasikan Amazon EC2 adalah dalam Gigabits per second (Gbps), di mana NetworkBytesOut metrik NetworkBytesIn dan dalam Gigabytes per menit (GBPM). Untuk mengonversi Gbps ke GBPM dan mengukur pemanfaatan, silakan kalikan bandwidth dasar dengan 7,5.

- Utilisasi memori cache dan ukuran yang dihapus, sehingga Anda dapat menentukan apakah jenis simpul kluster memiliki cukup memori untuk menahan Anda, dan jika tidak, beralih ke jenis simpul yang lebih besar.

Note

Dalam kasus sejumlah besar cache meleset dan menulis, pemanfaatan memori cache dapat meningkat hingga 100% dan dapat menyebabkan downtime ketersediaan.

- Koneksi klien, sehingga Anda dapat memantau setiap hambatan yang tidak bisa dijelaskan dalam koneksi ke kluster.

Logging operasi DAX menggunakan AWS CloudTrail

Amazon DynamoDB Accelerator (DAX) terintegrasi dengan AWS CloudTrail, yaitu layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau layanan AWS DAX.

Untuk mempelajari selengkapnya tentang DAX dan CloudTrail, lihat bagian DynamoDB Accelerator (DAX) di [Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail](#)

Instans Burstable DAX T3/T2

DAX memungkinkan Anda untuk memilih antara instans performa tetap (seperti R4 dan R5) dan instans performa burstable (seperti T2 dan T3). Instans performa burstable memberikan level baseline performa CPU dengan kemampuan untuk meningkat di atas baseline saat diperlukan.

performa baseline dan kemampuan burst diatur oleh kredit CPU. Instans performa burstable menumpuk kredit CPU secara terus menerus, pada tingkat yang ditentukan oleh ukuran instans,

ketika beban kerja di bawah ambang batas baseline. Kredit ini kemudian dapat dikonsumsi ketika beban kerja meningkat. Kredit CPU memberikan performa inti CPU penuh selama satu menit.

Banyak beban kerja tidak memerlukan CPU level tinggi secara konsisten, tetapi manfaat yang signifikan dari memiliki akses penuh ke CPU yang sangat cepat ketika mereka membutuhkannya. Instans performa burstable direkomendasikan khusus untuk kasus penggunaan ini. Jika Anda membutuhkan performa CPU yang tinggi secara konsisten untuk basis data Anda, kami sarankan Anda menggunakan instans performa tetap.

Rangkaian instans DAX T2

Instans DAX T2 adalah instans performa burstable umum yang memberikan level baseline performa CPU dengan kemampuan untuk meningkat di atas baseline. Instans T2 adalah pilihan yang baik untuk beban kerja pengujian dan pengembangan yang membutuhkan prediktabilitas harga. Instans DAX T2 dikonfigurasi untuk mode standar, yang berarti jika instans kehabisan kredit yang masih harus dibayar, pemakaian CPU secara bertahap diturunkan ke level baseline. Untuk informasi lebih lanjut tentang mode standar, lihat [Mode standar untuk instans performa burst](#) di Panduan Pengguna Amazon EC2 untuk Instans Linux.

Rangkaian instans DAX T3

Instans DAX T3 adalah jenis instans burstable umum generasi terbaru yang menyediakan level baseline performa CPU dengan kemampuan untuk meningkatkan penggunaan CPU burst selama diperlukan. Instans T3 menawarkan keseimbangan komputasi, memori, dan sumber daya jaringan dan ideal untuk beban kerja dengan penggunaan CPU moderat yang mengalami lonjakan sementara saat digunakan. Instans DAX T3 dikonfigurasi untuk mode tak terbatas, yang berarti mereka dapat di-burst melampaui baseline melalui jendela 24 jam dengan biaya tambahan. Untuk informasi lebih lanjut tentang mode tak terbatas, lihat [Mode tak terbatas untuk instans performa burst](#) di Panduan Pengguna Amazon EC2 untuk Instans Linux.

Instans DAX T3 dapat mempertahankan performa CPU yang tinggi selama beban kerja memerlukannya. Untuk sebagian besar beban kerja umum, instans T3 akan memberikan performa yang memadai tanpa biaya tambahan. Harga instans T3 per jam secara otomatis mencakup semua lonjakan sementara saat digunakan ketika pemakaian CPU rata-rata dari instans T3 berada pada atau kurang dari baseline di jendela 24 jam.

Misalnya, instans `dax.t3.small` menerima kredit terus menerus pada tingkat 24 kredit CPU per jam. Kemampuan ini memberikan performa baseline setara dengan 20% dari inti CPU ($20\% \times 60$ menit = 12 menit). Jika instans tidak menggunakan kredit yang diterima, kredit itu akan disimpan

dalam saldo kredit CPU hingga maksimum 576 kredit CPU. Saat instans `t3.small` perlu di-burst lebih dari 20% dari inti, maka instans akan menarik saldo kredit CPU untuk menangani lonjakan ini secara otomatis.

Saat instans DAX T2 dibatasi ke level performa baseline setelah saldo kredit CPU digunakan semuanya, instans DAX T3 dapat di-burst di atas baseline bahkan ketika saldo kredit CPU mereka nol. Untuk sebagian besar beban kerja, di mana penggunaan CPU rata-rata berada pada atau di bawah performa baseline, harga per jam dasar untuk `t3.small` mencakup semua burst CPU. Jika instans berjalan pada 25% utilisasi CPU rata-rata (5% di atas baseline) selama periode 24 jam setelah saldo kredit CPU digunakan semuanya, maka akan dikenakan tambahan 11,52 sen (9,6 sen/vCPU-jam \times 1 vCPU \times 5% \times 24 jam). Lihat [Perincian Amazon DynamoDB](#) untuk mendapatkan rincian harga.

Kontrol akses DAX

DynamoDB Accelerator (DAX) dirancang untuk bekerja sama dengan DynamoDB guna menambahkan lapisan caching ke aplikasi Anda dengan lancar. Namun, DAX dan DynamoDB memiliki mekanisme kontrol akses terpisah. Kedua layanan menggunakan AWS Identity and Access Management (IAM) untuk menerapkan kebijakan keamanan masing-masing, tetapi model keamanan untuk DAX dan DynamoDB berbeda.

Kami sangat menyarankan agar Anda memahami kedua model keamanan, sehingga dapat menerapkan langkah-langkah keamanan yang tepat untuk aplikasi Anda yang menggunakan DAX.

Bagian ini menjelaskan mekanisme kontrol akses yang disediakan oleh DAX dan memberikan contoh kebijakan IAM yang dapat disesuaikan dengan kebutuhan Anda.

Dengan DynamoDB, Anda dapat membuat kebijakan IAM yang membatasi tindakan yang dapat dilakukan pengguna pada sumber daya DynamoDB individu. Misalnya, Anda dapat membuat peran pengguna yang hanya mengizinkan pengguna melakukan tindakan hanya baca di tabel DynamoDB tertentu. (Untuk informasi selengkapnya, lihat [Manajemen Identitas dan Akses untuk Amazon DynamoDB](#).) Sebagai perbandingan, model keamanan DAX berfokus pada keamanan klaster dan kemampuan klaster untuk melakukan tindakan API DynamoDB atas nama Anda.

Warning

Jika saat ini Anda menggunakan kebijakan dan peran IAM untuk membatasi akses ke data tabel DynamoDB, penggunaan DAX dapat merusak kebijakan tersebut. Misalnya,

pengguna dapat memiliki akses ke tabel DynamoDB melalui DAX tetapi tidak memiliki akses eksplisit ke tabel yang sama yang mengakses DynamoDB secara langsung. Untuk informasi selengkapnya, lihat [Manajemen Identitas dan Akses untuk Amazon DynamoDB](#).

DAX tidak memberlakukan pemisahan tingkat pengguna pada data di DynamoDB.

Sebaliknya, pengguna mewarisi izin kebijakan IAM klaster DAX ketika mengakses klaster tersebut. Dengan demikian, ketika mengakses tabel DynamoDB melalui DAX, kontrol akses yang berlaku adalah izin di kebijakan IAM klaster DAX. Tidak ada izin lain yang diakui.

Jika memerlukan isolasi, sebaiknya Anda membuat klaster DAX tambahan dan lingkup kebijakan IAM untuk setiap klaster. Misalnya, Anda dapat membuat beberapa klaster DAX dan mengizinkan setiap klaster mengakses hanya satu tabel.

Peran layanan IAM untuk DAX

Ketika membuat sebuah klaster DAX, Anda harus mengasosiasikan instans dengan peran IAM. Hal ini dikenal sebagai peran layanan untuk klaster.

Misalnya, Anda ingin membuat klaster DAX baru bernama DAXCluster01. Anda dapat membuat peran layanan bernama DAX ServiceRole, dan mengaitkan peran tersebut dengan DAXCluster01. Kebijakan untuk DAX ServiceRole akan menentukan tindakan DynamoDB yang dapat dilakukan DAXCluster01, atas nama pengguna yang berinteraksi dengan DAXCluster01.

Saat Anda membuat peran layanan, Anda harus menentukan hubungan kepercayaan antara DAX ServiceRole dan layanan DAX itu sendiri. Hubungan kepercayaan menentukan entitas yang dapat mengambil peran dan memanfaatkan izinnya. Berikut ini adalah contoh dokumen hubungan kepercayaan untuk DAX ServiceRole:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "dax.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Hubungan kepercayaan ini memungkinkan cluster DAX untuk mengasumsikan DAX ServiceRole dan melakukan panggilan API DynamoDB atas nama Anda.

Tindakan API DynamoDB yang diizinkan dijelaskan dalam dokumen kebijakan IAM, yang Anda lampirkan ke DAX. ServiceRole Berikut adalah contoh dokumen kebijakan.

```
{
  "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"
      ]
    }
  ]
}
```

Kebijakan ini mengizinkan DAX melakukan tindakan API DynamoDB yang diperlukan di tabel DynamoDB. Tindakan `dynamodb:DescribeTable` diperlukan agar DAX dapat mempertahankan metadata tentang tabel, dan tindakan lain adalah tindakan pembacaan dan penulisan yang dilakukan pada item dalam tabel. Tabel bernama `Books` berada di Wilayah `us-west-2` dan dimiliki oleh ID akun AWS `123456789012`.

Note

DAX mendukung mekanisme untuk mencegah masalah deputi yang membingungkan selama akses Cross-service. Untuk informasi selengkapnya, lihat [Masalah confused deputy](#) di Panduan Pengguna IAM.

Kebijakan IAM untuk mengizinkan akses kluster DAX

Setelah membuat kluster DAX, Anda perlu memberikan izin untuk pengguna sehingga pengguna dapat mengakses kluster DAX.

Misalnya, Anda ingin memberikan akses ke DAXCluster01 untuk pengguna bernama Alice. Pertama-tama Anda akan membuat kebijakan IAM (AliceAccessPolicy) yang mendefinisikan cluster DAX dan tindakan API DAX yang dapat diakses oleh penerima. Kemudian, Anda akan memberikan akses dengan melampirkan kebijakan ini ke pengguna Alice.

Dokumen kebijakan berikut memberikan akses penuh kepada penerima di DAXCluster01.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "dax:*"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
      ]
    }
  ]
}
```

Dokumen kebijakan mengizinkan akses ke kluster DAX, tetapi tidak memberikan izin DynamoDB apa pun. (Izin DynamoDB diberikan oleh peran layanan DAX).

Untuk pengguna Alice, Anda harus terlebih dahulu membuat AliceAccessPolicy dengan dokumen kebijakan yang ditampilkan sebelumnya. Kemudian, Anda akan melampirkan kebijakan tersebut pada Alice.

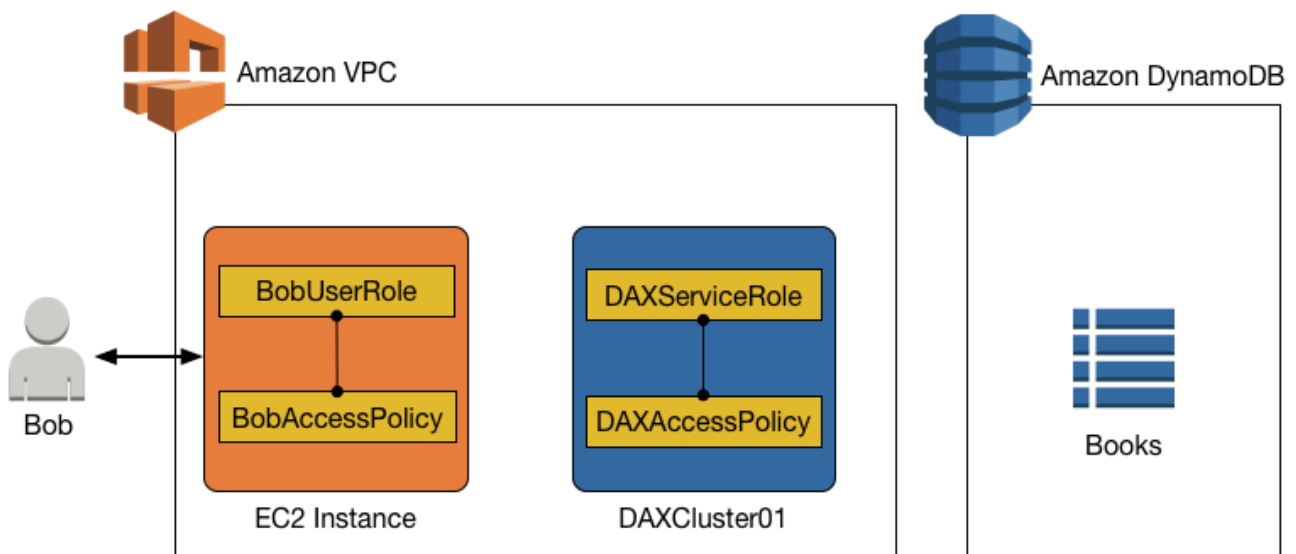
Note

Selain melampirkan kebijakan kepada pengguna, Anda dapat melampirkannya ke peran IAM. Dengan cara itu, semua pengguna yang mengambil peran tersebut akan memiliki izin yang Anda tetapkan dalam kebijakan.

Kebijakan pengguna dan peran layanan DAX menentukan sumber daya DynamoDB dan tindakan API yang dapat diakses oleh penerima melalui DAX.

Studi Kasus: Mengakses DynamoDB dan DAX

Skenario berikut dapat membantu Anda memahami lebih jauh tentang kebijakan IAM yang digunakan dengan DAX. (Skenario ini disebut di seluruh bagian ini). Diagram berikut menunjukkan gambaran umum tingkat tinggi skenario.



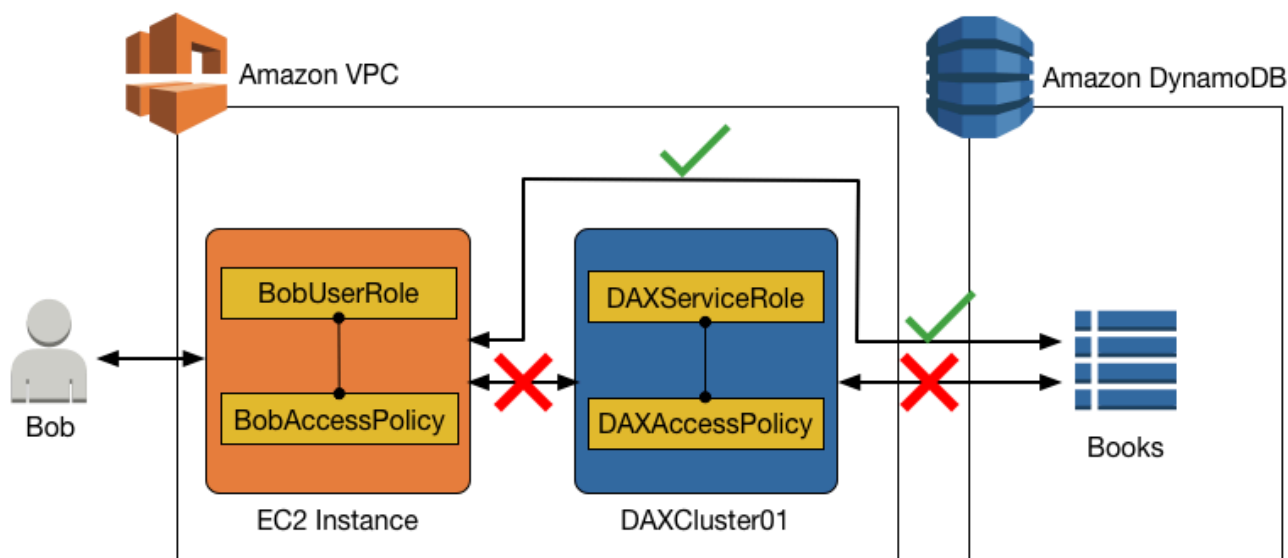
Dalam skenario ini, terdapat entitas berikut:

- Pengguna (Bob).
- Peran IAM (BobUserRole). Bob mengambil peran ini saat runtime.

- Kebijakan IAM (BobAccessPolicy). Kebijakan ini dilampirkan pada BobUserRole. BobAccessPolicy menentukan sumber daya DynamoDB dan DAX yang dapat diakses BobUserRole.
- Klaster DAX (DAXCluster01).
- Peran layanan IAM (DAXServiceRole). Peran ini mengizinkan DAXCluster01 untuk mengakses DynamoDB.
- Kebijakan IAM (DAXAccessPolicy). Kebijakan ini dilampirkan pada DAXServiceRole. DAXAccessPolicy menentukan API dan sumber daya DynamoDB yang dapat diakses DAXCluster01.
- Tabel DynamoDB (Books).

Kombinasi pernyataan kebijakan di BobAccessPolicy dan DAXAccessPolicy menentukan apa yang dapat dilakukan Bob dengan tabel Books. Misalnya, Bob mungkin dapat mengakses Books secara langsung (menggunakan titik akhir DynamoDB), secara tidak langsung (menggunakan klaster DAX), atau keduanya. Bob mungkin juga dapat membaca data dari Books, menulis data ke Books, atau keduanya.

Akses ke DynamoDB, tetapi tidak ada akses ke DAX



Mengizinkan akses langsung ke tabel DynamoDB sekaligus mencegah akses tidak langsung menggunakan kluster DAX dapat dilakukan. Untuk akses langsung ke DynamoDB, izin untuk `BobUserRole` ditentukan oleh `BobAccessPolicy` (yang terlampir pada peran).

Akses hanya baca ke DynamoDB (khusus)

Bob dapat mengakses DynamoDB dengan `BobUserRole`. Kebijakan IAM yang terlampir pada peran ini (`BobAccessPolicy`) menentukan tabel DynamoDB yang dapat diakses `BobUserRole` dan API apa yang dapat diinvokasi `BobUserRole`.

Pertimbangkan dokumen kebijakan berikut untuk `BobAccessPolicy`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DynamoDBAccessStmnt",
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
    }
  ]
}
```

Ketika dilampirkan ke `BobAccessPolicy`, dokumen ini mengizinkan `BobUserRole` untuk mengakses titik akhir DynamoDB dan melakukan operasi hanya baca pada tabel `Books`.

DAX tidak muncul dalam kebijakan ini, sehingga akses melalui DAX ditolak.

Akses baca/tulis ke DynamoDB (khusus)

Jika `BobUserRole` memerlukan akses baca/tulis ke DynamoDB, kebijakan berikut akan berfungsi.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

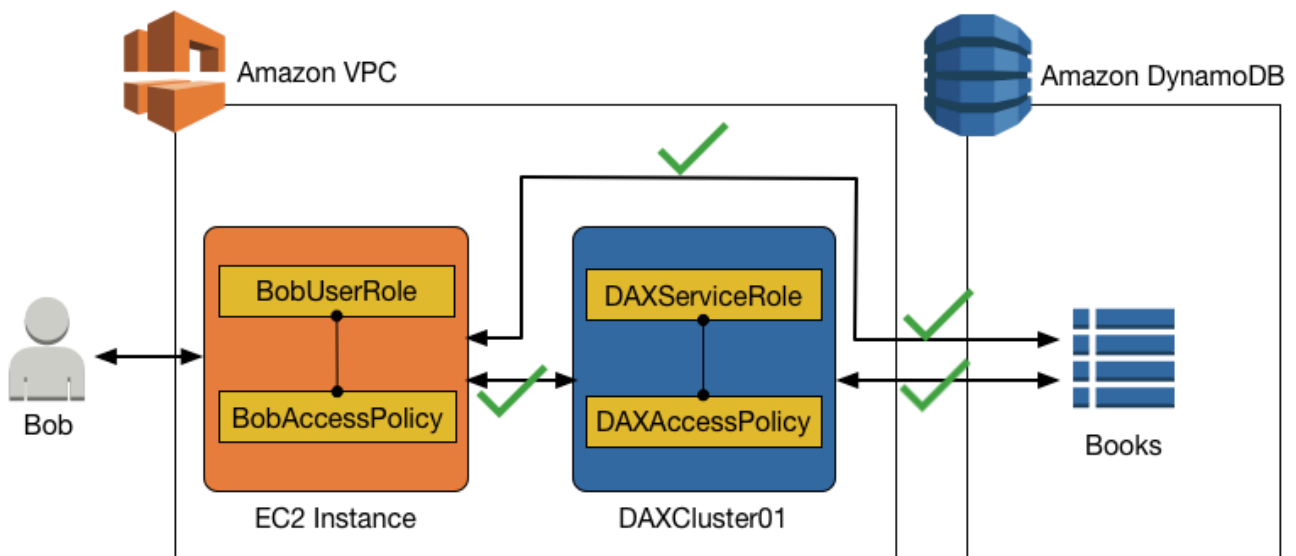
```

{
  "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"
}

```

Sekali lagi, DAX tidak muncul dalam kebijakan ini, sehingga akses melalui DAX ditolak.

Akses ke DynamoDB dan DAX



Untuk mengizinkan akses ke klaster DAX, Anda harus menyertakan tindakan khusus DAX dalam kebijakan IAM.

Tindakan khusus DAX berikut sesuai dengan tindakan serupa yang memiliki nama serupa di API DynamoDB:

- `dax:GetItem`
- `dax:BatchGetItem`
- `dax:Query`
- `dax:Scan`
- `dax:PutItem`
- `dax:UpdateItem`
- `dax>DeleteItem`
- `dax:BatchWriteItem`
- `dax:ConditionCheckItem`

Hal yang sama berlaku untuk kunci syarat `dax:EnclosingOperation`.

Akses hanya baca ke DynamoDB dan akses hanya baca ke DAX

Anggaplah Bob membutuhkan akses hanya baca ke tabel Books, dari DynamoDB dan dari DAX. Kebijakan berikut (dilampirkan pada `BobUserRole`) memberikan akses ini.

```
{
  "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"
}
]
```

Kebijakan tersebut memiliki pernyataan untuk akses DAX (`DAXAccessStmt`) dan pernyataan lain untuk akses DynamoDB (`DynamoDBAccessStmt`). Pernyataan tersebut mengizinkan Bob mengirim permintaan `GetItem`, `BatchGetItem`, `Query`, dan `Scan` ke `DAXCluster01`.

Namun, peran layanan untuk `DAXCluster01` juga akan membutuhkan akses hanya baca ke tabel `Books` di DynamoDB. Kebijakan IAM berikut, yang terlampir pada `DAXServiceRole`, akan memenuhi persyaratan ini.

```
{
  "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"
    }
  ]
}
```

Akses baca/tulis ke DynamoDB dan hanya baca dengan DAX

Untuk peran pengguna tertentu, Anda dapat memberikan akses baca/tulis ke tabel DynamoDB sekaligus mengizinkan akses hanya baca melalui DAX.

Untuk Bob, kebijakan IAM untuk `BobUserRole` perlu mengizinkan DynamoDB membaca dan menulis tindakan pada tabel `Books`, sekaligus mendukung tindakan hanya baca melalui `DAXCluster01`.

Dokumen kebijakan contoh untuk `BobUserRole` berikut memberikan akses ini.

```
{
  "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"
    }
  ]
}
```

Selain itu, `DAXServiceRole` akan memerlukan kebijakan IAM yang mengizinkan `DAXCluster01` melakukan tindakan hanya baca pada tabel `Books`.

```
{
  "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"
}
]
```

Akses baca/tulis ke DynamoDB dan akses baca/tulis ke DAX

Sekarang anggaplah bahwa Bob memerlukan akses baca/tulis ke tabel Books, langsung dari DynamoDB atau secara tidak langsung dari DAXCluster01. Dokumen kebijakan yang dilampirkan pada BobAccessPolicy berikut memberikan akses ini.

```
{
  "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"
}
]
}

```

Selain itu, `DAXServiceRole` akan memerlukan kebijakan IAM yang mengizinkan `DAXCluster01` melakukan tindakan baca/tulis pada tabel `Books`.

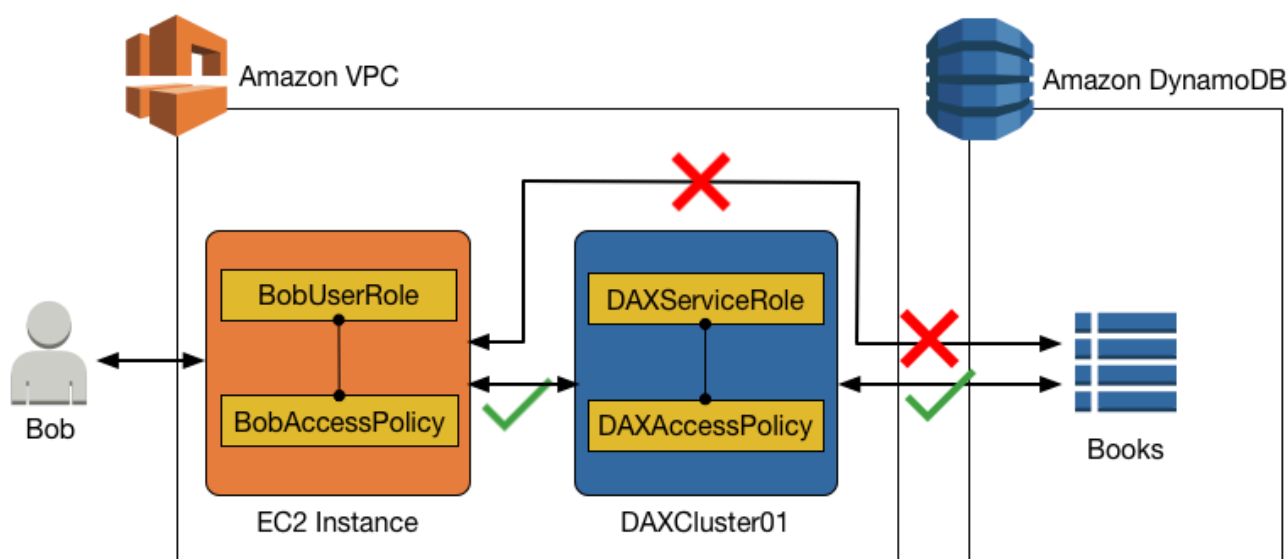
```

{
  "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"
    }
  ]
}

```


Akses ke DynamoDB melalui DAX, tetapi tidak ada akses langsung ke DynamoDB

Dalam skenario ini, Bob dapat mengakses tabel Books melalui DAX, tetapi tidak memiliki akses langsung ke tabel Books di DynamoDB. Dengan demikian, ketika mendapatkan akses ke DAX, Bob juga mendapatkan akses ke tabel DynamoDB yang mungkin tidak dapat diakses. Ketika Anda mengonfigurasi kebijakan IAM untuk peran layanan DAX, ingat bahwa setiap pengguna yang diberikan akses ke kluster DAX melalui kebijakan akses pengguna mendapatkan akses ke tabel yang ditentukan dalam kebijakan tersebut. Dalam kasus ini, `BobAccessPolicy` mendapat akses ke tabel yang ditentukan di `DAXAccessPolicy`.



Jika saat ini Anda menggunakan kebijakan dan peran IAM untuk membatasi akses ke tabel dan data DynamoDB, penggunaan DAX dapat merusak kebijakan tersebut. Dalam kebijakan berikut, Bob memiliki akses ke tabel DynamoDB melalui DAX, tetapi tidak memiliki akses langsung eksplisit ke tabel yang sama di DynamoDB.

Dokumen kebijakan berikut (`BobAccessPolicy`), yang dilampirkan pada `BobUserRole`, memberikan akses ini.

```
{
  "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"
}
```

Dalam kebijakan akses ini, tidak ada izin untuk mengakses DynamoDB secara langsung.

Bersama dengan BobAccessPolicy, DAXAccessPolicy berikut memberi BobUserRole akses ke tabel DynamoDB Books meskipun BobUserRole tidak dapat langsung mengakses tabel Books.

```
{
  "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"
    }
  ]
}
```

```
    }  
  ]  
}
```

Seperti yang ditunjukkan contoh ini, ketika Anda mengonfigurasi kontrol akses untuk kebijakan akses pengguna dan kebijakan akses klaster DAX, Anda harus sepenuhnya memahami end-to-end akses untuk memastikan bahwa prinsip hak istimewa paling sedikit dipatuhi. Pastikan juga bahwa pemberian akses ke klaster DAX kepada pengguna tidak merusak kebijakan kontrol akses yang telah ada sebelumnya.

Enkripsi DAX saat istirahat

Enkripsi Amazon DynamoDB Accelerator (DAX) saat istirahat menyediakan lapisan perlindungan data tambahan dengan membantu mengamankan data Anda dari akses tidak sah ke penyimpanan dasar. Kebijakan organisasi, peraturan industri atau peraturan pemerintah, dan persyaratan kepatuhan mungkin memerlukan penggunaan enkripsi secara tidak langsung untuk melindungi data Anda. Anda dapat menggunakan enkripsi untuk meningkatkan keamanan data aplikasi Anda yang disebar di cloud.

Dengan enkripsi saat istirahat, data yang disimpan oleh DAX pada disk dienkripsi menggunakan Standar Enkripsi Lanjutan 256-bit, juga dikenal sebagai enkripsi AES-256. DAX membuat perintah tulis data ke disk sebagai bagian dari menyebarkan perubahan dari simpul primer untuk replika baca.

Enkripsi DAX saat istirahat secara otomatis terintegrasi dengan AWS Key Management Service (AWS KMS) untuk mengelola kunci default layanan tunggal yang digunakan untuk mengenkripsi klaster Anda. Jika kunci default layanan tidak ada saat Anda membuat klaster DAX terenkripsi, AWS KMS secara otomatis membuat kunci baru terkelola AWS untuk Anda. Kunci ini digunakan dengan klaster terenkripsi yang dibuat di masa depan. AWS KMS menggabungkan perangkat keras dan perangkat lunak yang aman dan selalu tersedia untuk menyediakan sistem manajemen kunci yang disesuaikan untuk cloud.

Setelah data Anda dienkripsi, DAX menangani dekripsi data Anda secara transparan dengan dampak minimal terhadap performa. Anda tidak perlu mengubah aplikasi Anda untuk menggunakan enkripsi.

Note

DAX tidak memanggil AWS KMS untuk setiap operasi DAX tunggal. DAX hanya menggunakan kunci pada peluncuran klaster. Bahkan jika akses dicabut, DAX masih dapat

mengakses data sampai kluster dimatikan. Kunci AWS KMS yang ditentukan pelanggan tidak didukung.

Enkripsi DAX saat istirahat tersedia untuk jenis simpul kluster berikut.

Rangkaian	Tipe simpul
Memori yang dioptimalkan (R4 dan R5)	dax.r4.large
	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
Tujuan Umum (T2)	dax.t2.small
	dax.t2.medium
Tujuan Umum (T3)	dax.t3.small

Rangkaian	Tipe simpul
	dax.t3.medium

 Important

Enkripsi DAX saat istirahat tidak didukung untuk jenis simpul `dax.t3.*`.

Anda tidak dapat mengaktifkan atau menonaktifkan enkripsi saat istirahat setelah klaster dibuat. Anda harus membuat ulang klaster untuk mengaktifkan enkripsi saat istirahat jika belum diaktifkan saat dibuat.

Enkripsi DAX saat istirahat ditawarkan tanpa biaya tambahan (biaya penggunaan kunci enkripsi AWS KMS berlaku). Untuk informasi tentang harga, lihat [Harga Amazon DynamoDB](#).

Mengaktifkan enkripsi saat istirahat menggunakan AWS Management Console

Ikuti langkah berikut untuk mengaktifkan enkripsi DAX saat istirahat pada tabel menggunakan konsol.

Untuk mengaktifkan enkripsi DAX saat istirahat

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi di sisi kiri konsol, di bawah DAX, pilih Klaster.
3. Pilih Buat klaster.
4. Untuk Nama klaster, masukkan nama singkat untuk klaster Anda. Pilih jenis simpul untuk semua simpul di klaster, dan untuk ukuran klaster, gunakan simpul **3**.
5. Di Enkripsi, pastikan bahwa Aktifkan enkripsi dipilih.

Encryption

Enable encryption at rest

Protects your data while it is stored, at no additional cost. You cannot change this settings after the cluster is created. We recommend enabling encryption when possible.

Enable encryption in transit

Protects your data in transit, at no additional cost. Only the latest versions of the DAX client are compatible with encryption in transit. You cannot change this settings after the cluster is created. We recommend enabling encryption when possible.

6. Setelah memilih IAM role, grup subnet, kelompok keamanan, dan pengaturan klaster, pilih Luncurkan klaster.

Untuk mengonfirmasi bahwa klaster telah dienkripsi, periksa detail klaster di bawah panel Klaster. Enkripsi harus DIAKTIFKAN.

Enkripsi DAX dalam transit

Amazon DynamoDB Accelerator (DAX) mendukung enkripsi dalam transit data antara aplikasi Anda dan klaster DAX Anda, memungkinkan Anda untuk menggunakan DAX dalam aplikasi dengan persyaratan enkripsi yang ketat.

Terlepas dari apakah Anda memilih enkripsi dalam transit atau tidak, lalu lintas antara aplikasi Anda dan klaster DAX Anda tetap ada di Amazon VPC Anda. Lalu lintas ini diarahkan ke Antarmuka Jaringan Elastis dengan IP privat di VPC Anda yang melekat pada simpul klaster Anda. Dengan VPC Anda sebagai batas kepercayaan, Anda memiliki kontrol yang signifikan atas keamanan data Anda melalui penggunaan alat-alat standar seperti kelompok keamanan, subnet segmentasi dengan jaringan ACL, dan VPC aliran tracing. Enkripsi DAX dalam transit menambah tingkat dasar kerahasiaan, memastikan bahwa semua permintaan dan tanggapan antara aplikasi dan klaster dienkripsi oleh Keamanan Transportasi Tingkat (TLS), dan koneksi ke klaster dapat dikonfirmasi oleh verifikasi sertifikat klaster x509. Data yang ditulis ke disk oleh DAX juga dapat dienkripsi jika Anda memilih [Enkripsi saat diam](#) di saat membuat klaster DAX Anda.

Menggunakan enkripsi dalam transit dengan DAX adalah mudah. Cukup pilih opsi ini saat membuat klaster baru, dan gunakan versi terbaru dari salah satu [Klien DAX](#) di aplikasi Anda. Klaster yang menggunakan enkripsi dalam transit tidak mendukung lalu lintas yang tidak terenkripsi, jadi tidak ada kesalahan mengkonfigurasi aplikasi Anda dan enkripsi bypass. Klien DAX akan menggunakan

sertifikat klaster x509 untuk mengotentikasi identitas klaster ketika menetapkan koneksi, memastikan bahwa permintaan DAX Anda pergi ke arah yang diinginkan. Semua metode yang menciptakan klaster DAX mendukung enkripsi dalam transit: AWS Management Console, AWS CLI, semua SDK, dan AWS CloudFormation.

Enkripsi dalam transit tidak dapat diaktifkan pada klaster DAX yang sudah ada. Untuk menggunakan enkripsi dalam transit dalam aplikasi DAX yang sudah ada, buat klaster baru dengan enkripsi dalam transit diaktifkan, ubah lalu lintas aplikasi Anda padanya, lalu hapus klaster lama.

Menggunakan peran IAM tertaut layanan untuk DAX

Amazon DynamoDB Accelerator (DAX) menggunakan AWS Identity and Access Management Peran terkait layanan ([IAM](#)). Peran terkait layanan adalah jenis IAM role unik yang terhubung langsung ke DAX. Peran terkait layanan ditentukan sebelumnya oleh DAX dan mencakup semua izin yang diperlukan layanan untuk menghubungi layanan AWS lainnya atas nama Anda.

Peran terkait layanan memudahkan pengaturan DAX menjadi lebih mudah karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. DAX menentukan izin peran terkait layanan, kecuali jika ditentukan berbeda, hanya DAX yang dapat mengasumsikan perannya. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin. Kebijakan izin tidak dapat dilampirkan ke entitas IAM lainnya.

Anda dapat menghapus peran setelah duluan menghapus sumber daya terkait mereka. Hal ini akan melindungi sumber daya DAX Anda karena Anda tidak dapat secara tidak sengaja menghapus izin untuk mengakses sumber daya.

Untuk informasi tentang layanan lain yang mendukung peran terkait layanan, lihat [AWS Layanan yang Bekerja dengan IAM](#) di Panduan Pengguna IAM. Untuk beberapa layanan, lihat dan cari layanan yang memiliki Ya dalam kolom Peran terkait layanan. Pilih Ya untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Topik

- [Izin peran tertaut layanan untuk DAX](#)
- [Membuat peran tertaut layanan untuk DAX](#)
- [Mengedit peran tertaut layanan untuk DAX](#)
- [Menghapus peran tertaut layanan untuk DAX](#)

Izin peran tertaut layanan untuk DAX

DAX menggunakan peran terkait layanan bernama `AWSServiceRoleForDAX`. Peran ini memungkinkan DAX untuk memanggil layanan atas nama kluster DAX Anda.

Important

Peran terkait layanan `AWSServiceRoleForDAX` membuat Anda lebih mudah untuk mengatur dan memelihara kluster DAX. Namun, Anda masih harus memberikan setiap akses kluster ke DynamoDB sebelum Anda dapat menggunakannya. Untuk informasi selengkapnya, lihat [Kontrol akses DAX](#).

Peran terkait layanan `AWSServiceRoleForDAX` memercayakan layanan berikut untuk menjalankan peran tersebut:

- `dax.amazonaws.com`

Kebijakan izin peran memungkinkan DAX untuk menyelesaikan tindakan berikut pada sumber daya yang ditentukan:

- Tindakan pada `ec2`:
 - `AuthorizeSecurityGroupIngress`
 - `CreateNetworkInterface`
 - `CreateSecurityGroup`
 - `DeleteNetworkInterface`
 - `DeleteSecurityGroup`
 - `DescribeAvailabilityZones`
 - `DescribeNetworkInterfaces`
 - `DescribeSecurityGroups`
 - `DescribeSubnets`
 - `DescribeVpcs`
 - `ModifyNetworkInterfaceAttribute`
 - `RevokeSecurityGroupIngress`

Anda harus mengonfigurasi izin untuk mengizinkan entitas IAM (seperti pengguna, grup, atau peran) untuk membuat, menyunting, atau menghapus peran terhubung dengan layanan. Untuk informasi lebih lanjut, lihat [Izin Peran Tertaut Layanan](#) di Panduan Pengguna IAM.

Untuk memperbolehkan suatu entitas IAM untuk membuat `AWSServiceRoleForDAX` peran tertaut layanan

Tambahkan pernyataan kebijakan berikut ini ke izin untuk entitas IAM.

```
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateServiceLinkedRole"
  ],
  "Resource": "*",
  "Condition": {"StringLike": {"iam:AWSServiceName": "dax.amazonaws.com"}}
}
```

Membuat peran tertaut layanan untuk DAX

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda membuat kluster, DAX menciptakan peran terkait layanan untuk Anda.

Important

Jika Anda menggunakan layanan DAX sebelum 28 Februari 2018, saat mulai mendukung peran terkait layanan, DAX membuat `AWSServiceRoleForDAX` di akun Anda. Untuk informasi selengkapnya, lihat [Peran Baru yang Muncul di Akun AWS Saya](#) di Panduan Pengguna IAM.

Jika Anda menghapus peran terkait layanan ini, lalu ingin membuatnya lagi, Anda dapat menggunakan proses yang sama untuk membuat ulang peran tersebut di akun Anda. Saat Anda membuat instans atau kluster, DAX menciptakan peran terkait layanan untuk Anda.

Mengedit peran tertaut layanan untuk DAX

DAX tidak mengizinkan Anda untuk mengedit peran terkait layanan `AWSServiceRoleForDAX`. Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai

entitas mungkin mereferensikan peran tersebut. Namun, Anda dapat mengedit penjelasan peran menggunakan IAM. Untuk informasi selengkapnya, lihat [Mengedit Peran terkait Layanan](#) di Panduan Pengguna IAM.

Menghapus peran tertaut layanan untuk DAX

Jika Anda tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran terkait layanan, kami menyarankan Anda menghapus peran tersebut. Dengan begitu, Anda tidak memiliki entitas yang tidak digunakan yang tidak dipantau atau dipelihara secara aktif. Namun, Anda harus menghapus semua kluster DAX Anda sebelum Anda dapat menghapus peran terkait layanan.

Membersihkan peran tertaut layanan

Sebelum Anda dapat menggunakan IAM untuk menghapus peran terkait layanan, Anda harus mengonfirmasi terlebih dahulu bahwa peran tersebut tidak memiliki sesi aktif dan menghapus sumber daya yang digunakan oleh peran tersebut.

Untuk memastikan peran tertaut layanan memiliki sesi aktif di konsol IAM

1. Masuk ke AWS Management Console dan buka konsol IAM di <https://console.aws.amazon.com/iam/>.
2. Di panel navigasi konsol IAM, pilih Peran. Lalu pilih nama (bukan kotak centang) dari peran `AWSServiceRoleForDAX`.
3. Pada halaman Ringkasan untuk peran yang dipilih, pilih tab Penasihat Akses.
4. Di tab Penasihat Akses, Tinjau aktivitas terbaru untuk peran terkait layanan.

Note

Jika Anda tidak yakin apakah DAX menggunakan peran `AWSServiceRoleForDAX` tersebut, coba hapus peran tersebut. Jika layanan menggunakan peran tersebut, maka penghapusan gagal dan Anda dapat melihat Wilayah tempat peran tersebut digunakan. Jika peran tersebut digunakan, Anda harus menghapus kluster DAX sebelum dapat menghapus peran tersebut. Anda tidak dapat mencabut sesi untuk peran terkait layanan.

Jika Anda ingin menghapus peran `AWSServiceRoleForDAX`, Anda harus terlebih dahulu menghapus semua kluster DAX Anda.

Menghapus semua klaster DAX Anda

Gunakan salah satu dari prosedur ini untuk menghapus setiap klaster DAX Anda.

Untuk menghapus klaster DAX (konsol)

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi, di bawah DAX, pilih klaster.
3. Pilih Tindakan, lalu pilih Hapus.
4. Di kotak Hapus konfirmasi klaster, pilih Hapus.

Untuk menghapus klaster DAX (AWS CLI)

Lihat [hapus klaster](#) di AWS CLIREferensi Perintah.

Untuk menghapus klaster DAX (API)

Lihat [DeleteCluster](#) di dalam Referensi API Amazon DynamoDB.

Menghapus peran tertaut layanan

Untuk menghapus peran terkait layanan secara manual menggunakan IAM

Gunakan konsol IAM, CLI IAM, atau API IAM untuk menghapus peran terkait layanan `AWSserviceRoleForDAX`. Untuk informasi selengkapnya, lihat [Menghapus Peran Terkait Layanan](#) di Panduan Pengguna IAM.

Mengakses DAX di seluruh akun AWS

Bayangkan bahwa Anda memiliki klaster DynamoDB Accelerator (DAX) berjalan dalam satu AWS akun (akun A), dan klaster DAX harus dapat diakses dari instans Amazon Elastic Compute Cloud (Amazon EC2) di AWS akun (akun B) lain. Dalam tutorial ini, Anda mencapai hal ini dengan meluncurkan instans EC2 di akun B dengan IAM role dari akun B. Anda kemudian menggunakan kredensial keamanan sementara dari instans EC2 untuk menganggap IAM role dari akun A. Terakhir, Anda menggunakan kredensial keamanan sementara dari asumsi IAM role di akun A untuk membuat panggilan aplikasi melalui koneksi peering Amazon VPC ke klaster DAX di akun A. Untuk melakukan tugas-tugas ini Anda akan memerlukan akses administratif di kedua AWS akun.

⚠ Important

Cluster DAX tidak dapat mengakses tabel DynamoDB dari akun yang berbeda.

Topik

- [Mengatur IAM](#)
- [Siapkan VPC](#)
- [Ubah klien DAX untuk memungkinkan akses lintas akun](#)

Mengatur IAM

1. Membuat file teks bernama `AssumeDaxRoleTrust.json` dengan konten berikut, yang mengizinkan Amazon EC2 untuk bekerja mewakili Anda.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Di akun B, buat peran yang dapat digunakan oleh Amazon EC2 saat meluncurkan instans.

```
aws iam create-role \
  --role-name AssumeDaxRole \
  --assume-role-policy-document file://AssumeDaxRoleTrust.json
```

3. Membuat file teks bernama `AssumeDaxRolePolicy.json` dengan konten berikut, yang mengizinkan kode yang berjalan pada instans EC2 di akun B untuk menganggap IAM role di akun A. Ganti *accountA* dengan ID akun A yang sebenarnya.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::accountA:role/DaxCrossAccountRole"
  }
]
}
```

4. Tambahkan kebijakan tersebut ke peran yang baru Anda buat.

```
aws iam put-role-policy \
  --role-name AssumeDaxRole \
  --policy-name AssumeDaxRolePolicy \
  --policy-document file://AssumeDaxRolePolicy.json
```

5. Buat profil instans untuk mengizinkan instans untuk menggunakan peran.

```
aws iam create-instance-profile \
  --instance-profile-name AssumeDaxInstanceProfile
```

6. Associate peran dengan profil instans.

```
aws iam add-role-to-instance-profile \
  --instance-profile-name AssumeDaxInstanceProfile \
  --role-name AssumeDaxRole
```

7. Membuat file teks bernama `DaxCrossAccountRoleTrust.json` dengan konten berikut, yang mengizinkan akun B untuk mengambil peran akun A. Ganti *accountB* dengan ID akun B yang sebenarnya.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::accountB:role/AssumeDaxRole"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
}
```

- Di akun A, buat peran yang dapat diasumsikan oleh akun B.

```
aws iam create-role \  
  --role-name DaxCrossAccountRole \  
  --assume-role-policy-document file://DaxCrossAccountRoleTrust.json
```

- Buat file teks bernama `DaxCrossAccountPolicy.json` yang mengizinkan akses ke kluster DAX. Ganti `dax-cluster-arn` dengan Amazon Resource Name (ARN) yang benar dari cluster DAX Anda.

```
{  
  "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": "dax-cluster-arn"  
    }  
  ]  
}
```

- Di akun A, tambahkan kebijakan ke peran tersebut.

```
aws iam put-role-policy \  
  --role-name DaxCrossAccountRole \  
  --policy-name DaxCrossAccountPolicy \  
  --policy-document file://DaxCrossAccountPolicy.json
```

Siapkan VPC

1. Temukan grup subnet klaster DAX akun A. Ganti *cluster-name* dengan nama klaster DAX yang harus diakses akun B.

```
aws dax describe-clusters \  
  --cluster-name cluster-name \  
  --query 'Clusters[0].SubnetGroup'
```

2. Gunakan *Grup subnet* tersebut, lalu cari VPC klaster.

```
aws dax describe-subnet-groups \  
  --subnet-group-name subnet-group \  
  --query 'SubnetGroups[0].VpcId'
```

3. Gunakan *VPC-id* tersebut, lalu cari CIDR VPC.

```
aws ec2 describe-vpcs \  
  --vpc vpc-id \  
  --query 'Vpcs[0].CidrBlock'
```

4. Dari akun B, buat VPC menggunakan CIDR yang berbeda dan tidak tumpang tindih dari yang ditemukan di langkah sebelumnya. Kemudian, buat setidaknya satu subnet. Anda dapat menggunakan salah satu [Wizard pembuatan VPC](#) dalam AWS Management Console atau [AWS CLI](#).
5. Dari akun B, minta koneksi peering ke akun VPC seperti yang dijelaskan [dalam Membuat dan menerima koneksi peering VPC](#). Dari akun A, terima koneksi.
6. Dari akun B, temukan tabel perutean VPC baru. Ganti *vpc-id* dengan ID VPC yang Anda buat di akun B.

```
aws ec2 describe-route-tables \  
  --filters 'Name=vpc-id,Values=vpc-id' \  
  --query 'RouteTables[0].RouteTableId'
```

7. Tambahkan rute untuk mengirim lalu lintas yang dibuat untuk akun A CIDR ke koneksi peering VPC. Ingatlah untuk mengganti masing-masing *Pengganti input pengguna* dengan nilai yang benar untuk akun Anda.

```
aws ec2 create-route \  
  --route-table-id accountB-route-table-id \  
  --destination-cidr-block accountA-cidr-block \  
  --destination-prefix-list-id accountA-prefix-list-id \  
  --route-id accountB-route-id \  
  --route-type accountB-route-type \  
  --vpc-id accountB-vpc-id
```

```
--destination-cidr accountA-vpc-cidr \  
--vpc-peering-connection-id peering-connection-id
```

8. Dari akun A, temukan tabel rute kluster DAX menggunakan *vpc-id* yang Anda temukan sebelumnya.

```
aws ec2 describe-route-tables \  
--filters 'Name=vpc-id, Values=accountA-vpc-id' \  
--query 'RouteTables[0].RouteTableId'
```

9. Dari akun A, tambahkan rute untuk mengirim lalu lintas yang dibuat untuk akun B CIDR ke koneksi peering VPC. Ingatlah untuk mengganti masing-masing *Pengganti input pengguna* dengan nilai yang benar untuk akun Anda.

```
aws ec2 create-route \  
--route-table-id accountA-route-table-id \  
--destination-cidr accountB-vpc-cidr \  
--vpc-peering-connection-id peering-connection-id
```

10. Dari akun B, meluncurkan instans EC2 di VPC yang Anda buat sebelumnya. Berikan AssumeDaxInstanceProfile. Anda dapat menggunakan salah satu [launch wizard](#) dalam AWS Management Console atau [AWS CLI](#). Perhatikan grup keamanan instans.
11. Dari akun A, temukan grup keamanan yang digunakan oleh kluster DAX. Ingatlah untuk mengganti *klaster* dengan nama kluster DAX Anda.

```
aws dax describe-clusters \  
--cluster-name cluster-name \  
--query 'Clusters[0].SecurityGroups[0].SecurityGroupIdentifier'
```

12. Perbarui grup keamanan klaster DAX untuk mengizinkan lalu lintas masuk dari grup keamanan instans EC2 yang Anda buat di akun B. Ingat untuk mengganti *pengganti input pengguna* dengan nilai yang benar untuk akun Anda.

```
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
```


Pada titik ini, aplikasi pada akun B instans EC2 mampu menggunakan profil instans untuk mengasumsikan peran `arn:aws:iam::accountA-id:role/DaxCrossAccountRole` dan menggunakan kluster DAX.

Ubah klien DAX untuk memungkinkan akses lintas akun

Note

Kredensial AWS Security Token Service (AWS STS) adalah kredensial sementara. Beberapa klien menangani penyegaran secara otomatis, sementara klien lain memerlukan logika tambahan untuk menyegarkan kredensialnya. Kami menyarankan agar Anda mengikuti panduan dokumentasi yang sesuai.

Java

Bagian ini membantu Anda mengubah kode klien DAX yang ada untuk mengizinkan akses DAX lintas akun. Jika Anda tidak memiliki kode klien DAX, Anda dapat menemukan contoh kode yang dapat diandalkan di tutorial [Java dan DAX](#).

1. Tambahkan impor berikut:

```
import com.amazonaws.auth.STSAssumeRoleSessionCredentialsProvider;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import
    com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
```

2. Dapatkan penyedia kredensial dari AWS STS dan buat objek klien DAX. Ingatlah untuk mengganti masing-masing *Pengganti input pengguna* dengan nilai yang benar untuk akun Anda.

```
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

Bagian ini membantu Anda mengubah kode klien DAX yang ada untuk mengizinkan akses DAX lintas akun. Jika Anda tidak memiliki kode klien DAX, Anda dapat menemukan contoh kode yang dapat diandalkan di tutorial [.NET dan DAX](#).

1. Tambahkan [AWSSDK.SecurityToken](#) NuGet paket untuk solusinya.

```
<PackageReference Include="AWSSDK.SecurityToken" Version="latest version" />
```

2. Gunakan paket SecurityToken dan SecurityToken.Model.

```
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
```

3. Dapatkan kredensial sementara dari AmazonSimpleTokenService dan buat objek ClusterDaxClient. Ingatlah untuk mengganti masing-masing *Pengganti input pengguna* dengan nilai yang benar untuk akun Anda.

```
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

Bagian ini membantu Anda mengubah kode klien DAX yang ada untuk mengizinkan akses DAX lintas akun. Jika Anda belum memiliki kode klien DAX, Anda dapat menemukan [contoh kode yang berfungsi](#). [GitHub](#)

1. Impor AWS STS dan paket sesi.

```
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. Dapatkan kredensial sementara dari AmazonSimpleTokenService dan buat objek klien DAX. Ingatlah untuk mengganti masing-masing *Pengganti input pengguna* dengan nilai yang benar untuk akun Anda.

```
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

Bagian ini membantu Anda mengubah kode klien DAX yang ada untuk mengizinkan akses DAX lintas akun. Jika Anda tidak memiliki kode klien DAX, Anda dapat menemukan contoh kode yang dapat diandalkan di tutorial [Python dan DAX](#).

1. Impor boto3.

```
import boto3
```

2. Dapatkan kredensial sementara dari sts dan buat objek AmazonDaxClient. Ingatlah untuk mengganti masing-masing *Pengganti input pengguna* dengan nilai yang benar untuk akun Anda.

```
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

Bagian ini membantu Anda mengubah kode klien DAX yang ada untuk mengizinkan akses DAX lintas akun. Jika Anda tidak memiliki kode klien DAX, Anda dapat menemukan contoh kode yang dapat diandalkan di tutorial [Node.js dan DAX](#). Ingatlah untuk mengganti masing-masing *Pengganti input pengguna* dengan nilai yang benar untuk akun Anda.

```
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);
});
```

Panduan ukuran cluster DAX

Panduan ini memberikan saran untuk memilih pengukuran klaster Amazon DynamoDB Accelerator (DAX) dan jenis simpul yang sesuai untuk aplikasi Anda. Petunjuk ini memandu Anda melalui

langkah-langkah yang memperkirakan lalu lintas DAX aplikasi Anda, memilih konfigurasi kluster, dan mengujinya.

Jika Anda memiliki kluster DAX yang sudah ada dan ingin mengevaluasi apakah kluster memiliki jumlah dan pengukuran simpul yang sesuai, lihat [Penskalaan kluster DAX](#).

Topik

- [Gambaran Umum](#)
- [Memperkirakan lalu lintas](#)
- [Pengujian beban](#)

Gambaran Umum

Sangat penting untuk menskalakan kluster DAX Anda untuk beban kerja Anda secara tepat, baik Anda membuat kluster baru atau mempertahankan kluster yang sudah ada. Seiring berjalannya waktu dan beban kerja aplikasi Anda berubah, Anda harus secara berkala meninjau kembali keputusan penskalaan Anda untuk memastikan bahwa semuanya masih tepat.

Proses biasanya mengikuti langkah-langkah berikut:

1. **Memperkirakan lalu lintas.** Dalam langkah ini, Anda membuat prediksi tentang volume lalu lintas yang akan dikirimkan oleh aplikasi Anda ke DAX, sifat lalu lintas (operasi baca vs tulis), dan laju cache-hit yang diharapkan.
2. **Pengujian beban.** Pada langkah ini, Anda membuat sebuah kluster dan mengirim lalu lintas untuk mencerminkan perkiraan Anda dari langkah sebelumnya. Ulangi langkah ini sampai Anda menemukan konfigurasi kluster yang sesuai.
3. **Pemantauan produksi.** Saat aplikasi Anda menggunakan DAX dalam produksi, Anda harus [memantau kluster](#) untuk terus memvalidasi bahwa kluster masih melakukan penskalaan dengan benar sebagai perubahan beban kerja Anda dari waktu ke waktu.

Memperkirakan lalu lintas

Ada tiga faktor utama yang menjadi ciri beban kerja DAX yang khas:

- Laju cache-hit
- [Unit kapasitas baca](#) (RCU) per detik
- [Unit kapasitas tulis](#) (WCU) per detik

Memperkirakan tingkat hit cache

Jika Anda sudah memiliki cluster DAX, Anda dapat menggunakan [CloudWatch metrik `ItemCacheHits` dan `ItemCacheMisses` Amazon](#) untuk menentukan tingkat hit cache. Laju cache-hit sama dengan $\text{ItemCacheHits} / (\text{ItemCacheHits} + \text{ItemCacheMisses})$. Jika beban kerja Anda mencakup Query atau Scan, Anda juga harus melihat metrik `QueryCacheHits`, `QueryCacheMisses`, `ScanCacheHits`, dan `ScanCacheMisses`. Laju cache-hit bervariasi dari satu aplikasi yang lain dan sangat dipengaruhi oleh pengaturan kluster Waktu untuk Tayang (TTL). Laju temuan yang khas untuk aplikasi yang menggunakan DAX adalah 85-95 persen.

Memperkirakan unit kapasitas baca dan tulis

Jika Anda sudah memiliki tabel DynamoDB untuk aplikasi Anda, lihat [ConsumedReadCapacityUnits dan metrik `ConsumedWriteCapacityUnits` CloudWatch](#). Gunakan statistik Sum dan bagi dengan jumlah detik dalam periode tersebut.

Jika Anda juga sudah memiliki kluster DAX, ingat bahwa metrik `ConsumedReadCapacityUnits` DynamoDB hanya akun untuk yang dilewatkan oleh cache. Jadi, untuk mendapatkan ide dari unit kapasitas baca per detik ditangani oleh kluster DAX Anda, bagi nomor dengan laju kegagalan cache Anda (yaitu, $1 - \text{laju cache-hit}$).

Jika Anda belum memiliki tabel DynamoDB, lihat dokumentasi tentang [kapasitas baca unit](#) dan [unit kapasitas tulis](#) untuk memperkirakan lalu lintas Anda berdasarkan perkiraan tarif permintaan aplikasi, item yang diakses per permintaan, dan pengukuran item.

Ketika membuat perkiraan lalu lintas, rencanakan pertumbuhan di masa depan dan untuk puncak yang diharapkan dan tak terduga untuk memastikan bahwa kluster Anda memiliki ruang kosong yang cukup untuk meningkatkan lalu lintas.

Pengujian beban

Langkah berikutnya setelah memperkirakan lalu lintas adalah untuk menguji konfigurasi kluster dengan beban.

1. Untuk pengujian beban awal Anda, kami menyarankan Anda untuk memulai dengan jenis simpul `dax.r4.large`, biaya performa tetap paling murah, jenis simpul dengan memori yang dioptimalkan.
2. Sebuah kluster yang toleran terhadap kesalahan memerlukan setidaknya tiga simpul, yang tersebar di tiga Availability Zone. Dalam kasus ini, jika Availability Zone menjadi tidak tersedia,

jumlah efektif Availability Zone berkurang sepertiga. Untuk pengujian beban awal Anda, kami sarankan Anda mulai dengan klaster dua simpul, yang mensimulasikan kegagalan satu Availability Zone di klaster tiga simpul.

3. Dorong lalu lintas berkelanjutan (seperti yang diperkirakan pada langkah sebelumnya) ke klaster pengujian selama pengujian beban.
4. Memantau performa klaster selama pengujian beban.

Idealnya, profil lalu lintas yang Anda dorong selama pengujian beban harus semirip mungkin dengan lalu lintas nyata aplikasi Anda. Ini termasuk distribusi operasi (misalnya, 70 persen `GetItem`, 25 persen `Query`, dan 5 persen `PutItem`), tingkat permintaan untuk setiap operasi, jumlah item yang diakses per permintaan, dan distribusi pengukuran item. Untuk mencapai laju cache-hit yang mirip dengan laju cache-hit yang diharapkan aplikasi Anda, perhatikan distribusi utama di lalu lintas pengujian Anda.

Note

Hati-hati saat memuat pengujian jenis simpul T2 (`dax.t2.small` dan `dax.t2.medium`). Jenis simpul T2 menyediakan [performa CPU yang burstable](#) yang bervariasi dari waktu ke waktu tergantung pada saldo kredit CPU simpul. Sebuah klaster DAX berjalan pada simpul T2 mungkin tampak beroperasi secara normal, tetapi jika setiap simpul mengalami bursting di atas [performa baseline](#) instansnya, simpul akan menghabiskan saldo kredit CPU yang masih harus dibayar. Ketika saldo kredit semakin berkurang, [performa secara bertahap akan diturunkan](#) ke level performa baseline.

[Pantau klaster DAX Anda](#) selama pengujian beban untuk menentukan apakah jenis simpul yang Anda gunakan untuk pengujian beban adalah jenis simpul yang tepat untuk Anda. Selain itu, selama pengujian beban, Anda harus memantau laju permintaan dan laju cache-hit untuk memastikan bahwa infrastruktur pengujian Anda benar-benar mendorong jumlah lalu lintas yang Anda inginkan.

Anda harus memperhatikan konsumsi byte jaringan dari jenis instance cluster yang Anda pilih. Melebihi bandwidth dasar yang tersedia untuk instans Amazon EC2 menunjukkan bahwa klaster Anda mungkin tidak menopang beban kerja aplikasi Anda, dan perlu diskalakan.

Jika pengujian beban menunjukkan bahwa konfigurasi klaster yang dipilih tidak dapat mempertahankan beban kerja aplikasi Anda, Anda harus [beralih ke tipe node yang lebih besar](#), terutama jika Anda melihat pemanfaatan CPU yang tinggi pada node utama di cluster, tingkat

penggusuran tinggi, atau pemanfaatan memori cache yang tinggi. Jika Laju temuan secara konsisten tinggi, dan rasio lalu lintas baca ke tulis tinggi, Anda mungkin ingin mempertimbangkan [menambahkan lebih banyak simpul ke kluster Anda](#). Lihat [Penskalaan kluster DAX](#) untuk mendapat panduan tambahan tentang kapan menggunakan jenis simpul yang lebih besar (penskalaan vertikal) atau menambahkan lebih banyak simpul (penskalaan horizontal).

Anda harus mengulangi pengujian beban Anda setelah membuat perubahan konfigurasi kluster Anda.

Referensi API DAX DAX

Untuk informasi lebih lanjut tentang Amazon DynamoDB Accelerator (DAX), lihat di sini [Amazon DynamoDB Accelerator](#) di Referensi API Amazon DynamoDB.

Pemodelan data untuk tabel DynamoDB

Sebelum kita menyelami pemodelan data, penting untuk memahami beberapa dasar DynamoDB. DynamoDB adalah basis data NoSQL kunci-nilai yang memungkinkan skema fleksibel. Kumpulan atribut data selain atribut kunci untuk setiap item dapat berupa seragam atau diskrit. Skema kunci DynamoDB adalah dalam bentuk kunci primer sederhana di mana kunci partisi secara unik mengidentifikasi item, atau dalam bentuk kunci primer komposit di mana kombinasi kunci partisi dan kunci sortir secara unik mendefinisikan item. Kunci partisi di-hash untuk menentukan lokasi fisik data dan mengambilnya. Oleh karena itu, penting untuk memilih kardinalitas tinggi dan atribut yang dapat diskalkan secara horizontal sebagai kunci partisi untuk memastikan distribusi data yang merata. Atribut kunci sortir adalah opsional dalam skema kunci dan memiliki kunci pengurutan memungkinkan pemodelan one-to-many hubungan dan membuat koleksi item di DynamoDB. Kunci sortir juga disebut sebagai tombol rentang — mereka digunakan untuk mengurutkan item dalam koleksi item dan juga memungkinkan operasi berbasis rentang yang fleksibel.

Untuk detail selengkapnya dan praktik terbaik tentang skema kunci DynamoDB, Anda dapat merujuk ke yang berikut:

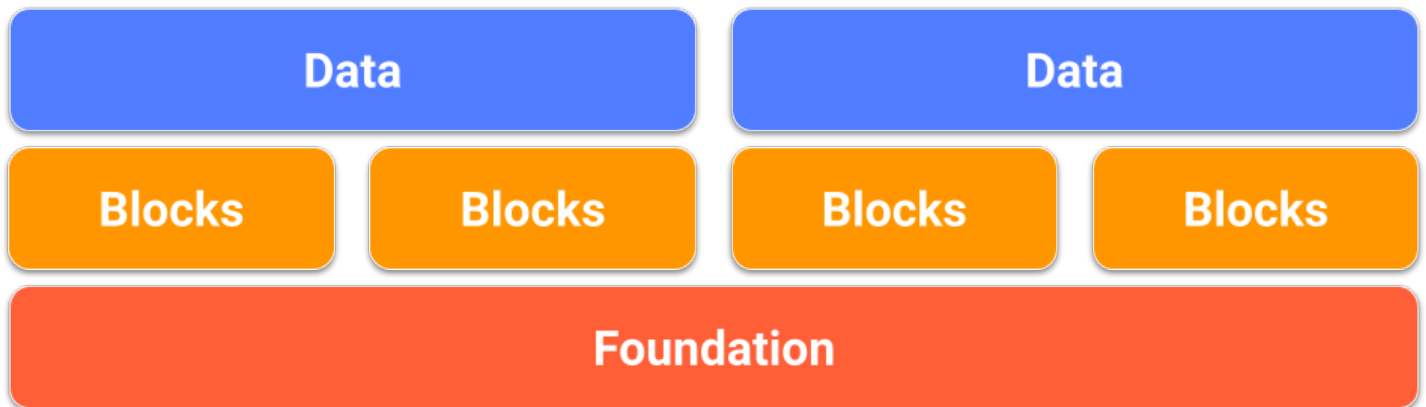
- [the section called “Partisi dan distribusi data”](#)
- [the section called “Desain kunci partisi”](#)
- [the section called “Desain kunci urutan”](#)
- [Memilih kunci partisi DynamoDB yang tepat](#)

Indeks sekunder sering diperlukan untuk mendukung pola kueri tambahan di DynamoDB. Indeks sekunder adalah tabel bayangan di mana data yang sama diatur melalui skema kunci yang berbeda dibandingkan dengan tabel dasar. Indeks sekunder lokal (LSI) berbagi kunci partisi yang sama dengan tabel dasar dan memungkinkan memiliki kunci pengurutan alternatif yang memungkinkannya untuk berbagi kapasitas tabel dasar. Indeks sekunder global (GSI) dapat memiliki kunci partisi yang berbeda serta atribut kunci pengurutan yang berbeda dari tabel dasar yang berarti manajemen throughput untuk GSI tidak tergantung pada tabel dasar.

Untuk detail lebih lanjut tentang indeks sekunder dan praktik terbaik, Anda dapat merujuk ke yang berikut:

- [the section called “Bekerja dengan indeks”](#)
- [the section called “Indeks sekunder”](#)

Sekarang mari kita lihat pemodelan data sedikit lebih dekat. Proses merancang skema yang fleksibel dan sangat dioptimalkan pada DynamoDB, atau database NoSQL apa pun dalam hal ini, dapat menjadi keterampilan yang menantang untuk dipelajari. Tujuan dari modul ini adalah untuk membantu Anda mengembangkan diagram alur mental untuk merancang skema yang akan membawa Anda dari kasus penggunaan ke produksi. Kami akan mulai dengan pengantar pilihan dasar dari desain apa pun, tabel tunggal versus beberapa desain tabel. Kemudian kami akan meninjau banyak pola desain (blok bangunan) yang dapat digunakan untuk mencapai berbagai hasil organisasi atau kinerja untuk aplikasi Anda. Akhirnya, kami menyertakan berbagai paket desain skema lengkap untuk berbagai kasus penggunaan dan industri.

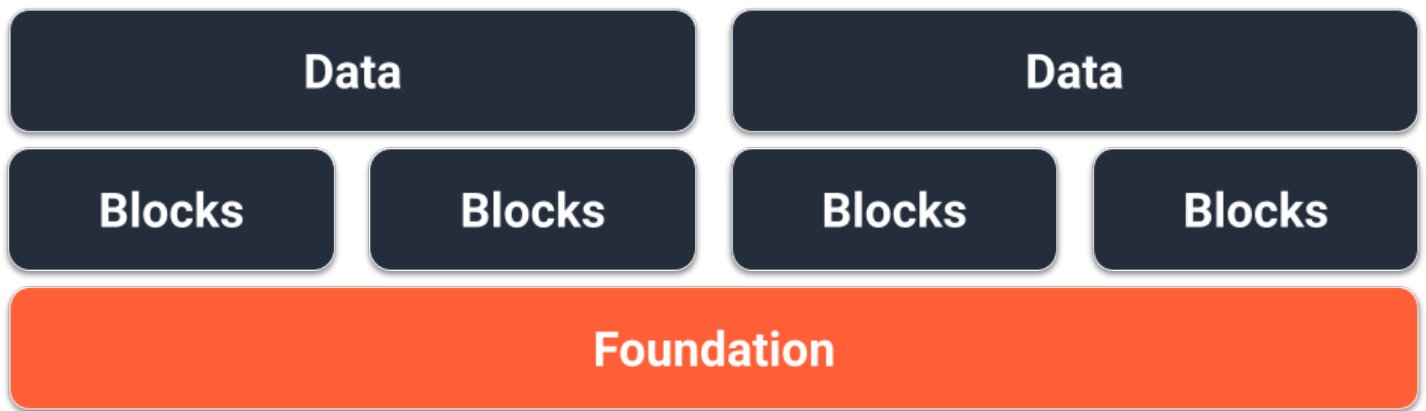


Topik

- [Fondasi Pemodelan Data di DynamoDB](#)
- [Blok bangunan pemodelan data di DynamoDB](#)
- [Paket desain skema pemodelan data di DynamoDB](#)

Fondasi Pemodelan Data di DynamoDB

Bagian ini mencakup lapisan fondasi dengan memeriksa dua jenis desain tabel: tabel tunggal dan multitabel.



Fondasi desain tabel tunggal

Salah satu pilihan untuk fondasi skema DynamoDB adalah desain tabel tunggal. Desain tabel tunggal adalah pola yang memungkinkan Anda menyimpan beberapa jenis (entitas) data dalam tabel DynamoDB tunggal. Hal ini bertujuan untuk mengoptimalkan pola akses data, meningkatkan performa, dan mengurangi biaya dengan menghilangkan kebutuhan untuk memelihara beberapa tabel dan hubungan yang kompleks di antara tabel-tabel tersebut. Hal ini dimungkinkan karena DynamoDB menyimpan item dengan kunci partisi yang sama (dikenal sebagai koleksi item) pada partisi yang sama satu sama lain. Dalam desain ini, berbagai jenis data disimpan sebagai item dalam tabel yang sama, dan setiap item diidentifikasi dengan kunci urutan yang unik.

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

Keuntungan

- Lokalitas data untuk mendukung kueri untuk beberapa jenis entitas dalam satu panggilan basis data
- Mengurangi biaya keuangan dan latensi pembacaan secara keseluruhan:
 - Satu kueri untuk dua item dengan total kurang dari 4KB adalah 0,5 RCU yang pada akhirnya konsisten
 - Dua kueri untuk dua item dengan total kurang dari 4KB adalah 1 RCU yang pada akhirnya konsisten (masing-masing 0,5 RCU)
 - Waktu untuk mengembalikan dua panggilan database terpisah akan rata-rata lebih tinggi dari satu panggilan
- Mengurangi jumlah tabel untuk dikelola:
 - Izin tidak perlu dipertahankan di beberapa kebijakan atau peran IAM
 - Manajemen kapasitas untuk tabel dirata-ratakan di seluruh entitas, biasanya menghasilkan pola konsumsi yang lebih dapat diprediksi

- Pemantauan membutuhkan alarm yang lebih sedikit
- Kunci Enkripsi yang Dikelola Pelanggan hanya perlu dirotasi pada satu tabel
- Memperlancar lalu lintas ke tabel:
 - Dengan menggabungkan beberapa pola penggunaan ke tabel yang sama, penggunaan secara keseluruhan cenderung lebih lancar (seperti performa indeks saham yang cenderung lebih lancar dibandingkan dengan saham individual) yang bekerja lebih baik untuk mencapai penggunaan yang lebih tinggi dengan tabel mode yang telah disediakan

Kekurangan

- Kurva pembelajaran bisa curam karena desain paradoks dibandingkan dengan basis data relasional
- Persyaratan data harus konsisten di semua jenis entitas
 - Pencadangan semuanya atau tidak sama sekali jadi jika beberapa data tidak penting untuk misi, pertimbangkan untuk menyimpannya di tabel terpisah
 - Enkripsi tabel dibagikan di semua item. Untuk aplikasi multi-penghuni dengan persyaratan enkripsi penghuni individual, enkripsi di sisi klien akan diperlukan
 - Tabel dengan campuran data historis dan data operasional tidak akan mendapatkan banyak manfaat dengan mengaktifkan Kelas Penyimpanan Akses Jarang. Lihat informasi yang lebih lengkap di [Kelas tabel](#)
- Semua data yang diubah akan disebar ke DynamoDB Streams meskipun hanya sebagian dari entitas yang perlu diproses.
 - Berkat filter peristiwa Lambda, hal ini tidak akan memengaruhi tagihan Anda saat menggunakan Lambda, tetapi akan menjadi biaya tambahan saat menggunakan Kinesis Consumer Library
- Saat menggunakan GraphQL, desain tabel tunggal akan lebih sulit diterapkan
- Saat menggunakan klien SDK tingkat tinggi seperti [DynamoDBMapper](#) Java atau [Enhanced Client](#), akan lebih sulit untuk memproses hasil karena item dalam respons yang sama dapat dikaitkan dengan kelas yang berbeda

Kapan harus digunakan

Desain tabel tunggal adalah pola desain yang direkomendasikan untuk DynamoDB kecuali kasus penggunaan Anda akan sangat dipengaruhi oleh salah satu kekurangan di atas. Bagi sebagian

besar pelanggan, manfaat jangka panjang lebih besar daripada tantangan jangka pendeknya, yaitu merancang tabel mereka dengan cara ini.

Fondasi desain multitabel

Pilihan kedua untuk fondasi skema DynamoDB kita adalah desain multitabel. Desain multitabel adalah pola yang lebih mirip dengan desain basis data tradisional tempat Anda menyimpan satu jenis (entitas) data dalam setiap tabel DynamoDB. Data dalam setiap tabel tetap akan diatur oleh kunci partisi, sehingga performa dalam satu jenis entitas akan dioptimalkan untuk skalabilitas dan performa, tetapi kueri di beberapa tabel harus dilakukan secara independen.

Visualizer

Data model ⓘ

AWS Discussion Forum ▾

⬇ ⬆

Forum Update ^

Thread Update ▾

Aggregate view

Forum

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	

Visualizer

Data model ⓘ

AWS Discussion Forum ▾

⬇ ⬆

Forum Update ^

Thread Update ^

Aggregate view

Thread

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
Amazon DynamoDB	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

Keuntungan

- Perancangannya lebih sederhana bagi mereka yang tidak terbiasa bekerja dengan desain tabel tunggal
- Implementasi resolver GraphQL yang lebih mudah karena setiap resolver dipetakan ke satu entitas (tabel)

- Memungkinkan persyaratan data unik di berbagai jenis entitas:
 - Cadangan dapat dibuat untuk tabel individu yang sangat penting untuk misi
 - Enkripsi tabel dapat dikelola untuk setiap tabel. Untuk aplikasi multi-penghuni dengan persyaratan enkripsi penghuni individual, tabel penghuni terpisah memungkinkan setiap pelanggan memiliki kunci enkripsinya sendiri
 - Kelas Penyimpanan Akses Jarang dapat diaktifkan hanya pada tabel dengan data historis untuk merealisasikan manfaat penghematan biaya penuh. Lihat informasi yang lebih lengkap di [Kelas tabel](#)
- Setiap tabel akan memiliki aliran data perubahannya sendiri yang memungkinkan fungsi Lambda khusus dirancang untuk setiap jenis item daripada prosesor monolitik tunggal

Kekurangan

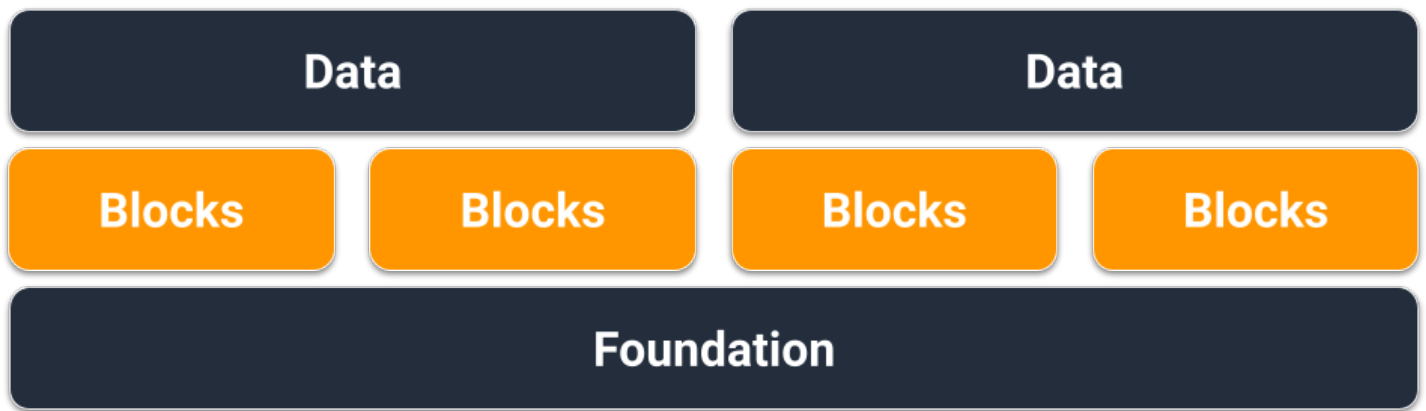
- Untuk pola akses yang memerlukan data di beberapa tabel, beberapa pembacaan dari DynamoDB akan diperlukan dan data mungkin perlu diproses/digabungkan pada kode klien.
- Operasi dan pemantauan beberapa tabel membutuhkan lebih banyak CloudWatch alarm dan setiap tabel harus diskalakan secara independen
- Setiap izin tabel perlu dikelola secara terpisah. Penambahan tabel nantinya akan memerlukan perubahan pada kebijakan atau peran IAM yang diperlukan

Kapan harus digunakan

Jika pola akses aplikasi Anda tidak memiliki kebutuhan untuk melakukan kueri untuk beberapa entitas atau tabel bersama-sama, maka desain multitabel adalah pendekatan yang baik dan memadai.

Blok bangunan pemodelan data di DynamoDB

Bagian ini membahas lapisan blok bangunan yang dapat Anda gunakan dalam pola desain untuk aplikasi Anda.



Topik

- [Blok bangunan kunci urutan komposit](#)
- [Blok bangunan multi-penghunian](#)
- [Blok bangunan indeks jarang](#)
- [Blok bangunan waktu untuk beroperasi](#)
- [Blok bangunan pengarsipan waktu untuk beroperasi](#)
- [Blok bangunan partisi vertikal](#)
- [Blok bangunan sharding penulisan](#)

Blok bangunan kunci urutan komposit

Bicara tentang NoSQL, orang-orang mungkin juga menganggapnya sebagai non-relasional. Pada akhirnya, tidak ada hal yang menghalangi relasi untuk ditempatkan ke dalam skema DynamoDB, meski tampilannya berbeda dari basis data relasional dan kunci asingnya. Salah satu pola paling kritis yang dapat kita gunakan untuk mengembangkan hierarki logis data kita di DynamoDB adalah kunci urutan komposit. Gaya yang paling umum untuk merancanginya adalah dengan memisahkan setiap lapisan hierarki (layer induk > lapisan turunan > lapisan cucu) dengan tagar. Misalnya, 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

Sementara kunci partisi di DynamoDB selalu membutuhkan nilai yang tepat untuk kueri data, kita dapat menerapkan kondisi parsial ke kunci urutan dari kiri ke kanan, mirip dengan melintasi pohon biner.

Dalam contoh di atas, ada toko e-Commerce dengan Keranjang Belanja yang perlu dipertahankan di seluruh sesi pengguna. Setiap kali pengguna masuk, mereka mungkin ingin melihat seluruh Keranjang Belanja, termasuk item yang disimpan untuk nanti. Namun ketika mereka memasuki halaman checkout, hanya item di keranjang aktif yang harus dimuat untuk dibeli. Karena kedua `KeyConditions` ini secara eksplisit meminta kunci urutan CART, data daftar keinginan tambahan diabaikan begitu saja oleh DynamoDB pada waktu baca. Meskipun item yang disimpan dan aktif adalah bagian dari keranjang yang sama, kita perlu memperlakukannya secara berbeda di berbagai bagian aplikasi, jadi menerapkan `KeyCondition` ke awalan kunci urutan adalah cara yang paling optimal untuk hanya mengambil data yang diperlukan untuk setiap bagian aplikasi.

Fitur utama dari blok bangunan ini

- Item terkait disimpan secara lokal satu sama lain untuk akses data yang efektif
- Menggunakan ekspresi `KeyCondition`, subset bagian dari hierarki dapat diambil secara selektif, yang berarti tidak ada RCU yang terbuang
- Bagian yang berbeda dari aplikasi dapat menyimpan itemnya di bawah awalan tertentu, yang mencegah penimpaan item atau penulisan yang bertentangan

Blok bangunan multi-penghunian

Banyak pelanggan menggunakan DynamoDB sebagai host data untuk aplikasi multi-penghuni mereka. Untuk skenario ini, kami ingin merancang skema dengan cara yang menyimpan semua data dari penghuni tunggal di partisi logisnya sendiri dari tabel. Hal ini memanfaatkan konsep Koleksi Item, yang merupakan istilah untuk semua item dalam tabel DynamoDB dengan kunci partisi yang sama. [Untuk informasi selengkapnya tentang cara pendekatan DynamoDB terhadap multi-penghuni, lihat Multi-penghuni di DynamoDB.](#)

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]

Untuk contoh ini, kami menjalankan situs hosting foto dengan potensi ribuan pengguna. Setiap pengguna hanya akan mengunggah foto ke profil mereka sendiri pada awalnya, tetapi secara default kami tidak akan mengizinkan pengguna untuk melihat foto pengguna lain. Tingkat isolasi tambahan idealnya akan ditambahkan ke otorisasi panggilan setiap pengguna ke API Anda untuk memastikan mereka hanya meminta data dari partisi mereka sendiri, tetapi pada tingkat skema, kunci partisi unik sudah cukup.

Fitur utama dari blok bangunan ini

- Jumlah data yang dibaca oleh satu pengguna atau penghuni hanya bisa sebanyak jumlah total item di partisi mereka
- Penghapusan data penghuni karena penutupan akun atau permintaan kepatuhan dapat dilakukan dengan taktis dan murah. Cukup jalankan kueri di mana kunci partisi sama dengan ID penyewanya, lalu jalankan operasi `DeleteItem` untuk setiap kunci primer yang dikembalikan

Note

Dirancang dengan mempertimbangkan multi-penghungan, Anda dapat menggunakan penyedia kunci enkripsi yang berbeda di satu tabel untuk mengisolasi data dengan aman. [AWS SDK Enkripsi Basis Data](#) untuk Amazon DynamoDB memungkinkan Anda menyertakan enkripsi di sisi klien dalam beban kerja DynamoDB Anda. Anda dapat melakukan enkripsi tingkat atribut, memungkinkan Anda mengenkripsi nilai atribut tertentu sebelum menyimpannya di tabel DynamoDB Anda dan mencari atribut terenkripsi tanpa mendekripsi seluruh basis data sebelumnya.

Blok bangunan indeks jarang

Terkadang pola akses memerlukan pencarian item yang cocok dengan item langka atau item yang menerima status (yang memerlukan respons yang dieskalasikan). Daripada melakukan kueri secara reguler di seluruh set data untuk item-item ini, kita dapat memanfaatkan fakta bahwa indeks sekunder global (GSI) jarang dimuat dengan data. Ini berarti bahwa hanya item dalam tabel dasar yang memiliki atribut yang ditentukan dalam indeks yang akan direplikasi ke indeks.

Primary key		Attributes		
Partition key: DeviceID	Sort key: State#Date			
df12345	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	
df54321	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	
df11223	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
		df11223	Sue

Dalam contoh ini, kita melihat kasus penggunaan IOT di mana setiap perangkat di lapangan melaporkan kembali status secara reguler. Untuk sebagian besar laporan, kita berharap perangkat

melaporkan semuanya baik-baik saja, tetapi kadang-kadang kesalahan bisa saja muncul, dan hal itu harus dieskalasikan ke teknisi perbaikan. Untuk laporan dengan eskalasi, atribut `EscalatedTo` ditambahkan item, tetapi malah tidak muncul. GSI dalam contoh ini dipartisi oleh `EscalatedTo`, dan karena GSI membawa kunci dari tabel dasar, kita masih dapat melihat `DeviceID` yang melaporkan kesalahan dan pada pukul berapa.

Meskipun pembacaan lebih murah daripada penulisan di DynamoDB, indeks jarang adalah alat yang sangat efektif untuk kasus penggunaan di mana instans jenis item tertentu jarang terjadi tetapi pembacaan untuk menemukannya merupakan hal biasa.

Fitur utama dari blok bangunan ini

- Biaya penulisan dan penyimpanan untuk GSI jarang hanya berlaku untuk item yang cocok dengan pola kunci, sehingga biaya GSI dapat jauh lebih rendah daripada GSI lain yang memiliki semua item yang direplikasi ke GSI tersebut
- Kunci urutan komposit masih dapat digunakan untuk menyaring item yang cocok dengan kueri yang diinginkan, misalnya, stempel waktu dapat digunakan untuk kunci urutan agar hanya melihat kesalahan yang dilaporkan dalam X menit terakhir (`SK > 5 minutes ago`, `ScanIndexForward: False`)

Blok bangunan waktu untuk beroperasi

Sebagian besar data memiliki beberapa durasi waktu yang dapat dianggap layak disimpan dalam penyimpanan data primer. Untuk memfasilitasi penuaan data dari DynamoDB, ada fitur yang disebut waktu untuk beroperasi (TTL). Fitur [TTL](#) memungkinkan Anda untuk menentukan atribut tertentu pada tingkat tabel yang memerlukan pemantauan untuk item dengan stempel waktu epoch (itu di masa lalu). Hal ini memungkinkan Anda untuk menghapus catatan kedaluwarsa dari tabel secara gratis.

Note

Jika Anda menggunakan [Global Tables versi 2019.11.21 \(Saat ini\)](#) dari tabel global dan Anda juga menggunakan fitur [Time to Live](#), DynamoDB mereplikasi penghapusan TTL ke semua tabel replika. Penghapusan TTL awal tidak mengonsumsi kapasitas tulis di Wilayah tempat TTL kedaluwarsa. Namun, penghapusan TTL yang direplikasi untuk tabel replika mengonsumsi unit kapasitas tulis yang direplikasi ketika menggunakan kapasitas di setiap Wilayah replika dan biaya akan berlaku.

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

Dalam contoh ini, kami memiliki aplikasi yang dirancang untuk memungkinkan pengguna membuat pesan berumur pendek. Ketika pesan dibuat di DynamoDB, atribut TTL diatur ke tanggal tujuh hari di masa depan oleh kode aplikasi. Dalam kira-kira tujuh hari, DynamoDB akan melihat bahwa stempel jangka waktu item ini sudah ada di masa lalu kemudian menghapusnya.

Karena penghapusan yang dilakukan oleh TTL gratis, sangat disarankan untuk menggunakan fitur ini guna menghapus data historis dari tabel. Ini akan mengurangi tagihan penyimpanan keseluruhan setiap bulan dan kemungkinan akan mengurangi biaya pembacaan pengguna karena data yang akan diambil oleh kueri mereka menjadi lebih sedikit. Meskipun TTL diaktifkan pada tingkat tabel, Anda bebas menentukan item atau entitas mana yang akan membuat atribut TTL dan seberapa jauh stempel jangka waktu diatur ke depannya.

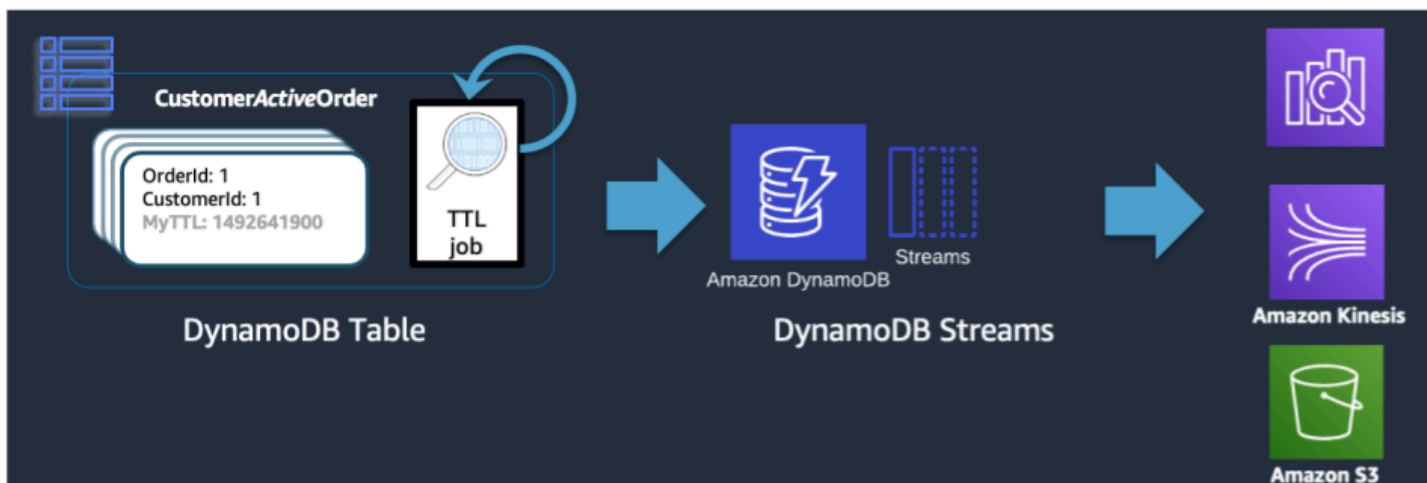
Fitur utama dari blok bangunan ini

- Penghapusan TTL dijalankan di belakang layar tanpa berdampak pada performa tabel Anda
- TTL adalah proses asinkron yang berjalan kira-kira setiap enam jam, tetapi penghapusan catatan kedaluwarsa dapat memakan waktu lebih dari 48 jam
 - Jangan mengandalkan penghapusan TTL untuk kasus penggunaan seperti catatan kunci atau manajemen status jika data basi harus dibersihkan dalam waktu kurang dari 48 jam
- Anda dapat memberi atribut TTL dengan nama atribut yang valid, tetapi nilainya harus berupa jenis angka

Blok bangunan pengarsipan waktu untuk beroperasi

Meski TTL adalah alat yang efektif untuk menghapus data lama dari DynamoDB, banyak kasus penggunaan mewajibkan arsip data disimpan untuk jangka waktu yang lebih lama dibandingkan

penyimpanan data primer. Dalam instans, kita dapat memanfaatkan penghapusan catatan berwaktu TTL untuk mendorong catatan kedaluwarsa ke dalam penyimpanan data jangka panjang.



Ketika penghapusan TTL dilakukan oleh DynamoDB, TTL masih didorong ke Aliran DynamoDB sebagai peristiwa Delete. Ketika penghapusan dilakukan oleh TTL DynamoDB, ada atribut pada catatan aliran `principal:dynamodb`. Menggunakan pelanggan Lambda ke Aliran DynamoDB, kita dapat menerapkan filter peristiwa hanya untuk atribut pengguna utama DynamoDB dan mengetahui bahwa catatan apa pun yang cocok dengan filter tersebut akan didorong ke penyimpanan arsip seperti S3 Glacier.

Fitur utama dari blok bangunan ini

- Setelah pembacaan latensi rendah DynamoDB tidak lagi diperlukan untuk item historis, memigrasikannya ke layanan penyimpanan yang lebih dingin, seperti S3 Glacier, dapat mengurangi biaya penyimpanan secara signifikan sekaligus memenuhi kebutuhan kepatuhan data dari kasus penggunaan Anda
- Jika data disimpan di Amazon S3, alat analisis hemat biaya seperti Amazon Athena atau Redshift Spectrum dapat digunakan untuk melakukan analisis historis data

Blok bangunan partisi vertikal

Pengguna yang sudah familiar dengan basis data model dokumen juga akan familier dengan gagasan menyimpan semua data terkait dalam satu dokumen JSON. Meski DynamoDB mendukung jenis data JSON, eksekusi `KeyConditions` pada JSON tertanam tidak didukung. Karena `KeyConditions` adalah hal yang menentukan berapa banyak data yang dibaca dari disk dan berapa banyak RCU yang dikonsumsi kueri secara efektif, hal ini dapat mengakibatkan inefisiensi pada skala besar. Untuk mengoptimalkan penulisan dan pembacaan DynamoDB dengan lebih baik,

kami sarankan untuk memecah entitas individual dokumen menjadi item DynamoDB individual, yang juga disebut sebagai partisi vertikal.

```
{
  "UserProfile" : {
    "FirstName": "Paul",
    "LastName": "Atreides",
    "DateJoined": "1965-08-01"},
  "Store" : {
    "store_id": "STOREUID",
    "city": "Los Angeles",
    "zip_code": "90029"}
  "ShoppingCart" : [
    {"Spice":
      { "SKU": "SpicesSKU",
        "CategoryID": "FictionalSpice",
        "DateAdded " : "2019-06-11"}},
    {"EspressoBeans":
      { "SKU": "CaffeineSKU",
        "CategoryID": "FOODANDDRINK",
        "DateAdded " : "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

Partisi vertikal, seperti yang ditunjukkan di atas, adalah contoh kunci desain tabel tunggal dalam kenyataannya, tetapi juga dapat diimplementasikan di beberapa tabel jika diinginkan. Karena tagihan DynamoDB menulis dalam kelipatan 1 KB, Anda idealnya harus mempartisi dokumen dengan cara yang menghasilkan item di bawah 1 KB.

Fitur utama dari blok bangunan ini

- Hierarki hubungan data dipertahankan melalui awalan kunci urutan agar struktur dokumen tunggal dapat dibangun kembali sisi klien jika diperlukan
- Komponen tunggal dari struktur data dapat diperbarui secara independen, sehingga item kecil dapat diperbarui meski hanya 1 WCU
- Dengan menggunakan kunci urutan BeginsWith, aplikasi dapat mengambil data serupa dalam satu kueri, yang menggabungkan biaya baca untuk mengurangi total biaya/latensi
- Dokumen besar bisa saja lebih besar dari batas ukuran item individu 400 KB di DynamoDB dan partisi vertikal membantu mengatasi batas ini

Blok bangunan sharding penulisan

Salah satu dari sedikit batasan keras yang dimiliki DynamoDB adalah pembatasan jumlah throughput yang dapat dipertahankan oleh satu partisi fisik per detik (tidak harus satu kunci partisi). Batasan ini untuk saat ini:

- 1000 WCU (atau $1000 \leq 1$ KB item ditulis per detik) dan 3000 RCU (atau $3000 \leq 4$ KB dibaca per detik) sangat konsisten atau
- $6000 \leq 4$ KB dibaca per detik pada akhirnya konsisten

Jika permintaan terhadap tabel melebihi salah satu dari batas ini, kesalahan dikirim kembali ke SDK klien `ThroughputExceededException`, yang umumnya disebut sebagai throttling. Kasus penggunaan yang memerlukan operasi baca di luar batas itu sebagian besar akan dilayani paling baik dengan menempatkan cache baca di depan DynamoDB, tetapi operasi penulisan memerlukan desain tingkat skema yang dikenal sebagai sharding penulisan.



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

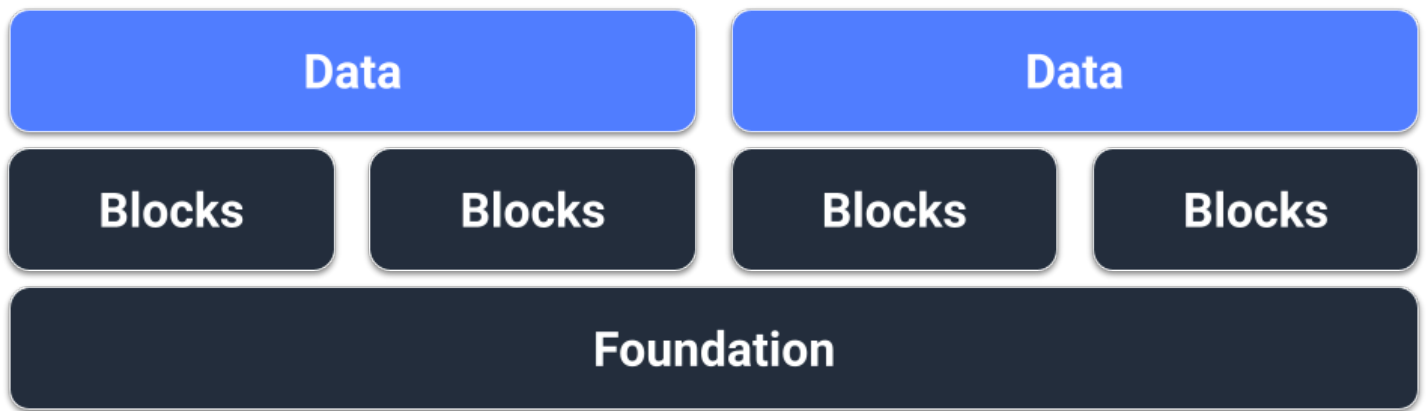
Untuk mengatasi masalah ini, kita akan menambahkan bilangan bulat acak ke akhir kunci partisi untuk setiap kontestan dalam kode `UpdateItem` aplikasi. Rentang generator bilangan bulat acak harus memiliki pencocokan batas atas atau melebihi jumlah penulisan per detik yang diharapkan untuk kontestan tertentu dibagi 1000. Untuk mendukung 20.000 suara per detik, hal ini akan terlihat seperti `rand(0,19)`. Setelah data disimpan di bawah partisi logis yang terpisah, data harus digabungkan bersama kembali pada waktu baca. Karena total suara tidak perlu real time, fungsi Lambda yang dijadwalkan untuk membaca semua partisi suara setiap X menit dapat melakukan agregasi sesekali untuk setiap kontestan dan menuliskannya kembali ke catatan total suara tunggal untuk pembacaan langsung.

Fitur utama dari blok bangunan ini

- Untuk kasus penggunaan dengan throughput tulis yang sangat tinggi untuk kunci partisi tertentu yang tidak dapat dihindari, operasi tulis dapat tersebar secara artifisial di beberapa partisi DynamoDB
- GSI dengan kunci partisi kardinalitas rendah juga harus menggunakan pola ini karena throttling pada GSI akan menerapkan tekanan balik untuk menulis operasi pada tabel dasar

Paket desain skema pemodelan data di DynamoDB

Bagian ini mencakup lapisan data untuk membahas berbagai contoh yang dapat Anda gunakan dalam desain tabel DynamoDB. Setiap contoh akan membahas kasus penggunaan, pola akses, cara mencapai pola akses, dan tampilan skema akhir.



Prasyarat

Sebelum mencoba merancang skema untuk DynamoDB, kita harus terlebih dahulu mengumpulkan beberapa data prasyarat tentang kasus penggunaan yang perlu didukung oleh skema. Tidak seperti basis data relasional, DynamoDB dipecah secara default, artinya data akan berada di beberapa server di belakang layar sehingga desain lokalitas data menjadi hal yang penting. Kita harus menyusun daftar berikut untuk setiap desain skema:

- Daftar entitas (Diagram ER)
- Estimasi volume dan throughput untuk setiap entitas
- Pola akses yang perlu didukung (kueri dan penulisan)
- Persyaratan retensi data

Topik

- [Desain skema jejaring sosial di DynamoDB](#)
- [Desain skema profil game di DynamoDB](#)
- [Desain skema sistem manajemen keluhan di DynamoDB](#)
- [Desain skema pembayaran berulang di DynamoDB](#)
- [Memantau pembaruan status perangkat di DynamoDB](#)
- [Menggunakan DynamoDB sebagai penyimpanan data untuk toko online](#)

Desain skema jejaring sosial di DynamoDB

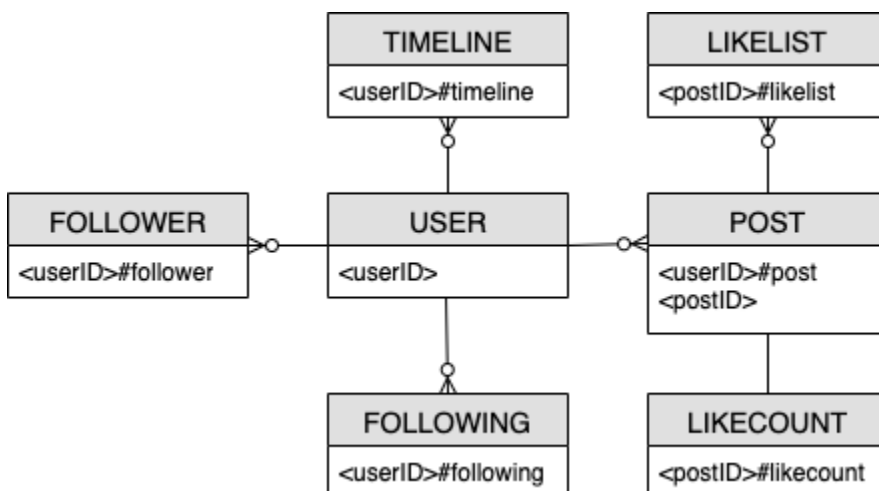
Kasus penggunaan bisnis jejaring sosial

Kasus penggunaan ini berbicara tentang penggunaan DynamoDB sebagai jejaring sosial. Jejaring sosial adalah layanan online yang memungkinkan pengguna yang berbeda berinteraksi satu sama lain. Jejaring sosial yang akan kami desain akan membiarkan pengguna melihat garis waktu yang terdiri dari posting mereka, pengikut mereka, siapa yang mereka ikuti, dan posting yang ditulis oleh siapa yang mereka ikuti. Pola akses untuk desain skema ini adalah:

- Mendapatkan informasi pengguna untuk UserID tertentu
- Dapatkan daftar pengikut untuk UserID tertentu
- Dapatkan daftar berikut untuk UserID tertentu
- Dapatkan daftar posting untuk UserID tertentu
- Dapatkan daftar pengguna yang suka posting untuk PostID tertentu
- Dapatkan hitungan seperti untuk PostID yang diberikan
- Dapatkan timeline untuk UserId tertentu

Diagram hubungan entitas jejaring sosial

Ini adalah diagram hubungan entitas (ERD) yang akan kita gunakan untuk desain skema jejaring sosial.



Pola akses jejaring sosial

Ini adalah pola akses yang akan kita pertimbangkan untuk desain skema jejaring sosial.

- `getUserInfoByUserID`
- `getFollowerListByUserID`
- `getFollowingListByUserID`
- `getPostListByUserID`
- `getUserLikesByPostID`
- `getLikeCountByPostID`
- `getTimelineByUserID`

Evolusi desain skema jejaring sosial

DynamoDB adalah database NoSQL, sehingga tidak memungkinkan Anda untuk melakukan gabungan - operasi yang menggabungkan data dari beberapa database. Pelanggan yang tidak terbiasa dengan DynamoDB mungkin menerapkan filosofi desain sistem manajemen database relasional (RDBMS) (seperti membuat tabel untuk setiap entitas) ke DynamoDB ketika mereka tidak perlu melakukannya. Tujuan dari desain tabel tunggal DynamoDB adalah untuk menulis data dalam bentuk pra-bergabung sesuai dengan pola akses aplikasi, dan kemudian segera menggunakan data tanpa perhitungan tambahan. Untuk informasi lebih lanjut, lihat [Desain meja tunggal vs multi-meja di DynamoDB](#).

Sekarang, mari kita melangkah melalui bagaimana kita akan mengembangkan desain skema kita untuk mengatasi semua pola akses.

Langkah 1: Pola akses alamat 1 (`getUserInfoByUserID`)

Untuk mendapatkan informasi pengguna tertentu, kita harus [Query](#) tabel dasar dengan kondisi kunci `PK=<userID>`. Operasi query memungkinkan Anda paginasi hasil, yang dapat berguna ketika pengguna memiliki banyak pengikut. Untuk informasi lebih lanjut tentang Query, lihat [Operasi kueri di DynamoDB](#).

Dalam contoh kami, kami melacak dua jenis data untuk pengguna kami: “count” dan “info” mereka. “Hitungan” pengguna mencerminkan berapa banyak pengikut yang mereka miliki, berapa banyak pengguna yang mereka ikuti, dan berapa banyak postingan yang telah mereka buat. “Info” pengguna mencerminkan informasi pribadi mereka seperti nama mereka.

Kita melihat dua jenis data yang diwakili oleh dua item di bawah ini. Item yang memiliki “count” dalam kunci sortir (SK) lebih cenderung berubah daripada item dengan “info.” DynamoDB mempertimbangkan ukuran item seperti yang muncul sebelum dan sesudah pembaruan dan

throughput yang disediakan yang digunakan akan mencerminkan ukuran item yang lebih besar. Jadi, bahkan jika Anda memperbarui hanya bagian dari atribut item, [UpdateItem](#) masih akan mengkonsumsi jumlah penuh throughput yang disediakan (semakin besar ukuran item sebelum dan sesudah). Anda bisa mendapatkan item melalui satu `Query` operasi dan penggunaan `UpdateItem` untuk menambah atau mengurangi dari atribut numerik yang ada.

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://...	...

Langkah 2: Pola akses alamat 2 (`getFollowerListByUserID`)

Untuk mendapatkan daftar pengguna yang mengikuti pengguna tertentu, kita harus `Query` tabel dasar dengan kondisi kunci `PK=<userID>#follower`.

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				

Langkah 3: Pola akses alamat 3 (`getFollowingListByUserID`)

Untuk mendapatkan daftar pengguna yang diberikan pengguna berikut, kita harus `Query` tabel dasar dengan kondisi kunci `PK=<userID>#following`. Anda kemudian dapat menggunakan [TransactWriteItems](#) kelompok operasi beberapa permintaan bersama-sama dan melakukan hal berikut:

- Tambahkan Pengguna A ke daftar pengikut Pengguna B, dan kemudian tingkatkan jumlah pengikut Pengguna B per satu
- Tambahkan Pengguna B ke daftar pengikut Pengguna A, dan kemudian tingkatkan jumlah pengikut Pengguna A per satu

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				

Langkah 4: Pola akses alamat 4 (**getPostListByUserID**)

Untuk mendapatkan daftar posting yang dibuat oleh pengguna tertentu, kita harus Query tabel dasar dengan kondisi kunci PK=<userID>#post. Satu hal penting yang perlu diperhatikan di sini adalah bahwa PostID pengguna harus bersifat inkremental: nilai PostID kedua harus lebih besar dari nilai PostID pertama (karena pengguna ingin melihat postingan mereka dengan cara yang diurutkan). Anda dapat melakukan ini dengan menghasilkan PostID berdasarkan nilai waktu seperti Universal Unique Lexicographically Sortable Identifier (ULID).

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	

Langkah 5: Pola akses alamat 5 (**getUserLikesByPostID**)

Untuk mendapatkan daftar pengguna yang menyukai postingan pengguna tertentu, kita harus Query tabel dasar dengan kondisi kunci PK = <postID>#likeList. Pendekatan ini adalah pola yang sama yang kita gunakan untuk mengambil pengikut dan mengikuti daftar dalam pola akses 2 (getFollowerListByUserID) dan pola akses 3 (getFollowingListByUserID).

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				

Langkah 6: Pola akses alamat 6 (getLikeCountByPostID)

Untuk mendapatkan hitungan suka untuk posting tertentu, kita harus melakukan [GetItem](#) operasi pada tabel dasar dengan kondisi kunci PK = <postID>#likecount. Pola akses ini dapat menyebabkan masalah throttling setiap kali pengguna dengan banyak pengikut (seperti selebriti) membuat posting karena throttling terjadi ketika throughput partisi melebihi 1000 WCU per detik. Masalah ini bukan hasil dari DynamoDB, itu hanya muncul di DynamoDB karena itu di akhir tumpukan perangkat lunak.

Anda harus mengevaluasi apakah itu benar-benar penting bagi semua pengguna untuk melihat hitungan seperti secara bersamaan atau apakah itu dapat terjadi secara bertahap dari waktu ke waktu. Secara umum, hitungan seperti posting tidak perlu segera 100% akurat. Anda dapat menerapkan strategi ini dengan menempatkan antrian antara aplikasi Anda dan DynamoDB agar pembaruan terjadi secara berkala.

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			

Langkah 7: Alamat akses pola 7 (**getTimelineByUserID**)

Untuk mendapatkan timeline untuk pengguna tertentu, kita harus melakukan `Query` operasi pada tabel dasar dengan kondisi kunci `PK=<userID>#timeline`. Mari pertimbangkan skenario di mana pengikut pengguna perlu melihat postingan mereka secara serempak. Setiap kali pengguna menulis posting, daftar pengikut mereka dibaca dan `userID` dan `PostID` mereka perlahan-lahan dimasukkan ke dalam kunci timeline semua pengikutnya. Kemudian, ketika aplikasi Anda dimulai, Anda dapat membaca kunci timeline dengan `Query` operasi dan mengisi layar timeline dengan kombinasi `userID` dan `PostID` menggunakan `BatchGetItem` operasi untuk setiap item baru. Anda tidak dapat membaca timeline dengan panggilan API, tetapi ini adalah solusi yang lebih hemat biaya jika posting dapat diedit sering.

Timeline adalah tempat yang menunjukkan posting terbaru, jadi kita akan membutuhkan cara untuk membersihkan yang lama. Alih-alih menggunakan `WCU` untuk menghapusnya, Anda dapat menggunakan `DynamoDB TTL` fitur untuk melakukannya secara gratis.

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			

Semua pola akses dan bagaimana skema desain alamat mereka diringkas dalam tabel di bawah ini:

Pola akses	Tabel dasar/ GSI/LSI	Operasi	Nilai kunci partisi	Urutkan nilai kunci	Kondisi lainnya/filter
getUserInfoByUserID	Tabel dasar	Kueri	PK = <userID>		
getFollowerListByUserID	Tabel dasar	Kueri	<userID>PK= #follower		

Pola akses	Tabel dasar/ GSI/LSI	Operasi	Nilai kunci partisi	Urutkan nilai kunci	Kondisi lainnya/filter
getFollowingListByUserID	Tabel dasar	Kueri	<userID>PK=#following		
getPostListByUserID	Tabel dasar	Kueri	<userID>PK=#post		
getUserLikesByPostID	Tabel dasar	Kueri	<postID>PK=#likelist		
getLikeCountByPostID	Tabel dasar	GetItem	<postID>PK=#likecount		
getTimelineByUserID	Tabel dasar	Kueri	<userID>PK=#timeline		

Skema akhir jejaring sosial

Berikut adalah desain skema akhir. Untuk mengunduh desain skema ini sebagai file JSON, lihat [Contoh DynamoDB](#) di atas GitHub.

Tabel dasar:

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			

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke [Meja Kerja NoSQL](#), alat visual yang menyediakan pemodelan data, visualisasi data, dan fitur pengembangan kueri untuk DynamoDB, untuk lebih mengeksplorasi dan mengedit proyek baru Anda. Ikuti langkah-langkah berikut untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called “Unduh”](#).
2. Unduh file skema JSON yang tercantum di atas, yang sudah ada dalam format model NoSQL Workbench.
3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called “Mengimpor model yang ada”](#).

4. Setelah Anda telah diimpor ke NOSQL Workbench, Anda dapat mengedit model data. Untuk informasi selengkapnya, lihat [the section called “Mengedit model yang ada”](#).
5. Untuk memvisualisasikan model data Anda, tambahkan data sampel, atau impor data sampel dari file CSV, gunakan [Visualizer Data](#) fitur NoSQL Workbench.

Desain skema profil game di DynamoDB

Kasus penggunaan bisnis profil game

Kasus penggunaan ini berbicara tentang penggunaan DynamoDB untuk menyimpan profil pemain untuk sistem game. Pengguna (dalam hal ini, pemain) perlu membuat profil sebelum mereka dapat berinteraksi dengan banyak game modern, terutama yang online. Profil game biasanya mencakup yang berikut:

- Informasi dasar seperti nama pengguna
- Data game seperti item dan peralatan
- Catatan permainan seperti tugas dan aktivitas
- Informasi sosial seperti daftar teman

Untuk memenuhi persyaratan akses kueri data berbutir halus untuk aplikasi ini, kunci utama (kunci partisi dan kunci sortir) akan menggunakan nama generik (PK dan SK) sehingga mereka dapat kelebihan beban dengan berbagai jenis nilai seperti yang akan kita lihat di bawah.

Pola akses untuk desain skema ini adalah:

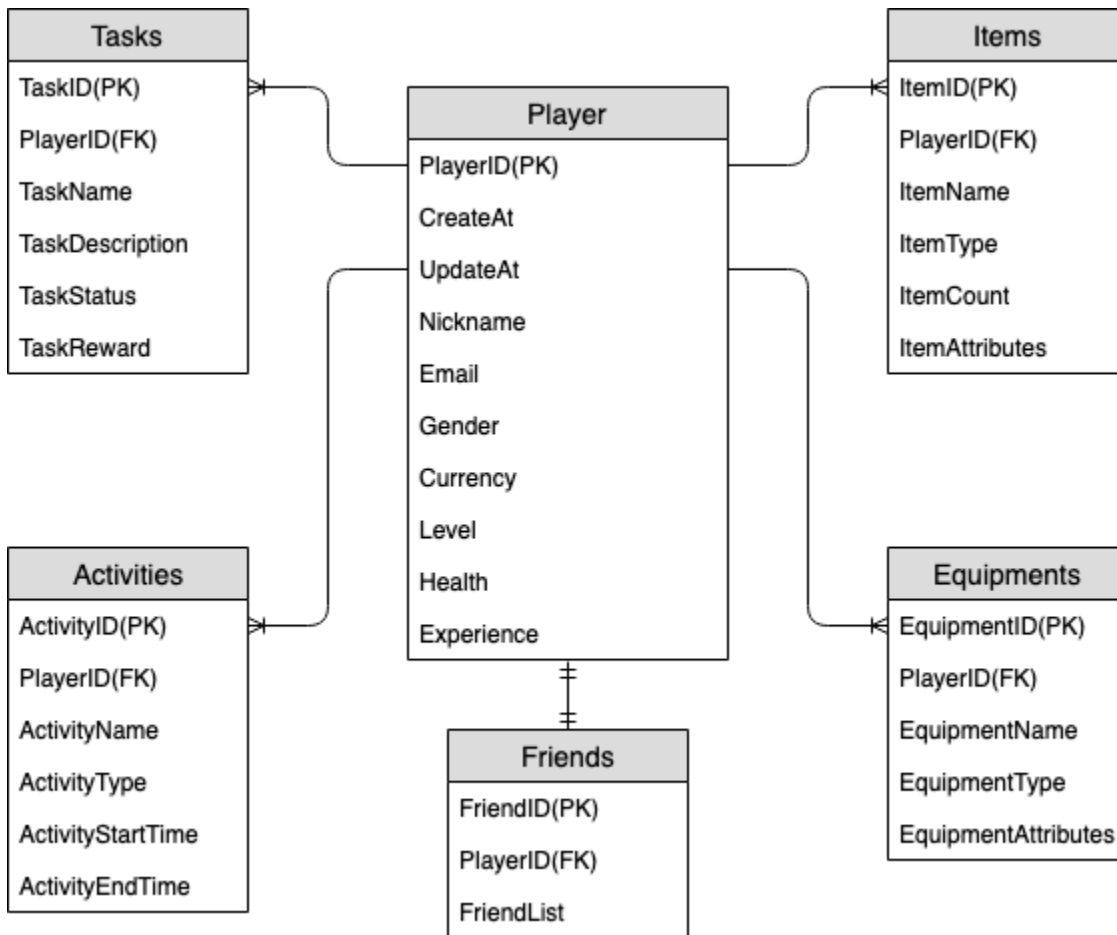
- Dapatkan daftar teman pengguna
- Dapatkan semua informasi pemain
- Dapatkan daftar item pengguna
- Dapatkan item tertentu dari daftar item pengguna
- Memperbarui karakter pengguna
- Perbarui jumlah item untuk pengguna

Ukuran profil game akan bervariasi di game yang berbeda. [Mengompresi nilai atribut besar](#) dapat membiarkan mereka masuk dalam batas item di DynamoDB dan mengurangi biaya. Strategi

manajemen throughput akan tergantung pada berbagai faktor seperti: jumlah pemain, jumlah permainan yang dimainkan per detik, dan musiman beban kerja. Biasanya untuk game yang baru diluncurkan, jumlah pemain dan tingkat popularitas tidak diketahui sehingga kita akan mulai dengan [mode throughput sesuai permintaan](#).

Diagram hubungan entitas profil game

Ini adalah diagram hubungan entitas (ERD) yang akan kita gunakan untuk desain skema profil game.



Pola akses profil game

Ini adalah pola akses yang akan kami pertimbangkan untuk desain skema jejaring sosial.

- getPlayerFriends
- getPlayerAllProfile
- getPlayerAllItems
- getPlayerSpecificItem

- `updateCharacterAttributes`
- `updateItemCount`

Evolusi desain skema profil game

Dari ERD di atas, kita dapat melihat bahwa ini adalah one-to-many jenis hubungan pemodelan data. Di DynamoDB, one-to-many Model data dapat diatur ke dalam koleksi item, yang berbeda dari database relasional tradisional di mana beberapa tabel dibuat dan ditautkan melalui kunci asing. Sebuah [koleksi barang](#) adalah sekelompok item yang memiliki nilai kunci partisi yang sama tetapi memiliki nilai kunci pengurutan yang berbeda. Dalam koleksi item, setiap item memiliki nilai kunci sortir unik yang membedakannya dari item lainnya. Dengan mengingat hal ini, mari kita gunakan pola berikut untuk `HASH` dan `RANGE` nilai untuk setiap jenis entitas.

Untuk memulai, kami menggunakan nama generik seperti `PK` dan `SK` untuk menyimpan berbagai jenis entitas dalam tabel yang sama untuk membuat model tahan masa depan. Untuk keterbacaan yang lebih baik, kita dapat menyertakan awalan untuk menunjukkan jenis data atau menyertakan atribut arbitrer yang disebut `Entity_type` atau `Type`. Dalam contoh saat ini, kita menggunakan string yang dimulai dengan `player` untuk menyimpan `player_ID` sebagai `PK`; gunakan `entity_name#` sebagai awalan dari `SK`, dan menambahkan `Type` atribut untuk menunjukkan jenis entitas mana bagian data ini. Hal ini memungkinkan kami untuk mendukung penyimpanan lebih banyak jenis entitas di masa depan, dan menggunakan teknologi canggih seperti GSI Overloading dan Sparse GSI untuk memenuhi lebih banyak pola akses.

Mari kita mulai menerapkan pola akses. Pola akses seperti menambahkan pemain dan menambahkan peralatan dapat diwujudkan melalui [PutItem](#) operasi, jadi kita bisa mengabaikannya. Dalam dokumen ini, kita akan fokus pada pola akses khas yang tercantum di atas.

Langkah 1: Pola akses alamat 1 (`getPlayerFriends`)

Kami membahas pola akses 1 (`getPlayerFriends`) dengan langkah ini. Dalam desain kami saat ini, persahabatan itu sederhana dan jumlah teman dalam gim ini kecil. Demi kesederhanaan, kami menggunakan tipe data daftar untuk menyimpan daftar teman (pemodelan 1:1). Dalam desain ini, kami menggunakan [GetItem](#) untuk memenuhi pola akses ini. Di `GetItem` operasi, kami secara eksplisit memberikan kunci partisi dan mengurutkan nilai kunci untuk mendapatkan item tertentu.

Namun, jika sebuah game memiliki banyak teman, dan hubungan di antara mereka rumit (seperti pertemanan menjadi dua arah dengan komponen undangan dan penerimaan), perlu menggunakan many-to-many hubungan untuk menyimpan setiap teman secara individual, untuk skala ke

ukuran daftar teman yang tidak terbatas. Dan jika perubahan pertemanan melibatkan operasi pada beberapa item pada saat yang sama, DynamoDB transactions dapat digunakan untuk mengelompokkan beberapa tindakan menjadi satu dan mengirimkannya sebagai satu all-or-nothing [TransactWriteItems](#) atau [TransactGetItems](#) operasi.

Primary key		Attributes	
Partition key: PK	Sort key: SK	Type	FriendList
player001	FRIENDS#player001	Friends	[{"M": {"FriendId": {"S": "player002"}, "FriendName": {"S": "Alice"}}, {"M": {"FriendId": {"S": "player003"}, "FriendName": {"S": "Bob"}}}]

Langkah 2: Pola akses alamat 2 (**getPlayerAllProfile**), 3 (**getPlayerAllItems**), dan 4 (**getPlayerSpecificItem**)

Kami membahas pola akses 2 (**getPlayerAllProfile**), 3 (**getPlayerAllItems**), dan 4 (**getPlayerSpecificItem**) menggunakan langkah ini. Kesamaan ketiga pola akses ini adalah kueri rentang, yang menggunakan [Query](#) operasi. Tergantung pada ruang lingkup kueri, [Kondisi Utamakan](#) dan [Filter Ekspresi](#) digunakan, yang biasa digunakan dalam pengembangan praktis.

Dalam operasi Query, kami memberikan nilai tunggal untuk Partition Key dan mendapatkan semua item dengan nilai Partition Key itu. Pola akses 2 (**getPlayerAllProfile**) diimplementasikan dengan cara ini. Secara opsional, kita dapat menambahkan ekspresi kondisi kunci pengurutan - string yang menentukan item yang akan dibaca dari tabel. Pola akses 3 (**getPlayerAllItems**) diimplementasikan dengan menambahkan kondisi kunci dari sort key `begins_withITEMS#`. Selanjutnya, untuk menyederhanakan pengembangan sisi aplikasi, kita dapat menggunakan ekspresi filter untuk mengimplementasikan pola akses 4 (**getPlayerSpecificItem**).

Berikut adalah contoh pseudocode menggunakan ekspresi filter yang memfilter itemWeapon kategori:

```
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

Ekspresi filter diterapkan setelah Query selesai, tetapi sebelum hasilnya dikembalikan ke klien. Oleh karena itu, Query mengkonsumsi jumlah kapasitas baca yang sama terlepas dari apakah ada ekspresi filter.

Jika pola aksesnya adalah menanyakan kumpulan data besar dan menyaring sejumlah besar data untuk menyimpan hanya sebagian kecil data, pendekatan yang tepat adalah merancang DynamoDB Partition Key dan Sort Key secara lebih efektif. Misalnya, dalam contoh di atas untuk mendapatkan tertentuItemType, jika ada banyak item untuk setiap pemain dan menanyakan untuk tertentuItemTipeadalah pola akses yang khas, akan lebih efisien untuk dibawaItemTipeke dalamSKsebagai kunci komposit. Model data akan terlihat seperti ini:ITEMS#ItemType#ItemId.

Langkah 3: Alamat pola akses 5 (**updateCharacterAttributes**) dan 6 (**updateItemCount**)

Kami membahas pola akses 5 (updateCharacterAttributes) dan 6 (updateItemCount) menggunakan langkah ini. Ketika pemain perlu memodifikasi karakter, seperti mengurangi mata uang, atau memodifikasi jumlah senjata tertentu dalam item mereka, gunakan[UpdateItem](#)untuk menerapkan pola akses ini. Untuk memperbarui mata uang pemain tetapi memastikannya tidak pernah berada di bawah jumlah minimum, kami dapat menambahkan[the section called "Ekspresi kondisi"](#)untuk mengurangi saldo hanya jika lebih besar dari atau sama dengan jumlah minimum. Berikut adalah contoh pseudocode:

```
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-blki65wn3b-gc-lob-avatar/player001.png	500 <small>Updated to 500-Amount</small>	10	80	1000

Saat mengembangkan dengan DynamoDB dan menggunakan [Penghitung Atom](#) untuk mengurangi inventaris, kami dapat memastikan idempotensi dengan menggunakan penguncian optimis. Berikut adalah contoh pseudocode untuk Penghitung Atom:

```
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 <small>Updated to 4</small>	{"M":{"HP": {"N":"50"}}	

Selain itu, dalam skenario di mana pemain membeli item dengan mata uang, seluruh proses perlu mengurangi mata uang dan menambahkan item pada saat yang sama. Kita dapat menggunakan Transaksi DynamoDB untuk mengelompokkan beberapa tindakan bersama-sama dan mengirimkannya sebagai satu all-or-nothing `TransactWriteItems` atau `TransactGetItems` operasi. `TransactWriteItems` adalah operasi penulisan sinkron dan idempoten yang mengelompokkan hingga 100 tindakan tulis menjadi satu all-or-nothing operasi. Tindakan diselesaikan secara atom sehingga baik semua dari mereka berhasil atau tidak satu pun dari mereka berhasil. Transaksi membantu menghilangkan risiko duplikasi atau lenyapnya mata uang. Untuk informasi lebih lanjut tentang transaksi, lihat [Contoh DynamoDB Transactions](#).

Semua pola akses dan bagaimana desain skema mengatasinya dirangkum dalam tabel di bawah ini:

Pola akses	Meja dasar/ GSI/LSI	Operasi	Nilai kunci partisi	Urutkan nilai kunci	Kondisi/filter lainnya
getPlayer Friends	Tabel dasar	GetItem	pk=ID pemain	SK= "TEMAN #playerID"	
getPlayer AllProfil	Tabel dasar	Kueri	pk=ID pemain		
getPlayer AllBarang	Tabel dasar	Kueri	pk=ID pemain	SK mulai_den gan "ITEMS#"	
getPlayer SpecificB arang	Tabel dasar	Kueri	pk=ID pemain	SK mulai_den gan "ITEMS#"	FilterExp ression:" ItemType =:ItemTyp e" expressio nAttribut eValues: {":itemType": "Senjata"}
updateCha racterAtt ributes	Tabel dasar	UpdateItem	pk=ID pemain	SK= "#METADAT A #playerID"	UpdateExp ression: "SET mata uang = mata uang -:jumlah" Condition Expressio n: "mata uang>=:mi nAmount"
updateIte mCount	Tabel dasar	UpdateItem	pk=ID pemain	SK = "ITEM #ItemID"	update-ex pression: "SET ItemCount

Pola akses	Meja dasar/ GSI/LSI	Operasi	Nilai kunci partisi	Urutkan nilai kunci	Kondisi/filter lainnya
					= ItemCount -:termasuk" expressio n-tribu te-values : {" :incr": {"N" : "1"}}

Skema akhir profil game

Berikut adalah desain skema akhir. Untuk mengunduh desain skema ini sebagai file JSON, lihat [DynamoDB Contoh](#) di atas GitHub.

Tabel dasar:

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-bliki65wn3bgc-lab-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					
	EQUIPMENT S#001	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Sword of the Dragon	Weapon	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}							
	EQUIPMENT S#001EQUIPMENTS#002	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Armor of the Knight	Armor	{"M":{"DEF":{"N":"100"}}							
	EQUIPMENT S#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 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"}}							

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke [NoSQL Workbench](#), alat visual yang menyediakan pemodelan data, visualisasi data, dan fitur pengembangan kueri untuk DynamoDB, untuk lebih mengeksplorasi dan mengedit proyek baru Anda. Ikuti langkah-langkah ini untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called “Unduh”](#).
2. Unduh file skema JSON yang tercantum di atas, yang sudah dalam format model NoSQL Workbench.
3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called “Mengimpor model yang ada”](#).
4. Setelah Anda mengimpor ke NoSQL Workbench, Anda dapat mengedit model data. Untuk informasi selengkapnya, lihat [the section called “Mengedit model yang ada”](#).
5. Untuk memvisualisasikan model data Anda, menambahkan data sampel, atau mengimpor data sampel dari file CSV, gunakan [Visualisasi Data](#) fitur NoSQL Workbench.

Desain skema sistem manajemen keluhan di DynamoDB

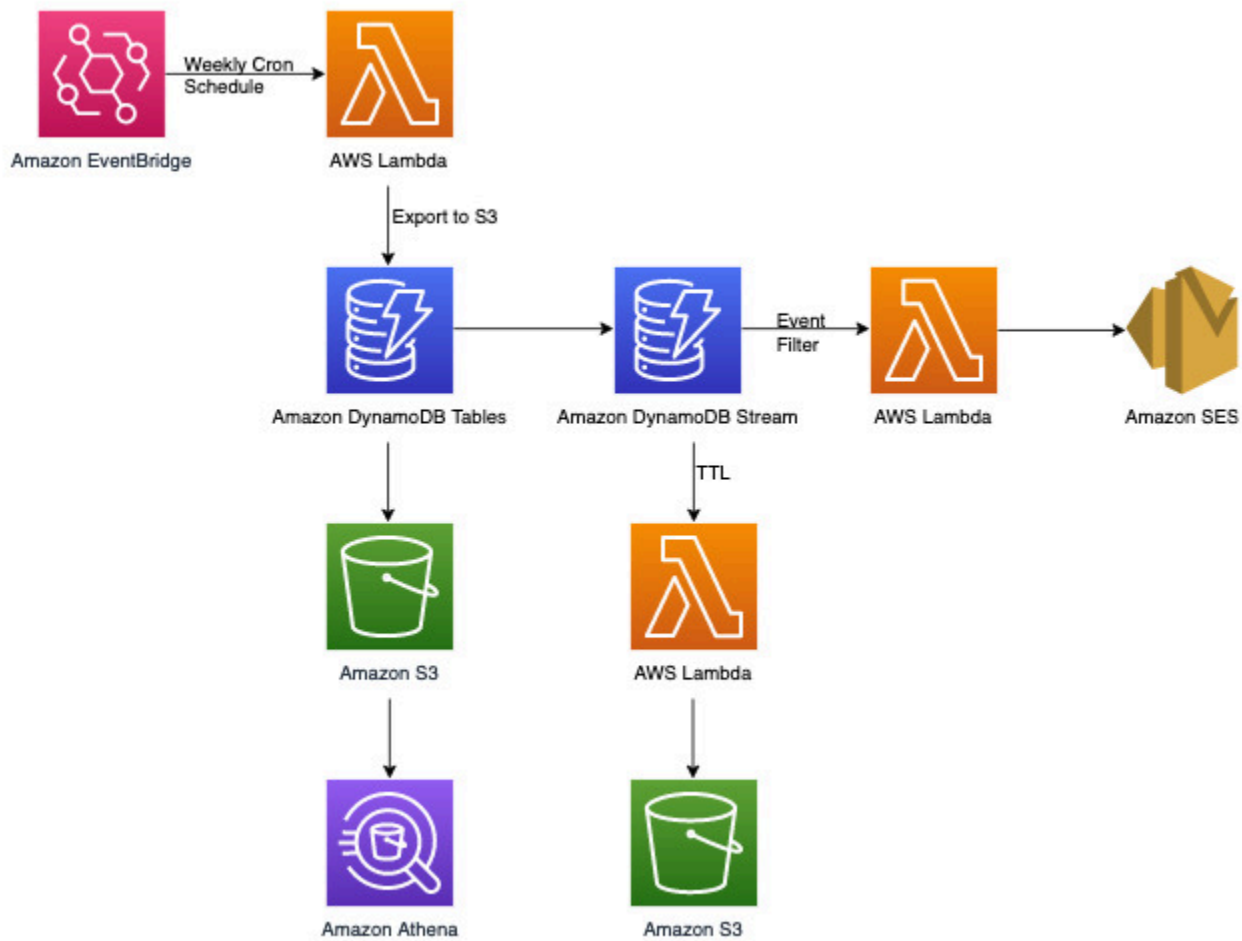
Kasus penggunaan bisnis sistem manajemen keluhan

DynamoDB adalah database yang cocok untuk sistem manajemen keluhan (atau pusat kontak) kasus penggunaan karena sebagian besar pola akses yang terkait dengan mereka akan kunci-nilai pencarian transaksional berbasis. Pola akses khas dalam skenario ini adalah:

- Membuat dan memperbarui keluhan
- Meningkatkan keluhan
- Membuat dan membaca komentar pada keluhan
- Dapatkan semua keluhan oleh pelanggan
- Dapatkan semua komentar oleh agen dan dapatkan semua eskalasi

Beberapa komentar mungkin memiliki lampiran yang menjelaskan keluhan atau solusi. Meskipun ini semua adalah pola akses kunci-nilai, mungkin ada persyaratan tambahan seperti mengirimkan pemberitahuan ketika komentar baru ditambahkan ke keluhan atau menjalankan kueri analitis untuk menemukan distribusi keluhan berdasarkan tingkat keparahan (atau kinerja agen) per minggu. Persyaratan tambahan terkait manajemen siklus hidup atau kepatuhan adalah mengarsipkan data pengaduan setelah tiga tahun mencatat keluhan.

Diagram arsitektur sistem manajemen keluhan



Terlepas dari pola akses transaksional kunci-nilai yang akan kita tangani di bagian pemodelan data DynamoDB nanti, kita memiliki tiga persyaratan non-transaksional. Diagram arsitektur di atas dapat dipecah menjadi tiga alur kerja berikut:

1. Mengirim pemberitahuan saat komentar baru ditambahkan ke keluhan
2. Menjalankan kueri analitik pada data mingguan
3. Data arsip yang lebih tua dari tiga tahun

Mari kita lihat lebih mendalam masing-masing.

Mengirim pemberitahuan saat komentar baru ditambahkan ke keluhan

Kita dapat menggunakan alur kerja di bawah ini untuk mencapai persyaratan ini:



[Aliran DynamoDB](#) adalah mekanisme perubahan data capture untuk merekam semua aktivitas menulis pada tabel DynamoDB Anda. Anda dapat mengonfigurasi fungsi Lambda untuk memicu sebagian atau semua perubahan ini. Sebuah [filter acara](#) dapat dikonfigurasi pada pemacu Lambda untuk menyaring peristiwa yang tidak relevan dengan kasus penggunaan. Dalam contoh ini, kita dapat menggunakan filter untuk memicu Lambda hanya ketika komentar baru ditambahkan dan mengirimkan pemberitahuan ke ID email yang relevan yang dapat diambil dari [AWSManajer Rahasia](#) atau toko kredensi lainnya.

Menjalankan kueri analitik pada data mingguan

DynamoDB cocok untuk beban kerja yang terutama berfokus pada pemrosesan transaksional online (OLTP). Untuk pola akses 10-20% lainnya dengan persyaratan analitis, data dapat diekspor ke S3 dengan pengelolaan [Ekspor ke Amazon S3](#) fitur tanpa dampak ke lalu lintas langsung pada tabel DynamoDB. Lihatlah alur kerja ini di bawah ini:



[Amazon EventBridge](#) dapat digunakan untuk memicu AWS Lambda sesuai jadwal - ini memungkinkan Anda untuk mengkonfigurasi ekspresi cron agar doa Lambda berlangsung secara berkala. Lambda dapat memanggil `ExportToS3API` memanggil dan menyimpan data DynamoDB di S3. Data S3 ini kemudian dapat diakses oleh mesin SQL seperti [Athena](#) untuk menjalankan kueri analitis pada data DynamoDB tanpa memengaruhi beban kerja transaksional langsung di atas meja. Contoh

permintaan Athena untuk menemukan sejumlah keluhan per tingkat keparahan akan terlihat seperti ini:

```
SELECT Item.severity.S as "Severity", COUNT(Item) as "Count"
FROM "complaint_management"."data"
WHERE NOT Item.severity.S = ''
GROUP BY Item.severity.S ;
```

Hal ini menghasilkan hasil query Athena berikut:

Results (3)

#	Severity	Count
1	P3	1
2	P2	2
3	P1	1

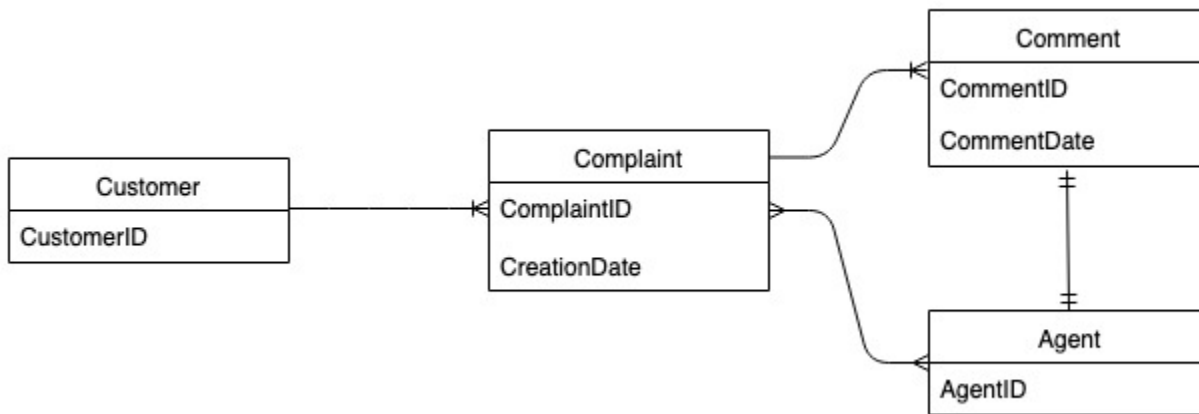
Data arsip yang lebih tua dari tiga tahun

Anda dapat memanfaatkan DynamoDB [Waktu untuk Hidup \(TTL\)](#) fitur untuk menghapus data usang dari tabel DynamoDB Anda tanpa biaya tambahan (kecuali dalam kasus replika tabel global untuk versi 2019.11.21 (Saat Ini), di mana TTL menghapus direplikasi ke Wilayah lain menggunakan kapasitas tulis). Data ini muncul dan dapat dikonsumsi dari DynamoDB Streams untuk diarsipkan ke Amazon S3. Alur kerja untuk persyaratan ini adalah sebagai berikut:



Diagram hubungan entitas sistem manajemen keluhan

Ini adalah diagram hubungan entitas (ERD) yang akan kita gunakan untuk desain skema sistem manajemen keluhan.



Pola akses sistem manajemen keluhan

Ini adalah pola akses yang akan kita pertimbangkan untuk desain skema manajemen keluhan.

1. CreateComplaint
2. UpdateComplaint
3. updateSeveritybyComplaintID
4. getComplaintByComplaintID
5. addCommentByComplaintID
6. getAllCommentsByComplaintID
7. getLatestCommentByComplaintID
8. GetaComplaintbyCustomerIDAndComplaintID
9. getAllComplaintsByCustomerID
- 10.escalateComplaintByComplaintID
- 11.getAllEscalatedKeluhan
- 12.getEscalatedComplaintsByAgentID (pesanan dari yang terbaru hingga yang tertua)
- 13.getCommentsByAgentID (antara dua tanggal)

Evolusi desain skema sistem manajemen keluhan

Karena ini adalah sistem manajemen keluhan, sebagian besar pola akses berputar di sekitar keluhan sebagai entitas utama. Yang `ComplaintID` menjadi sangat kardinal akan memastikan distribusi data yang merata di partisi yang mendasarinya dan juga merupakan kriteria pencarian yang paling umum

untuk pola akses kami yang teridentifikasi. Oleh karena itu, `ComplaintID` adalah kandidat kunci partisi yang baik dalam kumpulan data ini.

Langkah 1: Alamat pola akses 1 (`createComplaint`), 2 (`updateComplaint`), 3 (`updateSeveritybyComplaintID`), dan 4 (`getComplaintByComplaintID`)

Kita dapat menggunakan kunci sortir generik yang dihargai disebut “metadata” (atau “AA”) untuk menyimpan informasi spesifik keluhan seperti `CustomerID`, `State`, `Severity`, dan `CreationDate`. Kami menggunakan operasi tunggal dengan `PK=ComplaintID` dan `SK=“metadata”` untuk melakukan hal berikut:

1. [PutItem](#) untuk membuat keluhan baru
2. [UpdateItem](#) untuk memperbarui tingkat keparahan atau bidang lain dalam metadata keluhan
3. [GetItem](#) untuk mengambil metadata untuk keluhan

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>

Langkah 2: Pola akses alamat 5 (`addCommentByComplaintID`)

Pola akses ini membutuhkan one-to-many Model hubungan antara keluhan dan komentar pada keluhan. Kami akan menggunakan [partisi vertikal](#) teknik di sini untuk menggunakan kunci semacam dan membuat koleksi item dengan berbagai jenis data. Jika kita melihat pola akses 6 (`getAllCommentsByComplaintID`) dan 7 (`getLatestCommentByComplaintID`), kita tahu bahwa komentar perlu diurutkan berdasarkan waktu. Kami juga dapat memiliki beberapa komentar yang masuk pada saat yang sama sehingga kami dapat menggunakan [kunci sortir komposit](#) teknik untuk menambahkan waktu dan `CommentID` dalam atribut kunci semacam.

Opsi lain untuk menangani kemungkinan tabrakan komentar adalah meningkatkan perincian untuk stempel waktu atau menambahkan angka tambahan sebagai akhiran alih-alih menggunakan `Comment_ID`. Dalam kasus ini, kami akan mengawali nilai kunci sortir untuk item yang sesuai dengan komentar dengan “comm#” untuk mengaktifkan operasi berbasis rentang.

Kita juga perlu memastikan bahwa `currentState` dalam metadata keluhan mencerminkan keadaan ketika komentar baru ditambahkan. Menambahkan komentar mungkin menunjukkan bahwa keluhan telah ditugaskan ke agen atau telah diselesaikan dan sebagainya. Untuk menggabungkan

penambahan komentar dan pembaruan keadaan saat ini dalam metadata keluhan, dalam all-or-nothing cara, kita akan menggunakan `TransactWriteItems` API. Status tabel yang dihasilkan sekarang terlihat seperti ini:

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>

Mari tambahkan beberapa data lagi di tabel dan tambahkan juga `ComplaintID` sebagai bidang terpisah dari `PK` untuk pemeriksaan masa depan model jika kita membutuhkan indeks tambahan `ComplaintID`. Perhatikan juga bahwa beberapa komentar mungkin memiliki lampiran yang akan kami simpan di Amazon Simple Storage Service dan hanya mempertahankan referensi atau URL mereka di DynamoDB. Ini adalah praktik terbaik untuk menjaga basis data transaksional selingkuh mungkin untuk mengoptimalkan biaya dan kinerja. Data sekarang terlihat seperti ini:

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>

Langkah 3: Alamat pola akses 6 (`getAllCommentsByComplaintID`) dan 7 (`getLatestCommentByComplaintID`)

Untuk mendapatkan semua komentar untuk keluhan, kami dapat menggunakan `query` operasi dengan `begins_with` kondisi pada kunci semacam. Alih-alih mengonsumsi kapasitas baca tambahan untuk membaca entri metadata dan kemudian memiliki overhead penyaringan hasil yang relevan, memiliki kondisi kunci semacam seperti ini membantu kita hanya membaca apa yang kita butuhkan. Sebagai contoh, `query` operasi dengan `PK=Complaint123` dan `SKdimulai_dengancomm#` akan mengembalikan hal berikut saat melewati entri metadata:

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>

Karena kita membutuhkan komentar terbaru untuk keluhan dalam pola 7 (`getLatestCommentByComplaintID`), mari kita gunakan dua parameter kueri tambahan:

1. `ScanIndexForward` harus disetel ke `False` untuk mendapatkan hasil yang diurutkan dalam urutan menurun
2. `Limit` harus diatur ke 1 untuk mendapatkan komentar terbaru (hanya satu)

Mirip dengan access pattern 6 (`getAllCommentsByComplaintID`), kita melewati entri metadata menggunakan `begins_with comm#` sebagai kondisi kunci semacam. Sekarang, Anda dapat melakukan akses pola 7 pada desain ini menggunakan operasi query

denganPK=Complaint123danSK=begin_with comm#,ScanIndexForward=False, danLimit1. Item yang ditargetkan berikut akan dikembalikan sebagai hasilnya:

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>
	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	
Complaint1321	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Mari tambahkan lebih banyak data dummy ke tabel.

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>

Langkah 4: Alamat pola akses 8 (**getAComplaintbyCustomerIDAndComplaintID**) dan 9 (**getAllComplaintsByCustomerID**)

Pola akses 8 (**getAComplaintbyCustomerIDAndComplaintID**) dan 9 (**getAllComplaintsByCustomerID**) memperkenalkan baru kriteria pencarian: **CustomerID**. Mengambil dari tabel yang ada membutuhkan mahal [Scan](#) untuk membaca semua data dan kemudian memfilter item yang relevan untuk **CustomerID** yang bersangkutan. Kita dapat membuat pencarian

ini lebih efisien dengan membuat [indeks sekunder global \(GSI\)](#) bersama `CustomerID` sebagai kunci partisi. Menjaga dalam pikiran one-to-many hubungan antara pelanggan dan keluhan serta pola akses 9 (`getAllComplaintsByCustomerID`), `ComplaintID` akan menjadi kandidat yang tepat untuk kunci semacam.

Data di GSI akan terlihat seperti ini:

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>

Contoh kueri pada GSI ini untuk pola akses 8 (`getAComplaintbyCustomerIDAndComplaintID`) akan menjadi: `customer_id=custXYZ,sort key=Complaint1321`. Hasilnya adalah:

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>

Untuk mendapatkan semua keluhan bagi pelanggan untuk pola akses 9 (`getAllComplaintsByCustomerID`), kueri pada GSI adalah: `customer_id=custXYZ` sebagai kondisi kunci partisi. Hasilnya adalah:

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>

Langkah 5: Pola akses alamat 10 (`escalateComplaintByComplaintID`)

Akses ini memperkenalkan aspek eskalasi. Untuk meningkatkan keluhan, kita dapat menggunakan `UpdateItem` untuk menambahkan atribut seperti `escalated_to` dan `escalation_time` ke item metadata keluhan yang ada. DynamoDB menyediakan desain skema fleksibel yang berarti satu set atribut non-key dapat seragam atau diskrit di item yang berbeda. Lihat di bawah untuk contoh:

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

Langkah 6: Alamat akses pola 11 (`getAllEscalatedComplaints`) dan 12 (`getEscalatedComplaintsByAgentID`)

Hanya segelintir keluhan yang diharapkan akan meningkat dari seluruh kumpulan data. Oleh karena itu, membuat indeks pada atribut terkait eskalasi akan menghasilkan pencarian yang efisien serta penyimpanan GSI yang hemat biaya. Kita bisa melakukan ini dengan memanfaatkan [indeks jarang](#) teknik. GSI dengan kunci partisi sebagai `escalated_todan` mengurutkan kunci sebagai `escalation_time` akan terlihat seperti ini:

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
		Complaint144	metadata	custXYZ2	Complaint144	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
		Complaint132	metadata	custXYZ	Complaint132	investigating	2023-05-10T15:58:00	P2	<descr_text>

Untuk mendapatkan semua keluhan yang meningkat untuk pola akses 11 (`getAllEscalatedComplaints`), kami cukup memindai GSI ini. Perhatikan bahwa pemindaian ini akan berkinerja dan hemat biaya karena ukuran GSI. Untuk mendapatkan keluhan yang meningkat untuk agen tertentu (pola akses 12 (`getEscalatedComplaintsByAgentID`)), kunci partisi akan `escalated_to=agentID` dan kami tetapkan `ScanIndexForward` kepada `False` untuk memesan dari yang terbaru hingga tertua.

Langkah 7: Pola akses alamat 13 (`getCommentsByAgentID`)

Untuk pola akses terakhir, kita perlu melakukan pencarian dengan dimensi baru: `AgentID`. Kami juga membutuhkan pemesanan berbasis waktu untuk membaca komentar antara dua tanggal sehingga kami membuat GSI dengan `agent_id` sebagai kunci partisi dan `comm_date` sebagai kunci semacam. Data dalam GSI ini akan terlihat seperti berikut:

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
		Complaint144	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Contoh kueri pada GSI ini adalah `partition key agentID=AgentAdansort key=comm_date between (2023-04-30T12:30:00, 2023-05-01T09:00:00)`, yang hasilnya adalah:

Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
	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>	
AgentA	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"]

Semua pola akses dan bagaimana desain skema mengatasinya dirangkum dalam tabel di bawah ini:

Pola akses	Tabel dasar/ GSI/LSI	Operasi	Nilai kunci partisi	Urutkan nilai kunci	Kondisi lainnya/filter
CreateComplaint	Tabel dasar	PutItem	pk=Complaint_id	sk=Metadata	
UpdateComplaint	Tabel dasar	UpdateItem	pk=Complaint_id	sk=Metadata	
updateSeveritybyComplaintID	Tabel dasar	UpdateItem	pk=Complaint_id	sk=Metadata	
getComplaintByComplaintID	Tabel dasar	GetItem	pk=Complaint_id	sk=Metadata	
addCommentByComplaintID	Tabel dasar	TransactWriteItems	pk=Complaint_id	sk=Metadata, sk=Comm#comm_date #comm_id	

Pola akses	Tabel dasar/ GSI/LSI	Operasi	Nilai kunci partisi	Urutkan nilai kunci	Kondisi lainnya/filter
getAllCommentsByComplaintID	Tabel dasar	Kueri	pk=Complaint_id	SK dimulai_dengan "comm#"	
getLatestCommentByComplaintID	Tabel dasar	Kueri	pk=Complaint_id	SK dimulai_dengan "comm#"	scan_index_forward=Salah, Batas 1
GetComplaintbyCustomerIDandComplaintID	Customer_complaint_GSI	Kueri	customer_id=customer_id	complaint_id=complaint_id	
getAllComplaintsByCustomerID	Customer_complaint_GSI	Kueri	customer_id=customer_id	T/A	
escalateComplaintByComplaintID	Tabel dasar	UpdateItem	pk=Complaint_id	sk=Metadata	
getAllEscalatedKeluhan	Eskalasi_GSI	Pemindaian	T/A	T/A	
getEscalatedComplaintsByAgentID (pesanan dari yang terbaru hingga yang tertua)	Eskalasi_GSI	Kueri	escalated_to=agent_id	T/A	scan_index_forward=Salah

Pola akses	Tabel dasar/ GSI/LSI	Operasi	Nilai kunci partisi	Urutkan nilai kunci	Kondisi lainnya/filter
getCommentsByAgentID (antara dua tanggal)	Agents_comments_GSI	Kueri	agent_id= agent_id	SK antara (tanggal1, tanggal2)	

Skema akhir sistem manajemen keluhan

Berikut adalah desain skema akhir. Untuk mengunduh desain skema ini sebagai file JSON, lihat [Contoh DynamoDB](#) di atas GitHub.

Tabel dasar

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

Eskalasi_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"]

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke [Meja Kerja NoSQL](#), alat visual yang menyediakan pemodelan data, visualisasi data, dan fitur pengembangan kueri untuk DynamoDB, untuk lebih mengeksplorasi dan mengedit proyek baru Anda. Ikuti langkah-langkah berikut untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called "Unduh"](#).

2. Unduh file skema JSON yang tercantum di atas, yang sudah ada dalam format model NoSQL Workbench.
3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called “Mengimpor model yang ada”](#).
4. Setelah Anda telah diimpor ke NoSQL Workbench, Anda dapat mengedit model data. Untuk informasi selengkapnya, lihat [the section called “Mengedit model yang ada”](#).
5. Untuk memvisualisasikan model data Anda, tambahkan data sampel, atau impor data sampel dari file CSV, gunakan [Visualizer Data](#) fitur NoSQL Workbench.

Desain skema pembayaran berulang di DynamoDB

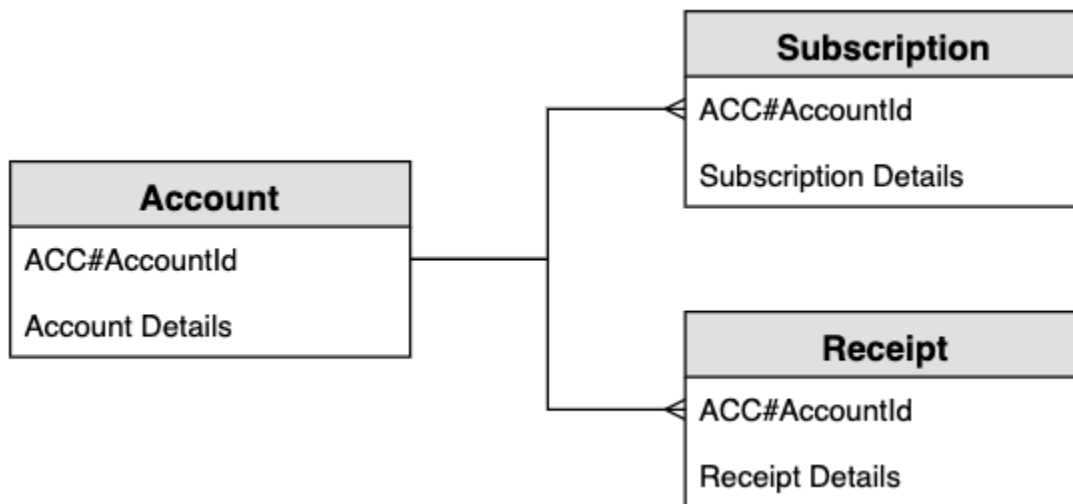
Kasus penggunaan bisnis pembayaran berulang

Kasus penggunaan ini berbicara tentang penggunaan DynamoDB untuk menerapkan sistem pembayaran berulang. Model data memiliki entitas berikut: rekening, langganan, dan tanda terima. Spesifikasi untuk kasus penggunaan kami meliputi:

- Masing-masing akun dapat memiliki beberapa langganan
- Yang berlangganan memiliki `NextPaymentDate` ketika pembayaran berikutnya perlu diproses dan `NextReminderDate` ketika pengingat email dikirim ke pelanggan
- Ada item untuk berlangganan yang disimpan dan diperbarui saat pembayaran diproses (ukuran item rata-rata sekitar 1KB dan throughput tergantung pada jumlah rekening dan langganan)
- Yang pembayaran prosesor juga akan membuat tanda terima sebagai bagian dari proses yang disimpan dalam tabel dan diatur untuk berakhir setelah jangka waktu dengan menggunakan [TTL](#) tambahan

Diagram hubungan entitas pembayaran berulang

Ini adalah diagram hubungan entitas (ERD) yang akan kita gunakan untuk desain skema sistem pembayaran berulang.



Pola akses sistem pembayaran berulang

Ini adalah pola akses yang akan kita pertimbangkan untuk desain skema sistem pembayaran berulang.

1. createSubscription
2. createReceipt
3. updateSubscription
4. getDueRemindersByDate
5. getDuePaymentsByDate
6. getSubscriptionsByAccount
7. getReceiptsByAccount

Desain skema pembayaran berulang

Nama-nama generik PK dan SK digunakan untuk atribut kunci untuk memungkinkan menyimpan berbagai jenis entitas dalam tabel yang sama seperti akun, langganan, dan entitas penerimaan. Pengguna pertama kali membuat langganan, di mana pengguna setuju untuk membayar jumlah pada hari yang sama setiap bulan sebagai imbalan atas suatu produk. Mereka mendapatkan pilihan pada hari mana dalam sebulan untuk memproses pembayaran. Ada juga pengingat yang akan dikirim sebelum pembayaran diproses. Aplikasi ini bekerja dengan memiliki dua pekerjaan batch yang berjalan setiap hari: satu pekerjaan batch mengirimkan pengingat karena hari itu dan pekerjaan batch lainnya memproses pembayaran apa pun yang jatuh tempo hari itu.

Langkah 1: Pola akses alamat 1 (`createSubscription`)

Pola akses 1 (`createSubscription`) digunakan untuk awalnya membuat langganan, dan rincian termasuk `SKU`, `NextPaymentDate`, `NextReminderDate` dan `PaymentDetails` ditetapkan. Langkah ini menunjukkan keadaan tabel hanya untuk satu akun dengan satu langganan. Ada beberapa langganan dalam koleksi item jadi ini adalah one-to-many hubungan.

Primary key		Attributes									
Partition key: PK	Sort key: SK										
ACC#123	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		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 75T"}}	2023-05-18T09:41:25.856Z

Langkah 2: Alamat pola akses 2 (`createReceipt`) dan 3 (`updateSubscription`)

Pola akses 2 (`createReceipt`) digunakan untuk membuat item tanda terima. Setelah pembayaran diproses setiap bulan, pemroses pembayaran akan menulis tanda terima kembali ke tabel dasar. Ada, bisa beberapa tanda terima dalam koleksi item jadi ini adalah one-to-many hubungan. Prosesor pembayaran juga akan memperbarui item berlangganan (akses Pola 3 (`updateSubscription`)) untuk memperbarui `NextReminderDate` atau `NextPaymentDate` untuk bulan berikutnya.

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	2023-05-28T14:15:39.24Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.24Z	2023-06-28	2023-05-21T14:15:39.24Z	2023-06-21	999	{"default-card": {"S": "1234123412341234"}, "default-address": {"S": "12 Bridge Street, Birmingham, B12 75T"}}	2023-05-18T09:41:25.856Z

Langkah 3: Pola akses alamat 4 (`getDueRemindersByDate`)

Aplikasi memproses pengingat untuk pembayaran dalam batch untuk hari ini. Oleh karena itu aplikasi perlu mengakses langganan pada dimensi yang berbeda: tanggal daripada akun. Ini adalah kasus penggunaan yang baik untuk [indeks sekunder global \(GSI\)](#). Pada langkah ini kita menambahkan indeks `GSI-1`, yang menggunakan `NextReminderDate` sebagai kunci partisi `GSI`. Kita tidak perlu mereplikasi semua item. `GSI` ini adalah [indeks jarang](#) dan item tanda terima tidak direplikasi. Kita juga tidak perlu memproyeksikan semua atribut—kita hanya perlu menyertakan subset atribut. Gambar di bawah ini menunjukkan skema `GSI-1` dan memberikan informasi yang diperlukan untuk aplikasi untuk mengirim email pengingat.

Primary key		Attributes				
Partition key: NextReminderDate	Sort key: LastReminderDate	SK	PK	SKU	Email	NextPaymentDate
2023-06-21	2023-05-21T14:15:39.24Z	SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

Langkah 4: Pola akses alamat 5 (**getDuePaymentsByDate**)

Aplikasi memproses pembayaran dalam batch untuk hari ini dengan cara yang sama seperti halnya dengan pengingat. Kami menambahkan GSI - 2 pada langkah ini, dan menggunakan `NextPaymentDate` sebagai kunci partisi GSI. Kita tidak perlu mereplikasi semua item. GSI ini adalah indeks jarang karena item penerimaan tidak direplikasi. Gambar di bawah ini menunjukkan skema 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"}}	

Langkah 5: Alamat pola akses 6 (**getSubscriptionsByAccount**) dan 7 (**getReceiptsByAccount**)

Aplikasi ini dapat mengambil semua langganan untuk akun dengan menggunakan [pertanyaan](#) pada tabel dasar yang menargetkan pengenal akun (PK) dan menggunakan operator jangkauan untuk mendapatkan semua item di mana SK dimulai dengan "SUB#". Aplikasi ini juga dapat menggunakan struktur query yang sama untuk mengambil semua tanda terima dengan menggunakan operator jangkauan untuk mendapatkan semua item di mana SK dimulai dengan "REC #". Hal ini memungkinkan kita untuk memenuhi pola akses 6 (`getSubscriptionsByAccount`) dan 7 (`getReceiptsByAccount`). Aplikasi ini menggunakan pola akses ini sehingga pengguna dapat melihat langganan mereka saat ini dan tanda terima masa lalu mereka selama enam bulan terakhir. Tidak ada perubahan pada skema tabel pada langkah ini dan kita dapat melihat di bawah ini bagaimana kita menargetkan hanya item berlangganan dalam pola akses 6 (`getSubscriptionsByAccount`).

Primary key		Attributes									
Partition key: PK	Sort key: SK	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
	REC#12023-05-28T14:15:39.24#SKU#999	s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
ACC#123	SUB#123#SKU#999	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

Semua pola akses dan bagaimana skema desain alamat mereka diringkas dalam tabel di bawah ini:

Pola akses	Tabel dasar/GSI /LSI	Operasi	Nilai kunci partisi	Urutkan nilai kunci
MembuatSubscription	Tabel dasar	PutItem	ACC #account_id	<SUBID>SUB# #SKU <SKUID>
Pembuatan RECEIPT	Tabel dasar	PutItem	ACC #account_id	REC#<ReceiptDate> #SKU <SKUID>
UpdateSubscription	Tabel dasar	UpdateItem	ACC #account_id	<SUBID>SUB# #SKU <SKUID>
getDueRemindersByDate	GSI-1	Kueri	<NextReminderDate>	
getDuePaymentsByDate	GSI-2	Kueri	<NextPaymentDate>	
getSubscriptionsByAkun	Tabel dasar	Kueri	ACC #account_id	SK dimulai dengan "SUB #"
getReceiptsByAkun	Tabel dasar	Kueri	ACC #account_id	SK dimulai dengan "REC #"

Pembayaran berulang skema akhir

Berikut adalah desain skema akhir. Untuk mengunduh desain skema ini sebagai file JSON, lihat [Contoh DynamoDB](#) di atas GitHub.

Tabel dasar

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	2023-05-28T14:15:39.24Z	12.99	1700318200						
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
	s@s.com	28	12.99	2023-05-18T14:15:39.24Z	2023-06-28	2023-05-21T14:15:39.24Z	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	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

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"}}

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke [Meja Kerja NoSQL](#), alat visual yang menyediakan pemodelan data, visualisasi data, dan fitur pengembangan kueri untuk DynamoDB, untuk lebih mengeksplorasi dan mengedit proyek baru Anda. Ikuti langkah-langkah berikut untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called “Unduh”](#).
2. Unduh file skema JSON yang tercantum di atas, yang sudah ada dalam format model NoSQL Workbench.
3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called “Mengimpor model yang ada”](#).
4. Setelah Anda telah diimpor ke NOSQL Workbench, Anda dapat mengedit model data. Untuk informasi selengkapnya, lihat [the section called “Mengedit model yang ada”](#).
5. Untuk memvisualisasikan model data Anda, tambahkan data sampel, atau impor data sampel dari file CSV, gunakan [Visualizer Data](#) fitur NoSQL Workbench.

Memantau pembaruan status perangkat di DynamoDB

Kasus penggunaan ini berbicara tentang penggunaan DynamoDB untuk memantau pembaruan status perangkat (atau perubahan status perangkat) di DynamoDB.

Kasus penggunaan

Dalam kasus penggunaan IoT (pabrik pintar misalnya) banyak perangkat perlu dipantau oleh operator dan mereka secara berkala mengirim status atau log mereka ke sistem pemantauan. Ketika ada masalah dengan perangkat, status untuk perangkat berubah daribiasakepadamemperingati. Ada tingkat log atau status yang berbeda tergantung pada tingkat keparahan dan jenis perilaku abnormal

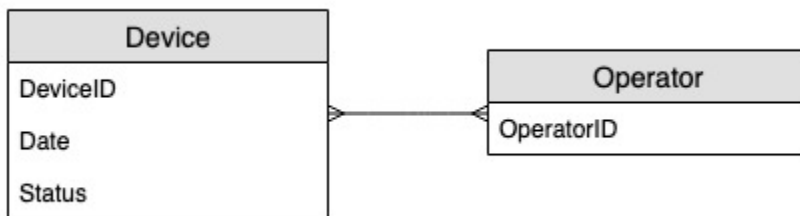
pada perangkat. Sistem kemudian menugaskan operator untuk memeriksa perangkat dan mereka dapat meningkatkan masalah kepada supervisor mereka jika diperlukan.

Beberapa pola akses khas untuk sistem ini meliputi:

- Buat entri log untuk perangkat
- Dapatkan semua log untuk status perangkat tertentu yang menampilkan log terbaru terlebih dahulu
- Dapatkan semua log untuk operator tertentu di antara dua tanggal
- Dapatkan semua log yang dieskalasi untuk supervisor tertentu
- Dapatkan semua log yang dieskalasi dengan status perangkat tertentu untuk supervisor tertentu
- Dapatkan semua log eskalasi dengan status perangkat tertentu untuk supervisor tertentu untuk tanggal tertentu

Diagram hubungan entitas

Ini adalah diagram hubungan entitas (ERD) yang akan kami gunakan untuk memantau pembaruan status perangkat.



Pola akses

Ini adalah pola akses yang akan kami pertimbangkan untuk memantau pembaruan status perangkat.

1. `createLogEntryForSpecificDevice`
2. `getLogsForSpecificDevice`
3. `getWarningLogsForSpecificDevice`
4. `getLogsForOperatorBetweenTwoDates`
5. `getEscalatedLogsForSupervisor`
6. `getEscalatedLogsWithSpecificStatusForSupervisor`
7. `getEscalatedLogsWithSpecificStatusForSupervisorForDate`

Evolusi desain skema

Langkah 1: Pola akses alamat 1 (**createLogEntryForSpecificDevice**) dan 2 (**getLogsForSpecificDevice**)

Unit penskalaan untuk sistem pelacakan perangkat adalah perangkat individual. Dalam sistem ini, `deviceId` mengidentifikasi perangkat secara unik. Hal ini membuat `deviceId` kandidat yang baik untuk kunci partisi. Setiap perangkat mengirimkan informasi ke sistem pelacakan secara berkala (katakanlah, setiap lima menit atau lebih). Urutan ini menjadikan tanggal sebagai kriteria penyortiran logis dan oleh karena itu, kunci pengurutan. Data sampel dalam kasus ini akan terlihat seperti ini:

Primary key		Attributes
Partition key: DeviceID	Sort key: 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

Untuk mengambil entri log untuk perangkat tertentu, kita dapat melakukan [pertanyaan](#) operasi dengan kunci `partisiDeviceID="d#12345"`.

Langkah 2: Pola akses alamat 3 (`getWarningLogsForSpecificDevice`)

`State` adalah atribut non-kunci, menangani pola akses 3 dengan skema saat ini akan membutuhkan [ekspresi filter](#). Di DynamoDB, ekspresi filter diterapkan setelah data dibaca menggunakan ekspresi kondisi kunci. Misalnya, jika kita mengambil log peringatan untuk `d#12345`, operasi kueri dengan kunci `partisiDeviceID="d#12345"` akan membaca empat item dari tabel di atas dan kemudian menyaring satu item dengan `memperingati status`. Pendekatan ini tidak efisien dalam skala. Ekspresi filter dapat menjadi cara yang baik untuk mengecualikan item yang ditanyakan jika rasio item yang dikecualikan rendah atau kueri jarang dilakukan. Namun, untuk kasus di mana banyak item diambil dari tabel dan sebagian besar item disaring, kami dapat terus mengembangkan desain tabel kami sehingga berjalan lebih efisien.

Mari kita ubah cara menangani pola akses ini dengan menggunakan [kunci sortir komposit](#). Anda dapat mengimpor data sampel dari [DeviceStateLog_3.json](#) di mana kunci sortir diubah menjadi `State#Date`. Kunci sortir ini adalah komposisi atribut `State`, `#`, dan `Date`. Dalam contoh ini, `#` digunakan sebagai pembatas. Data sekarang terlihat seperti ini:

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

Untuk mengambil hanya log peringatan untuk perangkat, kueri menjadi lebih ditargetkan dengan skema ini. Kondisi kunci untuk kueri menggunakan kunci `partisiDeviceID="d#12345"` dan

mengurutkan kunciState#Date begins_with "WARNING". Kueri ini hanya akan membaca tiga item yang relevan denganmemperingatinegara.

Langkah 3: Pola akses alamat 4 (**getLogsForOperatorBetweenTwoDates**)

Anda dapat mengimpor[DeviceStateLog_4.json](#) dimanaOperatoratribut ditambahkan keDeviceStateLogtabel dengan contoh data.

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	

Sejak Operator saat ini bukan kunci partisi, tidak ada cara untuk melakukan pencarian nilai kunci langsung pada tabel ini berdasarkan Operator ID. Kita harus membuat yang baru [koleksi](#)

[barang](#) Dengan indeks sekunder global pada `operatorID`. Pola akses memerlukan pencarian berdasarkan tanggal sehingga tanggal adalah atribut kunci sortir untuk [indeks sekunder global \(GSI\)](#). Seperti inilah tampilan GSI sekarang:

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	

Untuk pola akses 4 (`getLogsForOperatorBetweenTwoDates`), Anda dapat menanyakan GSI ini dengan kunci partisi `operatorID=Liz` dan mengurutkan kunci Date antara `2020-04-11T05:58:00` dan `2020-04-24T14:50:00`.

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	

Langkah 4: Alamat pola akses 5 (**getEscalatedLogsForSupervisor**)
6 (**getEscalatedLogsWithSpecificStatusForSupervisor**), dan 7
(**getEscalatedLogsWithSpecificStatusForSupervisorForDate**)

Kami akan menggunakan [indeks jarang](#) untuk mengatasi pola akses ini.

Indeks sekunder global jarang secara default, jadi hanya item dalam tabel dasar yang berisi atribut kunci utama indeks yang benar-benar akan muncul di indeks. Ini adalah cara lain untuk mengecualikan item yang tidak relevan untuk pola akses yang dimodelkan.

Anda dapat mengimpor [DeviceStateLog_6.json](#) dimana `EscalatedTo` atribut ditambahkan ke `DeviceStateLog` tabel dengan contoh data. Seperti disebutkan sebelumnya, tidak semua log akan ditingkatkan menjadi supervisor.

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	

Anda sekarang dapat membuat yang baru dimana `EscalatedTo` adalah kunci partisi dan `State#Date` adalah kunci sortir. Perhatikan bahwa hanya item yang memiliki keduanya `EscalatedTo` dan `State#Date` atribut muncul di indeks.

Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date				
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator	Date	State
		d#11223	Sue	2020-04-27T16:15:00	WARNING4

Sisa pola akses dirangkum sebagai berikut:

Semua pola akses dan bagaimana desain skema mengatasinya dirangkum dalam tabel di bawah ini:

Pola akses	Meja dasar/ GSI/LSI	Operasi	Nilai kunci partisi	Urutkan nilai kunci	Kondisi/filter lainnya
createLogEntryForSpecificDevice	Tabel dasar	PutItem	deviceID= DeviceID	Negara #Date =negara #date	
getLogsForSpecificDevice	Tabel dasar	Kueri	deviceID= DeviceID	Status #Date begins_with "state1#"	ScanIndex Forward = Salah
getWarningLogsForSpecificDevice	Tabel dasar	Kueri	deviceID= DeviceID	Status #Date begins_with "PERINGATAN"	
getLogsForOperatorBetweenTwoDates	GSI-1	Kueri	operator= OperatorName	Tanggal antara date1 dan date2	
getEscalatedLogsForSupervisor	GSI-2	Kueri	EscalatedTo=Nama pengawas		

Pola akses	Meja dasar/ GSI/LSI	Operasi	Nilai kunci partisi	Urutkan nilai kunci	Kondisi/filter lainnya
getEscalatedLogsWithSpecificStatusForSupervisor	GSI-2	Kueri	Escalated To>Nama pengawas	Status #Date begins_with "state1#"	
getEscalatedLogsWithSpecificStatusForSupervisorForDate	GSI-2	Kueri	Escalated To>Nama pengawas	Negara #Date begins_wi th "state1 #date1"	

Skema akhir

Berikut adalah desain skema akhir. Untuk mengunduh desain skema ini sebagai file JSON, lihat [DynamoDB Contoh](#) di atas GitHub.

Tabel dasar

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

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke [NoSQL Workbench](#) Alat visual yang menyediakan pemodelan data, visualisasi data, dan fitur pengembangan kueri untuk DynamoDB, untuk mengeksplorasi dan mengedit proyek baru Anda. Ikuti langkah-langkah ini untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called “Unduh”](#).
2. Unduh file skema JSON yang tercantum di atas, yang sudah dalam format model NoSQL Workbench.
3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called “Mengimpor model yang ada”](#).
4. Setelah Anda mengimpor ke NoSQL Workbench, Anda dapat mengedit model data. Untuk informasi selengkapnya, lihat [the section called “Mengedit model yang ada”](#).
5. Untuk memvisualisasikan model data Anda, menambahkan data sampel, atau mengimpor data sampel dari file CSV, gunakan [Visualisasi Data](#) fitur NoSQL Workbench.

Menggunakan DynamoDB sebagai penyimpanan data untuk toko online

Kasus penggunaan ini membahas penggunaan DynamoDB sebagai penyimpanan data untuk toko online (atau e-store).

Kasus penggunaan

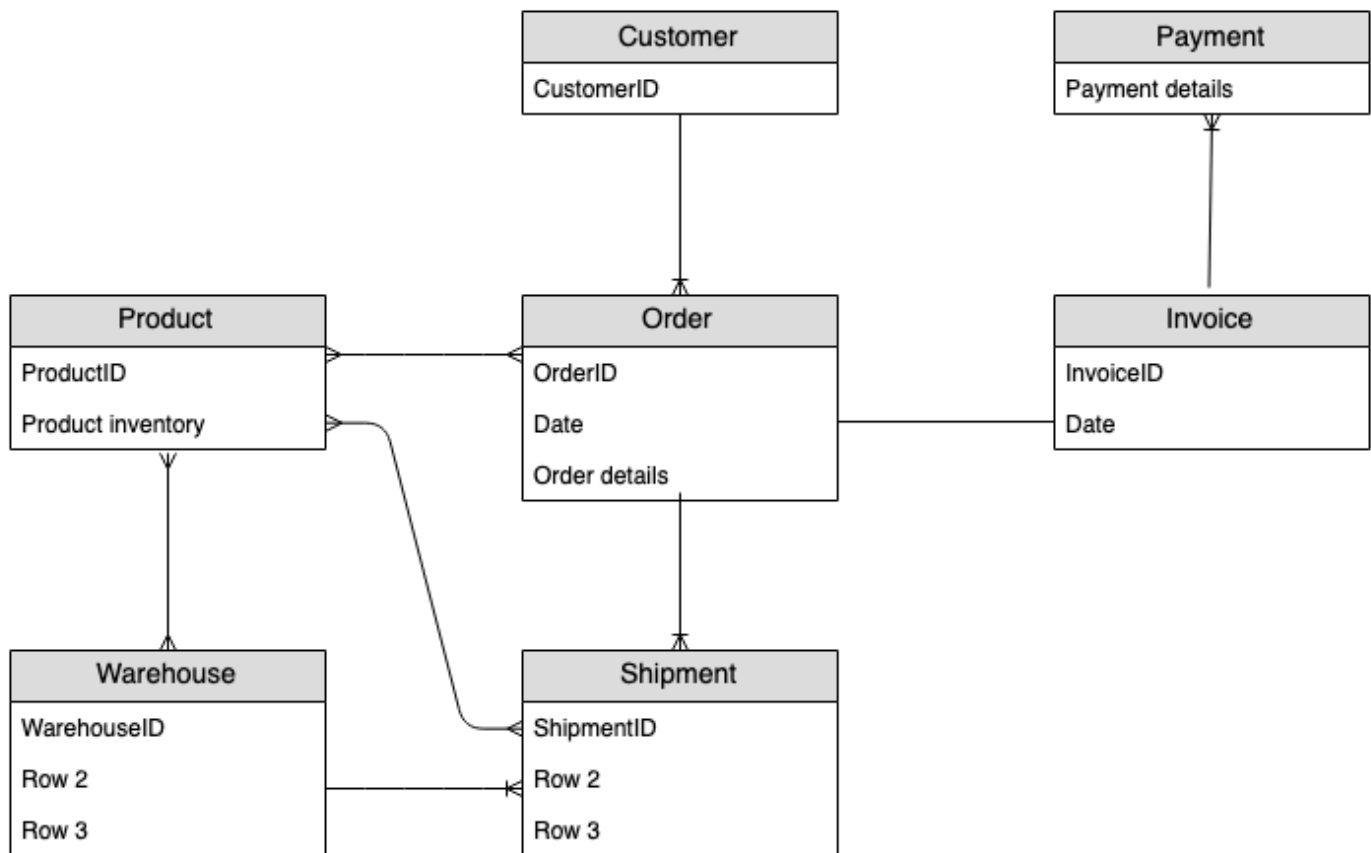
Toko online memungkinkan pengguna menelusuri berbagai produk dan pada akhirnya membelinya. Berdasarkan faktur yang dihasilkan, pelanggan dapat membayar menggunakan kode diskon atau kartu hadiah lalu membayar jumlah yang tersisa dengan kartu kredit. Produk yang dibeli akan diambil dari salah satu gudang dan akan dikirim ke alamat yang diberikan. Pola akses umum untuk toko online meliputi:

- Mendapatkan pelanggan untuk customerId tertentu

- Mendapatkan produk untuk productId tertentu
- Mendapatkan gudang untuk warehouseId tertentu
- Mendapatkan inventaris produk untuk semua gudang berdasarkan productId
- Mendapatkan pesanan untuk orderId tertentu
- Mendapatkan semua produk untuk orderId tertentu
- Mendapatkan faktur untuk orderId tertentu
- Mendapatkan semua pengiriman untuk orderId tertentu
- Mendapatkan semua pesanan untuk productId tertentu dan rentang tanggal tertentu
- Mendapatkan faktur untuk invoiceId tertentu
- Mendapatkan semua pembayaran untuk invoiceId tertentu
- Mendapatkan detail pengiriman untuk shipmentId tertentu
- Mendapatkan semua pengiriman untuk warehouseId tertentu
- Mendapatkan inventaris semua produk untuk warehouseId tertentu
- Mendapatkan semua faktur untuk customerId tertentu dan rentang tanggal tertentu
- Mendapatkan semua produk yang dipesan untuk customerId tertentu dan rentang tanggal tertentu

Diagram hubungan entitas

Ini adalah diagram hubungan entitas (ERD) yang akan kita gunakan untuk memodelkan DynamoDB sebagai penyimpanan data untuk toko online.



Pola akses

Ini adalah pola akses yang akan dipertimbangkan saat menggunakan DynamoDB sebagai penyimpanan data untuk toko online.

1. `getCustomerByCustomerId`
2. `getProductByProductId`
3. `getWarehouseByWarehouseId`
4. `getProductInventoryByProductId`
5. `getOrderDetailsByOrderId`
6. `getProductByOrderId`
7. `getInvoiceByOrderId`
8. `getShipmentByOrderId`
9. `getOrderByProductIdForDateRange`
10. `getInvoiceByInvoiceId`

- 11.getPaymentByInvoiceId
- 12.getShipmentDetailsByShipmentId
- 13.getShipmentByWarehouseId
- 14.getProductInventoryByWarehouseId
- 15.getInvoiceByCustomerIdForDateRange
- 16.getProductsByCustomerIdForDateRange

Evolusi desain skema

Menggunakan [NoSQL Workbench untuk DynamoDB](#), import [AnOnlineShop_1.json](#) untuk membuat model data baru dipanggil AnOnlineShop dan tabel baru dipanggil. OnLineShop Perhatikan bahwa kita menggunakan nama generik PK dan SK untuk kunci partisi dan kunci urutan. Hal ini adalah praktik yang digunakan untuk menyimpan berbagai jenis entitas dalam tabel yang sama.

Langkah 1: Tangani pola akses 1 (**getCustomerByCustomerId**)

Impor [AnOnlineShop_2.json](#) untuk menangani pola akses 1 (). getCustomerByCustomerId Beberapa entitas tidak memiliki hubungan dengan entitas lain, jadi kita akan menggunakan nilai yang sama dari PK dan SK untuk entitas tersebut. Dalam contoh data, perhatikan bahwa kunci menggunakan awalan c# untuk membedakan customerId dari entitas lain yang akan ditambahkan nanti. Praktik ini diulang untuk entitas lain juga.

Untuk menangani pola akses ini, operasi [GetItem](#) dapat digunakan dengan PK=customerId dan SK=customerId.

Langkah 2: Tangani pola akses 2 (**getProductByProductId**)

Impor [AnOnlineShop_3.json](#) ke alamat pola akses 2 (getProductByProductId) untuk entitas. product Entitas produk diawali oleh p# dan atribut kunci urutan yang sama digunakan untuk menyimpan customerId serta productId. Penamaan generik dan [partisi vertikal](#) memungkinkan kita membuat koleksi item untuk desain tabel tunggal yang efektif.

Untuk menangani pola akses ini, operasi GetItem dapat digunakan dengan PK=productId dan SK=productId.

Langkah 3: Tangani pola akses 3 (**getWarehouseByWarehouseId**)

Impor [AnOnlineShop_4.json](#) ke alamat pola akses 3 (getWarehouseByWarehouseId) untuk entitas. warehouse Saat ini entitas customer, product, dan warehouse ditambahkan ke tabel

yang sama. Entitas tersebut dibedakan menggunakan awalan dan atribut `EntityType`. Atribut jenis (atau penamaan awalan) meningkatkan keterbacaan model. Keterbacaan akan terpengaruh jika kita hanya menyimpan ID alfanumerik untuk entitas yang berbeda dalam atribut yang sama. Akan sulit untuk membedakan satu entitas dari yang lain tanpa adanya pengidentifikasi ini.

Untuk menangani pola akses ini, operasi `GetItem` dapat digunakan dengan `PK=warehouseId` dan `SK=warehouseId`.

Tabel dasar:

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"}}	

Langkah 4: Tangani pola akses 4 (`getProductInventoryByProductId`)

Impor [AnOnlineShop_5.json](#) ke pola akses alamat 4 (). `getProductInventoryByProductId` `warehouseItem` entitas digunakan untuk melacak jumlah produk di setiap gudang. Item ini biasanya akan diperbarui ketika produk ditambahkan atau dihapus dari gudang. Seperti yang terlihat di ERD, ada many-to-many hubungan antara `product` dan `warehouse`. Di sini, one-to-many hubungan dari `product` ke `warehouse` dimodelkan sebagai `warehouseItem`. Nantinya, one-to-many hubungan dari `warehouse` hingga `product` akan dimodelkan juga.

Pola akses 4 dapat ditangani dengan kueri pada `PK=ProductId` dan `SK begins_with "w#"`.

Untuk informasi selengkapnya tentang `begins_with()` dan ekspresi lain yang dapat diterapkan ke kunci urutan, lihat [Ekspresi Kondisi Kunci](#).

Tabel dasar:

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"}}	

Langkah 5: Tangani pola akses 5 (**getOrderDetailsByOrderId**) dan 6 (**getProductByOrderId**)

Tambahkan beberapa `logicustomer`, `product`, dan `warehouse` item ke tabel dengan mengimpor [AnOnlineShop_6.json](#). Kemudian, impor [AnOnlineShop_7.json](#) untuk membuat koleksi item `order` yang dapat mengatasi pola akses 5 (`getOrderDetailsByOrderId`) dan 6 (`getProductByOrderId`). Anda dapat melihat one-to-many hubungan antara `order` dan `product` dimodelkan sebagai entitas `OrderItem`.

Untuk menangani pola akses 5 (`getOrderDetailsByOrderId`), lakukan kueri tabel dengan `PK=orderId`. Tindakan ini akan memberikan semua informasi tentang pesanan termasuk `customerId` dan produk yang dipesan.

Tabel dasar:

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

Untuk menangani pola akses 6 (`getProductByOrderId`), kita hanya perlu membaca produk dalam order. Kueri tabel dengan `PK=orderId` dan `SK begins_with "p#"` untuk mencapai hal ini.

Tabel dasar:

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

Langkah 6: Tangani pola akses 7 (`getInvoiceByOrderId`)

Impor [AnOnlineShop_8.json](#) untuk menambahkan invoice entitas ke koleksi item pesanan untuk menangani pola akses 7 (`getInvoiceByOrderId`). Untuk menangani pola akses ini, operasi `PK=orderId` dan `SK begins_with "i#"` dapat digunakan.

Tabel dasar:

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

Langkah 7: Tangani pola akses 8 (**getShipmentByOrderId**)

Impor [AnOnlineShop_9.json](#) untuk menambahkan shipment entitas ke koleksi item pesanan untuk mengatasi pola akses 8 (). `getShipmentByOrderId` Model yang dipartisi secara vertikal yang sama diperluas dengan menambahkan lebih banyak jenis entitas dalam desain tabel tunggal. Perhatikan bagaimana koleksi item pesanan berisi hubungan yang berbeda dari hubungan entitas `order` dengan entitas `shipment`, `orderItem`, dan `invoice`.

Untuk mendapatkan pengiriman berdasarkan `orderId`, Anda dapat melakukan operasi kueri dengan `PK=orderId` dan `SK begins_with "sh#"`.

Tabel dasar:

Primary key		Attributes				
Partition key: PK	Sort key: SK					
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		
o#12345	EntityType	Address		Type	Date	WarehouseId
	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	WarehouseId
	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

Langkah 8: Tangani pola akses 9 (**getOrderByProductIdForDateRange**)

Kita membuat koleksi item pesanan pada langkah sebelumnya. Pola akses ini memiliki dimensi pencarian baru (ProductID dan Date) yang mengharuskan Anda memindai seluruh tabel dan memfilter catatan yang relevan untuk mengambil item yang ditargetkan. Untuk menangani pola akses ini, kita harus membuat [indeks sekunder global \(GSI\)](#). Impor [AnOnlineShop_10.json](#) untuk membuat koleksi item baru menggunakan GSI yang memungkinkan untuk mengambil `orderItem` data dari beberapa koleksi item pesanan. Sekarang data memiliki GSI1-PK dan GSI1-SK yang masing-masing akan menjadi kunci partisi dan kunci urutan GSI1.

DynamoDB otomatis mengisi item yang berisi atribut kunci GSI dari tabel ke GSI. Tidak perlu melakukan penyisipan tambahan secara manual ke GSI.

Untuk menangani pola akses 9, lakukan kueri GSI1 dengan GSI1-PK=productId dan GSI1SK between (date1, date2).

Tabel dasar:

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

Langkah 9: Tangani pola akses 10 (**getInvoiceByInvoiceId**) dan 11 (**getPaymentByInvoiceId**)

Impor [AnOnlineShop_11.json](#) untuk mengatasi pola akses 10 (**getInvoiceByInvoiceId**) dan 11 (**getPaymentByInvoiceId**), keduanya terkait dengan `invoice`. Meskipun berbeda, dua pola akses ini direalisasikan menggunakan kondisi kunci yang sama. `Payments` didefinisikan sebagai atribut dengan jenis data peta pada entitas `invoice`.

Note

GSI1-PK dan GSI1-SK kelebihan beban untuk menyimpan informasi berbagai entitas sehingga beberapa pola akses dapat dilayani dari GSI yang sama. Untuk informasi selengkapnya tentang kelebihan beban GSI, lihat [Muatan berlebih Indeks Sekunder Global](#).

Untuk menangani pola akses 10 dan 11, lakukan kueri GSI1 dengan GSI1-PK=invoiceId dan GSI1-SK=invoiceId.

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

Langkah 10: Tangani pola akses 12 (**getShipmentDetailsByShipmentId**) dan 13 (**getShipmentByWarehouseId**)

Impor [AnOnlineShop_12.json](#) untuk mengatasi pola akses 12 (getShipmentDetailsByShipmentId) dan 13 (). getShipmentByWarehouseId

Perhatikan bahwa entitas shipmentItem ditambahkan ke koleksi item pesanan pada tabel dasar agar dapat mengambil semua detail tentang pesanan dalam satu operasi kueri.

Tabel dasar:

Primary key		Attributes								
Partition key: PK	Sort key: SK									
o#12345	sh#88899	EntityType	GS11-PK	GS11-SK	GS12-PK	GS12-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	GS11-PK	GS11-SK	GS12-PK	GS12-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	GS11-PK	GS11-SK	Quantity					
		shipmentItem	sh#98765	p#99887	3					
	shp#54321	EntityType	GS11-PK	GS11-SK	Quantity					
		shipmentItem	sh#88899	p#99887	2					
	shp#55555	EntityType	GS11-PK	GS11-SK	Quantity					
		shipmentItem	sh#98765	p#12345	2					

Kunci GSI1 partisi dan sortir telah digunakan untuk memodelkan one-to-many hubungan antara shipment dan shipmentItem. Untuk menangani pola akses 12 (getShipmentDetailsByShipmentId), lakukan kueri GSI1 dengan GSI1-PK=shipmentId dan GSI1-SK=shipmentId.

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	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

Kita perlu membuat GSI (GSI2) lain untuk memodelkan one-to-many hubungan baru antara warehouse dan shipment untuk pola akses 13 (getShipmentByWarehouseId). Untuk

menangani pola akses ini, lakukan kueri GSI2 dengan GSI2-PK=warehouseId dan GSI2-SK begins_with "sh#".

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
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

Langkah 11: Tangani pola akses 14 (**getProductInventoryByWarehouseId**) 15 (**getInvoiceByCustomerIdForDateRange**), dan 16 (**getProductsByCustomerIdForDateRange**)

Impor [AnOnlineShop_13.json](#) untuk menambahkan data yang terkait dengan kumpulan pola akses berikutnya. Untuk menangani pola akses 14 (**getProductInventoryByWarehouseId**), lakukan kueri GSI2 dengan GSI2-PK=warehouseId dan GSI2-SK begins_with "p#".

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..." } } } } } } }	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	

Untuk menangani pola akses 15 (getInvoiceByCustomerIdForDateRange), lakukan kueri GSI2 dengan GSI2-PK=customerId dan GSI2-SK between (i#date1, i#date2).

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	

Untuk menangani pola akses 16 (getProductsByCustomerIdForDateRange), lakukan kueri GSI2 dengan GSI2-PK=customerId dan GSI2-SK between (p#date1, p#date2).

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

Di [NoSQL Workbench](#), faset mewakili pola akses data aplikasi yang berbeda untuk DynamoDB. Faset memberi Anda cara untuk melihat subset data dalam tabel tanpa harus melihat rekaman yang tidak memenuhi batasan faset. Faset dianggap sebagai alat pemodelan data visual, dan tidak ada sebagai konstruksi yang dapat digunakan di DynamoDB, karena aspek tersebut murni bantuan untuk memodelkan pola akses. Impor [AnOnlineShop_facets.json](#) untuk melihat aspek untuk kasus penggunaan ini.

Semua pola akses dan bagaimana desain skema menanganinya dirangkum dalam tabel di bawah ini:

Pola akses	Tabel dasar/GSI /LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan
getCustomerByCustomerId	Tabel dasar	GetItem	PK=customerId	SK=customerId
getProductById	Tabel dasar	GetItem	PK=productId	SK=productId
getWarehouseByWarehouseId	Tabel dasar	GetItem	PK=warehouseId	SK=warehouseId
getProductInventoryByProductId	Tabel dasar	Kueri	PK=productId	SK begins_with "w#"
getOrderDetailsByOrderId	Tabel dasar	Kueri	PK=orderId	
getProductByOrderId	Tabel dasar	Kueri	PK=orderId	SK begins_with "p#"
getInvoiceByOrderId	Tabel dasar	Kueri	PK=orderId	SK begins_with "i#"
getShipmentByOrderId	Tabel dasar	Kueri	PK=orderId	SK begins_with "sh#"
getOrderByIdForDateRange	GSI1	Kueri	PK=productId	SK antara date1 dan date2
getInvoiceById	GSI1	Kueri	PK=invoiceId	SK=invoiceId

Pola akses	Tabel dasar/GSI /LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan
getPaymentByInvoiceId	GSI1	Kueri	PK=invoiceId	SK=invoiceId
getShipmentDetailsByShipmentId	GSI1	Kueri	PK=shipmentId	SK=shipmentId
getShipmentByWarehouseId	GSI2	Kueri	PK=warehouseId	SK begins_with "sh#"
getProductInventoryByWarehouseId	GSI2	Kueri	PK=warehouseId	SK begins_with "p#"
getInvoiceByCustomerIdForDateRange	GSI2	Kueri	PK=customerId	SK antara i#date1 dan i#date2
getProductsByCustomerIdForDateRange	GSI2	Kueri	PK=customerId	SK antara p#date1 dan p#date2

Skema akhir toko online

Berikut adalah desain skema akhir. [Untuk mengunduh desain skema ini sebagai file JSON, lihat DynamoDB Design Patterns on GitHub](#)

Tabel dasar

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	GS12-PK	GS12-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	GS12-PK	GS12-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"}}		

Toko online

		EntityType	Address		
			{"Country": {"S": "Sweden"}, "County": {"S": "Vastra Gotaland"}, "City": {"S": "Goteborg"}, "Street": {"S": "MainStreet"}, "Number": {"S": "20"}, "ZipCode": {"S": "41111"}}		

GS1

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":"Master Card"}, "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	sh#88899	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
			o#12345	sh#88899	shipment	w#12376	sh#88899	{"Country": {"S":"Sweden"}, "Country": {"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	sh#98765	PK	SK	EntityType	Quantity				
			o#12345	shp#12345	shipmentItem	3				
sh#98765	sh#98765	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date	
		o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country": {"S":"Sweden"}, "Country": {"S":"Vastra Gotaland"}, "City": {"S":"Goteborg"}, "Street": {"S":"Slanbarsvagen"}, "Number": {"S":"34"}, "ZipCode": {"S":"41787"}}	Express	2020-06-22T10:20:00	
Toko online	sh#98765	o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country": {"S":"Sweden"}, "Country": {"S":"Vastra Gotaland"}, "City": {"S":"Goteborg"}, "Street": {"S":"Slanbarsvagen"}, "Number": {"S":"34"}, "ZipCode": {"S":"41787"}}	Express	2020-06-22T10:20:00	

GS12

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": "Slanbar svagen"}, "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": "Slanbar svagen"}, "Number": {"S": "34"}, "ZipCode": {"S": "41787"} }	Express	2020-06-22T08:20:00
Toko online								Versi API 2012-08-10 1480	

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke [NoSQL Workbench](#), sebuah alat visual yang menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri untuk DynamoDB, guna mengeksplorasi dan mengedit proyek baru Anda lebih lanjut. Ikuti langkah-langkah berikut untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called “Unduh”](#).
2. Unduh file skema JSON yang tercantum di atas, yang sudah dalam format model NoSQL Workbench.
3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat [the section called “Mengimpor model yang ada”](#).
4. Setelah mengimpor model data ke NoSQL Workbench, Anda dapat mengeditnya. Untuk informasi selengkapnya, lihat [the section called “Mengedit model yang ada”](#).
5. Untuk memvisualisasikan model data Anda, menambahkan data sampel, atau mengimpor data sampel dari file CSV, gunakan fitur [Data Visualizer](#) di NoSQL Workbench.

Migrasi ke DynamoDB dari basis data relasional

Migrasi basis data relasional ke DynamoDB memerlukan perencanaan yang matang untuk memastikan hasil yang sukses. Panduan ini akan membantu Anda memahami cara kerja proses ini, alat yang Anda miliki, lalu cara mengevaluasi strategi migrasi potensial dan memilih salah satu yang sesuai dengan kebutuhan Anda.

Topik

- [Alasan untuk bermigrasi ke DynamoDB](#)
- [Pertimbangan saat memigrasikan basis data relasional ke DynamoDB](#)
- [Memahami cara kerja migrasi ke DynamoDB](#)
- [Alat untuk membantu migrasi ke DynamoDB](#)
- [Memilih strategi yang tepat untuk migrasi ke DynamoDB](#)
- [Melakukan migrasi offline ke DynamoDB](#)
- [Melakukan migrasi hibrid ke DynamoDB](#)
- [Melakukan migrasi online ke DynamoDB dengan memigrasikan setiap tabel 1:1](#)
- [Lakukan migrasi online ke DynamoDB menggunakan tabel penahapan khusus](#)

Alasan untuk bermigrasi ke DynamoDB

Migrasi ke Amazon DynamoDB menghadirkan berbagai manfaat menarik bagi bisnis dan organisasi. Berikut adalah beberapa keuntungan utama yang menjadikan DynamoDB sebagai pilihan menarik untuk migrasi basis data:

- **Skalabilitas:** DynamoDB dirancang untuk menangani beban kerja yang besar dan menskalakan dengan mulus untuk mengakomodasi lalu lintas dan volume data yang terus bertambah. Dengan DynamoDB, Anda dapat dengan mudah meningkatkan atau menurunkan basis data berdasarkan permintaan sehingga memastikan aplikasi Anda dapat menangani lonjakan lalu lintas secara tiba-tiba tanpa mengorbankan performa.
- **Performa:** DynamoDB menawarkan akses data latensi rendah sehingga memungkinkan aplikasi untuk mengambil dan memproses data dengan kecepatan luar biasa. Arsitektur terdistribusinya memastikan operasi baca dan tulis didistribusikan di beberapa simpul sehingga memberikan waktu respons milidetik satu digit yang konsisten bahkan pada tingkat permintaan tinggi.

- **Terkelola penuh:** DynamoDB adalah layanan terkelola penuh yang disediakan oleh AWS. Ini berarti AWS menangani aspek operasional manajemen basis data, termasuk penyediaan, konfigurasi, penambalan, pencadangan, dan penskalaan. Ini memungkinkan Anda untuk lebih fokus pada pengembangan aplikasi dan tidak terlalu fokus pada tugas administrasi basis data.
- **Arsitektur nirserver:** DynamoDB mendukung model nirserver, yang dikenal sebagai [DynamoDB sesuai permintaan](#), yaitu Anda hanya membayar untuk permintaan baca dan tulis aktual yang dibuat aplikasi Anda tanpa memerlukan penyediaan kapasitas di muka. *pay-per-request* Model ini menawarkan efisiensi biaya dan overhead operasional minimal, karena Anda hanya membayar sumber daya yang Anda konsumsi tanpa perlu menyediakan dan memantau kapasitas.
- **Fleksibilitas NoSQL:** Tidak seperti basis data relasional tradisional, DynamoDB mengikuti model data NoSQL, sehingga memberikan fleksibilitas dalam desain skema. Dengan DynamoDB, Anda dapat menyimpan data terstruktur, semi-terstruktur, dan tidak terstruktur, sehingga cocok untuk menangani jenis data yang beragam dan berkembang. Fleksibilitas ini memungkinkan siklus pengembangan yang lebih cepat dan adaptasi yang lebih mudah terhadap perubahan kebutuhan bisnis.
- **Ketersediaan dan daya tahan tinggi:** DynamoDB mereplikasi data di beberapa zona ketersediaan dalam suatu Wilayah, sehingga memastikan ketersediaan tinggi dan daya tahan data. Replikasi, failover, dan pemulihan direplikasi secara otomatis, sehingga meminimalkan risiko kehilangan data atau gangguan layanan. DynamoDB menyediakan ketersediaan SLA hingga 99,999%.
- **Keamanan dan kepatuhan:** DynamoDB terintegrasi dengan AWS Identity and Access Management untuk kontrol akses terperinci. Ini menyediakan enkripsi diam dan bergerak, sehingga memastikan keamanan data Anda. DynamoDB juga mematuhi berbagai standar kepatuhan, termasuk HIPAA, PCI DSS, dan GDPR, sehingga memungkinkan Anda memenuhi persyaratan peraturan.
- **Integrasi dengan AWS Ekosistem:** Sebagai bagian dari AWS ekosistem, DynamoDB terintegrasi secara mulus dengan layanan AWS lain, seperti, dan. AWS Lambda AWS CloudFormation AWS AppSync Integrasi ini memungkinkan Anda membangun arsitektur nirserver, memanfaatkan infrastruktur sebagai kode, dan membuat aplikasi berbasis data waktu nyata.

Pertimbangan saat memigrasikan basis data relasional ke DynamoDB

Sistem basis data relasional dan basis data NoSQL memiliki keunggulan dan kelemahan yang berbeda. Perbedaan ini membuat desain basis data menjadi berbeda di antara kedua sistem.

	Basis data relasional	Basis data NoSQL
Mengkueri basis data	<p>Di basis data relasional, data dapat dikueri secara fleksibel, tetapi kueri relatif mahal dan tidak dapat diskalakan dengan baik dalam situasi lalu lintas tinggi (lihat Langkah pertama untuk memodelkan data relasional di DynamoDB). Aplikasi basis data relasional dapat menerapkan logika bisnis dalam prosedur tersimpan, subkueri SQL, kueri pembaruan massal, dan kueri agregasi.</p>	<p>Dalam basis data NoSQL seperti DynamoDB, data dapat dikueri secara efisien dalam sejumlah cara terbatas, di luar itu kueri bisa jadi mahal dan lambat. Penulisan ke DynamoDB merupakan singleton. Logika bisnis aplikasi yang sebelumnya dijalankan dalam prosedur tersimpan harus difaktorkan ulang untuk berjalan di luar DynamoDB dalam kode khusus yang berjalan pada host seperti Amazon Amazon EC2 atau. AWS Lambda</p>
Merancang basis data	<p>Anda merancang untuk fleksibilitas tanpa perlu mengkhawatirkan detail penerapan atau performa. Optimalisasi kueri umumnya tidak memengaruhi desain skema, tetapi normalisasi itu penting.</p>	<p>Anda merancang skema secara spesifik untuk membuat kueri yang paling umum dan penting seefisien dan seterjangkau mungkin. Struktur data Anda disesuaikan dengan kebutuhan spesifik kasus penggunaan bisnis Anda.</p>

Merancang untuk basis data NoSQL membutuhkan pola pikir yang berbeda dari merancang untuk sistem manajemen basis data relasional (RDBMS). Untuk RDBMS, Anda dapat membuat model data yang dinormalisasi tanpa memikirkan pola akses. Anda kemudian dapat memperluasnya nanti ketika ada pertanyaan dan persyaratan kueri baru. Anda dapat mengatur setiap jenis data ke dalam tabelnya sendiri.

Dengan desain NoSQL, Anda tidak boleh mulai merancang skema untuk DynamoDB sampai Anda mengetahui pertanyaan yang perlu dijawab. Memahami masalah bisnis dan pola baca dan tulis aplikasi sangatlah penting. Anda juga harus berusaha mempertahankan tabel sesedikit mungkin dalam aplikasi DynamoDB. Memiliki lebih sedikit tabel membuat segala sesuatunya lebih teratur, memerlukan lebih sedikit manajemen izin, dan mengurangi overhead untuk aplikasi DynamoDB Anda. Hal ini juga dapat membantu menjaga biaya pencadangan tetap rendah secara keseluruhan.

Tugas memodelkan data relasional untuk DynamoDB dan membangun versi baru aplikasi front-end merupakan [topik terpisah](#). Panduan ini mengasumsikan Anda memiliki versi baru aplikasi yang dibuat untuk menggunakan DynamoDB, tetapi Anda masih perlu menentukan cara terbaik untuk memigrasikan dan menyinkronkan data historis selama cutover.

Pertimbangan Ukuran

Ukuran maksimum setiap item (baris) yang Anda simpan dalam tabel DynamoDB adalah 400KB. Untuk informasi selengkapnya, lihat [Kuota dan batas](#). Ukuran item ditentukan oleh ukuran total semua nama atribut dan nilai atribut dalam item. Untuk informasi selengkapnya, lihat [the section called “Ukuran dan format item”](#).

Jika aplikasi Anda perlu menyimpan lebih banyak data dalam item daripada batas ukuran DynamoDB yang diizinkan, pisahkan item menjadi koleksi item, kompres data item, atau simpan item sebagai objek di Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) sambil menyimpan pengidentifikasi objek Amazon S3 di item DynamoDB Anda. Lihat [the section called “Item besar”](#). Biaya untuk memperbarui item didasarkan pada ukuran penuh item. Untuk beban kerja yang memerlukan pembaruan yang sering ke item yang ada, memiliki item kecil satu atau dua KB akan lebih murah untuk memperbarui daripada item yang lebih besar. Lihat [the section called “Bekerja dengan Kumpulan Item”](#) untuk informasi lebih lanjut tentang koleksi item.

Saat memilih partisi dan mengurutkan atribut kunci, pengaturan tabel lainnya, ukuran dan struktur item, dan apakah akan membuat indeks sekunder, pastikan untuk meninjau dokumentasi [Pemodelan DynamoDB](#) serta panduan untuk [the section called “Optimasi biaya”](#). Pastikan untuk menguji rencana migrasi Anda sehingga solusi DynamoDB Anda hemat biaya dan sesuai dengan fitur dan batasan DynamoDB.

Memahami cara kerja migrasi ke DynamoDB

Sebelum meninjau alat migrasi yang tersedia bagi kami, pertimbangkan cara penulisan diproses oleh DynamoDB.

Note

DynamoDB otomatis memecah dan mendistribusikan data Anda ke beberapa server bersama dan lokasi penyimpanan sehingga tidak ada cara langsung untuk mengimpor set data besar secara langsung ke server produksi.

Operasi tulis default dan paling umum adalah operasi API [PutItem](#) tunggal. Anda dapat melakukan operasi `PutItem` dalam satu putaran untuk memproses kumpulan data. DynamoDB mendukung koneksi bersamaan yang hampir tidak terbatas, jadi dengan asumsi Anda dapat mengonfigurasi dan menjalankan rutinitas pemuatan multi-utas besar-besaran MapReduce seperti atau Spark, kecepatan penulisan hanya dibatasi oleh kapasitas tabel target (yang juga umumnya tidak terbatas).

Saat memuat data ke DynamoDB, penting untuk memahami kecepatan tulis loader Anda. Jika item (baris) yang Anda muat berukuran 1KB atau kurang, kecepatan ini hanyalah jumlah item per detik. Tabel target kemudian dapat disediakan dengan WCU (unit kapasitas tulis) yang memadai untuk menangani tingkat ini. Jika loader Anda melebihi kapasitas yang disediakan dalam detik tertentu, permintaan tambahan dapat mengalami throttling atau ditolak sama sekali. Anda dapat memeriksa throttle di CloudWatch bagan yang ditemukan di tab pemantauan konsol DynamoDB.

Operasi kedua yang dapat dilakukan adalah dengan API terkait yang disebut [BatchWriteItem](#). `BatchWriteItem` memungkinkan Anda untuk menggabungkan hingga 25 permintaan tulis ke dalam satu panggilan API. Permintaan ini diterima oleh layanan dan diproses sebagai `PutItem` permintaan terpisah ke tabel. Saat memilih `BatchWriteItem`, Anda tidak akan mendapatkan keuntungan dari percobaan ulang otomatis yang disertakan dengan AWS SDK saat melakukan panggilan tunggal dengan `PutItem`. Jadi, jika ada kesalahan (seperti pengecualian throttling), Anda harus mencari daftar penulisan yang gagal pada panggilan respons ke `BatchWriteItem`. Untuk informasi selengkapnya tentang penanganan peringatan pelambatan jika ini terdeteksi di bagan CloudWatch pelambatan, lihat [the section called "Throttling"](#)

Jenis impor data ketiga dimungkinkan dengan [fitur impor DynamoDB dari S3](#). Fitur ini memungkinkan Anda untuk menampilkan set data besar di Amazon S3 dan meminta DynamoDB untuk mengimpor data secara otomatis ke tabel baru. Impor ini tidak instan dan memerlukan waktu yang sebanding dengan ukuran set data. Namun, menghadirkan kemudahan karena tidak memerlukan platform ETL atau kode DynamoDB khusus untuk ditulis. Fitur impor ini memiliki batasan yang membuatnya cocok untuk migrasi saat waktu henti dapat diterima. Data dari S3 dimuat ke dalam tabel baru yang dibuat oleh impor, dan tidak tersedia untuk memuat data ke tabel yang ada. Tidak ada transformasi data,

sehingga memerlukan proses hulu untuk menyiapkan dan menyimpan data dalam format akhir ke bucket S3.

Alat untuk membantu migrasi ke DynamoDB

Ada beberapa alat migrasi dan ETL umum yang dapat Anda gunakan untuk memigrasikan data ke DynamoDB.

Banyak pelanggan memilih untuk menulis skrip migrasi dan pekerjaan mereka sendiri untuk membangun transformasi data khusus untuk proses migrasi. Jika berencana mengoperasikan tabel DynamoDB bervolume tinggi dengan lalu lintas tulis yang padat atau pekerjaan pemuatan massal yang besar secara reguler, Anda dapat membuat alat migrasi sendiri untuk menghindari kekhawatiran dengan perilaku DynamoDB di bawah lalu lintas tulis yang padat. Skenario seperti penanganan throttle dan penyediaan tabel yang efisien dapat dialami di awal proyek saat melakukan migrasi praktik.

Amazon menyediakan sejumlah alat data yang dapat dimanfaatkan, termasuk [AWS Database Migration Service \(DMS\)](#), [AWS Glue](#), [Amazon EMR](#), dan [Amazon Managed Streaming for Apache Kafka](#). Semua alat ini dapat digunakan untuk melakukan migrasi waktu henti, dan alat tertentu yang dapat memanfaatkan fitur Change Data Capture (CDC) basis data relasional juga dapat mendukung migrasi online. Saat memilih alat terbaik, ada baiknya mempertimbangkan keahlian dan pengalaman yang dimiliki organisasi Anda dengan setiap alat beserta fitur, performa, dan biaya masing-masing alat.

Memilih strategi yang tepat untuk migrasi ke DynamoDB

Sebuah aplikasi basis data relasional besar dapat menjangkau seratus atau lebih tabel dan mendukung beberapa fungsi aplikasi yang berbeda. Saat melakukan migrasi besar, pertimbangkan untuk memecah aplikasi Anda menjadi komponen yang lebih kecil atau layanan mikro, dan memigrasikan sekumpulan tabel kecil dalam satu waktu. Anda kemudian dapat memigrasikan komponen lainnya ke DynamoDB secara bertahap.

Saat memilih strategi migrasi, parameter tertentu dapat mengarahkan Anda ke satu solusi atau solusi lainnya. Kami dapat menyajikan opsi-opsi ini dalam pohon keputusan untuk menyederhanakan opsi yang tersedia bagi kami berdasarkan kebutuhan dan sumber daya yang tersedia. Konsep-konsep tersebut disebutkan secara singkat di sini (tetapi akan dibahas lebih mendalam nanti dalam panduan ini):

- [Migrasi offline](#): jika aplikasi Anda dapat menoleransi beberapa waktu henti selama migrasi, hal ini akan sangat menyederhanakan proses migrasi.
- [Migrasi hibrid](#): pendekatan ini akan memungkinkan waktu aktif parsial selama migrasi, seperti mengizinkan membaca tetapi tidak menulis, atau mengizinkan membaca dan menyisipkan tetapi tidak memperbarui dan menghapus.
- [Migrasi online](#): aplikasi yang mengharuskan tidak adanya waktu henti selama migrasi tidak mudah dimigrasi, dan mungkin memerlukan perencanaan dan pengembangan khusus yang signifikan. Salah satu keputusan utama adalah memperkirakan dan menimbang biaya membuat proses migrasi khusus versus biaya untuk bisnis yang memiliki periode waktu henti selama cutover.

Jika	Dan	Maka
Anda boleh menghapus aplikasi selama beberapa waktu selama periode pemeliharaan untuk melakukan migrasi data. Ini adalah migrasi offline		Menggunakan AWS DMS dan melakukan migrasi offline menggunakan tugas pemuatan penuh. Bentuk data sumber terlebih dahulu dengan SQL VIEW jika diinginkan.
Anda boleh menjalankan aplikasi dalam mode hanya-baca selama		Nonaktifkan penulisan di dalam aplikasi atau basis data sumber. Menggunakan AWS DMS dan melakukan migrasi offline menggunakan tugas pemuatan penuh.

Jika	Dan	Maka
<p>migrasi. Ini adalah migrasi hibrid</p> <p>Anda boleh menjalankan aplikasi dengan pembacaan dan sisipan catatan baru, tetapi tidak boleh menjalankan pembaruan atau penghapusan, selama migrasi. Ini adalah migrasi hibrid</p>	<p>Anda memiliki keterampilan pengembangan aplikasi dan dapat memperbaiki aplikasi relasional yang ada untuk melakukan penulisan ganda termasuk ke DynamoDB, untuk semua catatan baru</p>	<p>Menggunakan AWS DMS dan melakukan migrasi offline menggunakan tugas pemuatan penuh. Secara bersamaan, deploy versi aplikasi yang ada yang memungkinkan pembacaan dan melakukan penulisan ganda.</p>
<p>Anda memerlukan migrasi dengan waktu henti minimal. Ini adalah migrasi online</p>	<p>Anda memigrasikan tabel sumber 1 per 1 ke DynamoDB tanpa perubahan skema besar</p>	<p>Gunakan AWS DMS untuk melakukan migrasi data online. Menjalankan tugas beban massal diikuti dengan tugas sinkronisasi CDC</p>

Jika	Dan		Maka
	Anda menggabungkan tabel sumber menjadi lebih sedikit tabel DynamoDB dengan mengikuti filosofi tabel tunggal	Anda memiliki keterampilan pengembangan basis data backend dan kapasitas cadangan pada host SQL	Buat tabel siap NoSQL dalam basis data SQL. Mengisi dan menyinkronkannya menggunakan JOIN, UNION, VIEW, pemicu, prosedur tersimpan
		Anda tidak memiliki keterampilan pengembangan basis data backend dan kapasitas cadangan pada host SQL	Pertimbangkan pendekatan migrasi hibrid atau offline
	Anda boleh melewatkan migrasi data transaksi historis, atau dapat mengarsipkannya di Amazon S3 sebagai pengganti migrasi. Anda hanya perlu memigrasikan beberapa tabel statis kecil		Tulis skrip atau gunakan alat ETL apa pun untuk memigrasikan tabel. Bentuk data sumber terlebih dahulu dengan SQL VIEW jika diinginkan.

Melakukan migrasi offline ke DynamoDB

Migrasi offline cocok ketika Anda dapat mengizinkan periode waktu henti untuk melakukan migrasi. Basis data relasional biasanya membutuhkan setidaknya beberapa waktu henti setiap bulan untuk pemeliharaan dan patching, atau mungkin membutuhkan waktu henti lebih lama untuk peningkatan perangkat keras atau peningkatan rilis utama.

Amazon S3 dapat digunakan sebagai area penahanan selama migrasi. Data yang disimpan dalam CSV (nilai dipisahkan koma) atau format DynamoDB JSON dapat diimpor ke tabel DynamoDB baru secara otomatis menggunakan [fitur impor DynamoDB dari S3](#).

Rencana

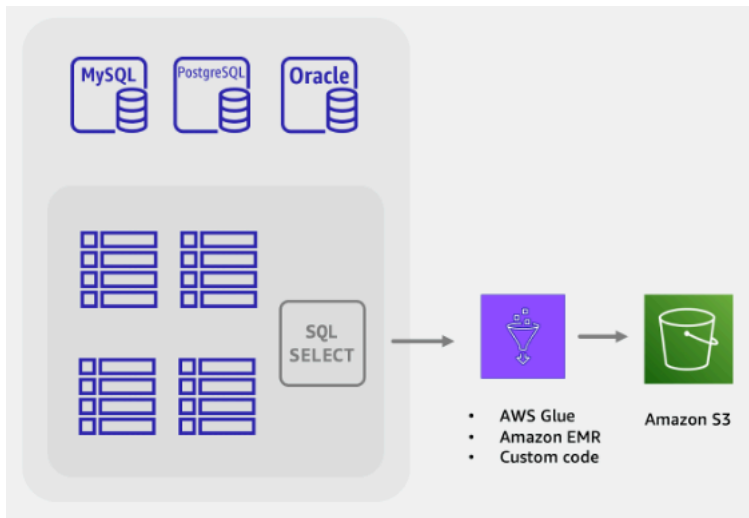
Melakukan migrasi offline menggunakan Amazon Amazon S3

Alat

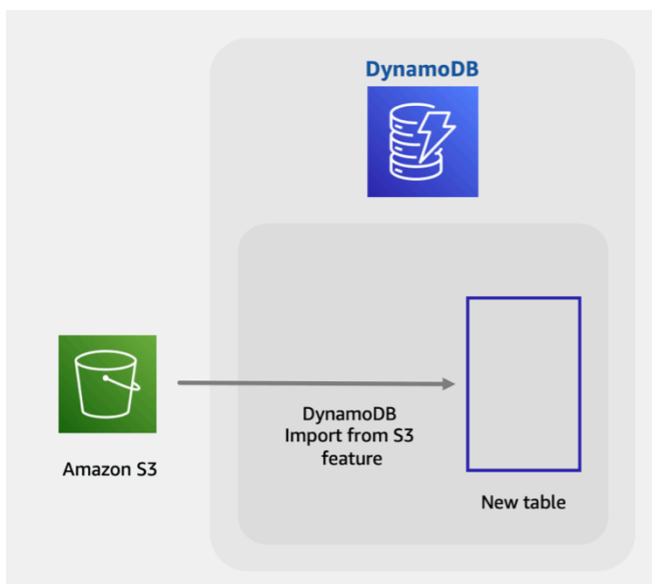
- Pekerjaan ETL untuk mengekstrak dan mengubah data SQL serta menyimpannya dalam bucket S3 seperti:
 - AWS Glue
 - Amazon EMR
 - Kode khusus Anda sendiri
- Fitur impor DynamoDB dari S3

Langkah-langkah migrasi offline:

1. Buat pekerjaan ETL yang dapat menyalin basis data SQL, mengubah data tabel menjadi DynamoDB JSON atau format CSV, dan menyimpannya ke bucket S3.



2. Fitur Impor DynamoDB dari S3 dipanggil untuk membuat tabel baru dan memuat data secara otomatis dari bucket S3 Anda.



Migrasi offline sangat sederhana dan mudah, tetapi mungkin tidak populer di kalangan pemilik aplikasi dan pengguna. Pengguna akan mendapatkan keuntungan jika aplikasi dapat memberikan tingkat layanan yang lebih rendah selama migrasi, daripada tidak memberikan layanan sama sekali.

Anda dapat menambahkan fungsionalitas untuk menonaktifkan penulisan selama migrasi offline, sambil mengizinkan pembacaan berlanjut seperti biasa. Pengguna aplikasi masih dapat menelusuri dan menanyakan data yang ada dengan aman saat data relasional sedang dimigrasikan. Jika ini yang Anda cari, lanjutkan membaca untuk mempelajari tentang [migrasi hibrid](#).

Melakukan migrasi hibrid ke DynamoDB

Meskipun semua aplikasi basis data melakukan operasi baca dan tulis, jenis operasi tulis yang dilakukan harus dipertimbangkan saat merencanakan migrasi hibrid atau online. Penulisan basis data dapat diklasifikasikan ke dalam tiga bucket: sisipan, pembaruan, dan penghapusan. Beberapa aplikasi tidak melakukan pembaruan apa pun pada catatan yang ada. Aplikasi lain mungkin tidak mengharuskan penghapusan untuk segera diproses, dan dapat menunda penghapusan ke proses pembersihan massal di akhir bulan, misalnya. Jenis aplikasi ini bisa jadi lebih mudah untuk dimigrasi sambil memungkinkan waktu aktif parsial.

Rencana

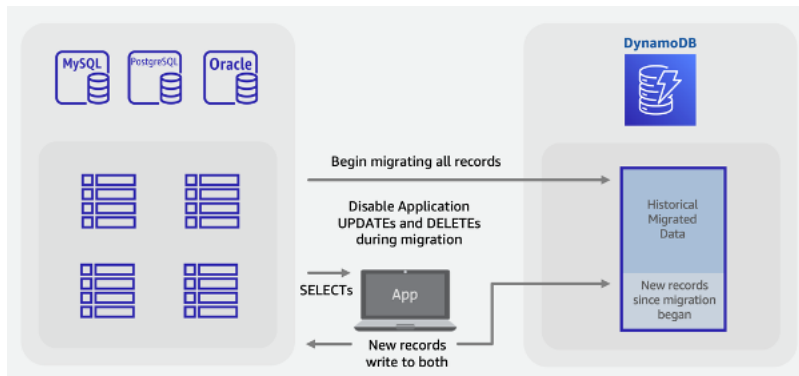
Lakukan migrasi online/offline hybrid dengan penulisan ganda aplikasi

Alat

- Pekerjaan ETL untuk mengekstrak dan mengubah data SQL serta menyimpannya dalam bucket S3 seperti:
 - AWS Glue
 - Amazon EMR
 - Kode khusus Anda sendiri

Langkah-langkah migrasi hibrid:

1. Buat tabel DynamoDB target. Tabel ini akan menerima data massal historis dan data langsung baru
2. Membuat versi aplikasi lama yang penghapusan dan pembaruannya dinonaktifkan saat melakukan semua penyisipan sebagai penulisan ganda ke basis data SQL dan DynamoDB
3. Memulai pekerjaan ETL untuk memigrasikan data yang ada dan men-deploy versi aplikasi baru secara bersamaan
4. Setelah pekerjaan ETL selesai, DynamoDB akan memiliki semua catatan yang ada dan baru, dan siap untuk cutover aplikasi



Note

Pekerjaan ETL menulis langsung dari SQL ke DynamoDB. Kami tidak dapat menggunakan fitur impor S3 seperti pada contoh migrasi offline, karena tabel target tidak bersifat publik dan tersedia untuk penulisan lain hingga seluruh impor selesai.

Melakukan migrasi online ke DynamoDB dengan memigrasikan setiap tabel 1:1

Banyak basis data relasional memiliki fitur yang disebut Change Data Capture (CDC), yaitu basis data memungkinkan pengguna untuk meminta daftar perubahan pada tabel yang terjadi sebelum atau setelah titik waktu tertentu. CDC menggunakan log internal untuk mengaktifkan fitur ini dan tidak mengharuskan tabel untuk memiliki kolom stempel waktu agar dapat berfungsi.

Saat memigrasikan skema tabel SQL ke basis data NoSQL, Anda mungkin ingin menggabungkan dan membentuk kembali data Anda menjadi tabel yang lebih sedikit. Melakukannya akan memungkinkan Anda mengumpulkan data di satu tempat dan menghindari keharusan untuk menggabungkan data terkait secara manual dalam operasi baca multi-langkah. Namun, pembentukan data tabel tunggal tidak selalu diperlukan dan terkadang Anda akan memigrasikan tabel 1 per 1 ke DynamoDB. Migrasi tabel 1 per 1 ini tidak terlalu rumit karena Anda dapat memanfaatkan fitur CDC basis data sumber, menggunakan alat ETL umum yang mendukung jenis migrasi ini. Data untuk setiap baris masih dapat diubah ke dalam format baru, tetapi cakupan setiap tabel tetap sama.

Pertimbangkan untuk memigrasikan tabel SQL 1 per 1 ke DynamoDB, dengan peringatan bahwa tidak ada gabungan sisi server.

Rencana

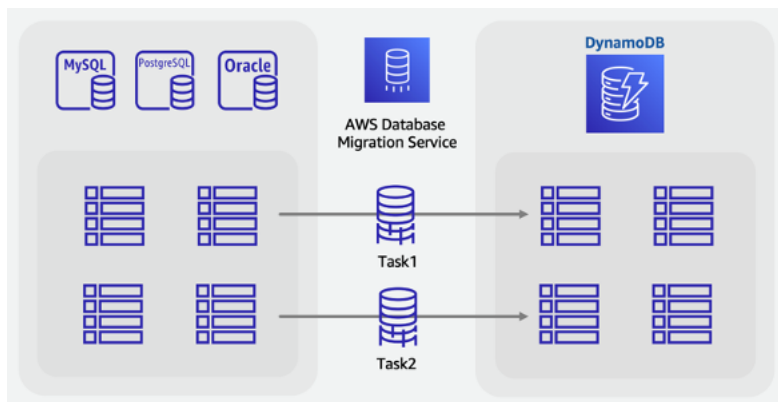
Lakukan migrasi online dari setiap tabel ke DynamoDB menggunakan AWS DMS

Alat

- [AWS Database Migration Service \(DMS\)](#), alat ETL yang dapat memuat data historis secara massal dan juga memanfaatkan CDC untuk menyinkronkan tabel sumber dan target

Langkah-langkah migrasi online:

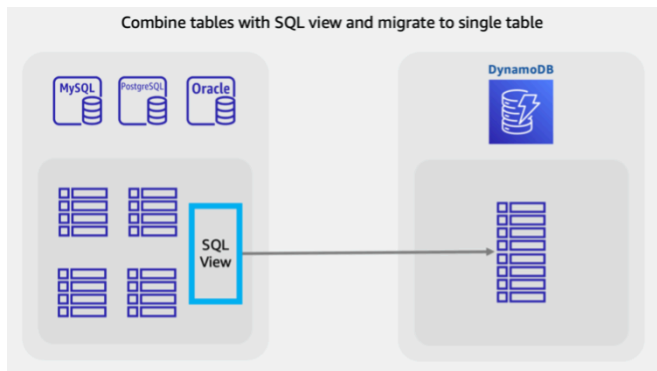
1. Identifikasi tabel dalam skema sumber yang akan dimigrasikan
2. Buat jumlah tabel yang sama di DynamoDB dengan struktur kunci yang serupa
3. Buat server replikasi AWS DMS dan konfigurasi titik akhir sumber dan target
4. Tentukan transformasi per baris yang diperlukan (seperti kolom gabungan atau konversi tanggal ke format string ISO-8601)
5. Buat tugas migrasi setiap tabel untuk Beban Penuh dan Change Data Capture
6. Pantau tugas-tugas ini sampai fase replikasi yang sedang berlangsung dimulai
7. Pada titik ini Anda dapat melakukan audit validasi, lalu mengalihkan pengguna ke aplikasi yang membaca dan menulis ke DynamoDB



Lakukan migrasi online ke DynamoDB menggunakan tabel penahanan khusus

Anda sebaiknya menggabungkan tabel untuk memanfaatkan pola akses NoSQL yang unik (misalnya, mengubah empat tabel lama menjadi satu tabel DynamoDB tunggal). Permintaan dokumen nilai kunci tunggal, atau kueri untuk kumpulan item yang dikelompokkan sebelumnya biasanya akan menghasilkan latensi yang lebih baik daripada basis data SQL yang melakukan gabungan multi-

tabel. Namun, hal ini membuat tugas migrasi menjadi lebih sulit. SQL VIEW dapat melakukan pekerjaan dalam basis data sumber untuk menyiapkan satu set data yang mewakili keempat tabel dalam satu set.



Tampilan ini mungkin JOIN tabel menjadi bentuk yang didenormalisasi, atau sebaliknya, menjaga entitas tetap normal dan menumpuk tabel menggunakan SQL. UNION Keputusan penting seputar pembentukan kembali data relasional tercakup dalam [video ini](#). Untuk migrasi offline, menggunakan tampilan untuk menggabungkan tabel adalah cara terbaik untuk membentuk data untuk skema tabel tunggal DynamoDB.

Namun, untuk migrasi online dengan data langsung dan berubah, Anda tidak dapat memanfaatkan fitur CDC karena fitur tersebut hanya didukung untuk kueri tabel tunggal, bukan dari dalam VIEW. Jika tabel Anda menyertakan kolom stempel waktu terakhir diperbarui, dan tabel tersebut digabungkan ke dalam VIEW, Anda kemudian dapat membuat pekerjaan ETL khusus yang menggunakan tabel tersebut untuk mencapai beban massal dengan sinkronisasi.

Pendekatan baru untuk tantangan ini adalah dengan menggunakan fitur SQL standar seperti tampilan, prosedur tersimpan, dan pemicu untuk membuat tabel SQL baru dalam format akhir DynamoDB NoSQL yang diinginkan.

Jika server basis data Anda dapat mengalokasikan sejumlah ruang penyimpanan tambahan, tabel penahanan tunggal ini dapat dibuat sebelum migrasi dimulai. Hal ini dapat dicapai dengan menulis prosedur tersimpan yang akan membaca dari tabel yang ada, mengubah data sesuai kebutuhan, dan menulis ke tabel penahanan baru. Serangkaian pemicu dapat ditambahkan untuk mereplikasi perubahan dalam tabel ke dalam tabel penahanan secara waktu nyata. Jika pemicu tidak diperbolehkan karena kebijakan perusahaan, perubahan pada prosedur tersimpan dapat mencapai hasil yang sama. Anda akan menambahkan beberapa baris kode ke prosedur apa pun yang menulis data, untuk menambahkan perubahan yang sama ke dalam tabel penahanan.

Memiliki tabel penahanan ini yang sepenuhnya disinkronkan dengan tabel aplikasi lama akan memberi Anda titik awal yang bagus untuk migrasi langsung. Alat yang menggunakan database CDC

untuk menyelesaikan migrasi langsung, seperti AWS DMS, sekarang dapat digunakan terhadap tabel ini. Keuntungan dari pendekatan ini adalah penggunaan keterampilan dan fitur SQL terkenal yang tersedia di mesin basis data relasional.

Rencana

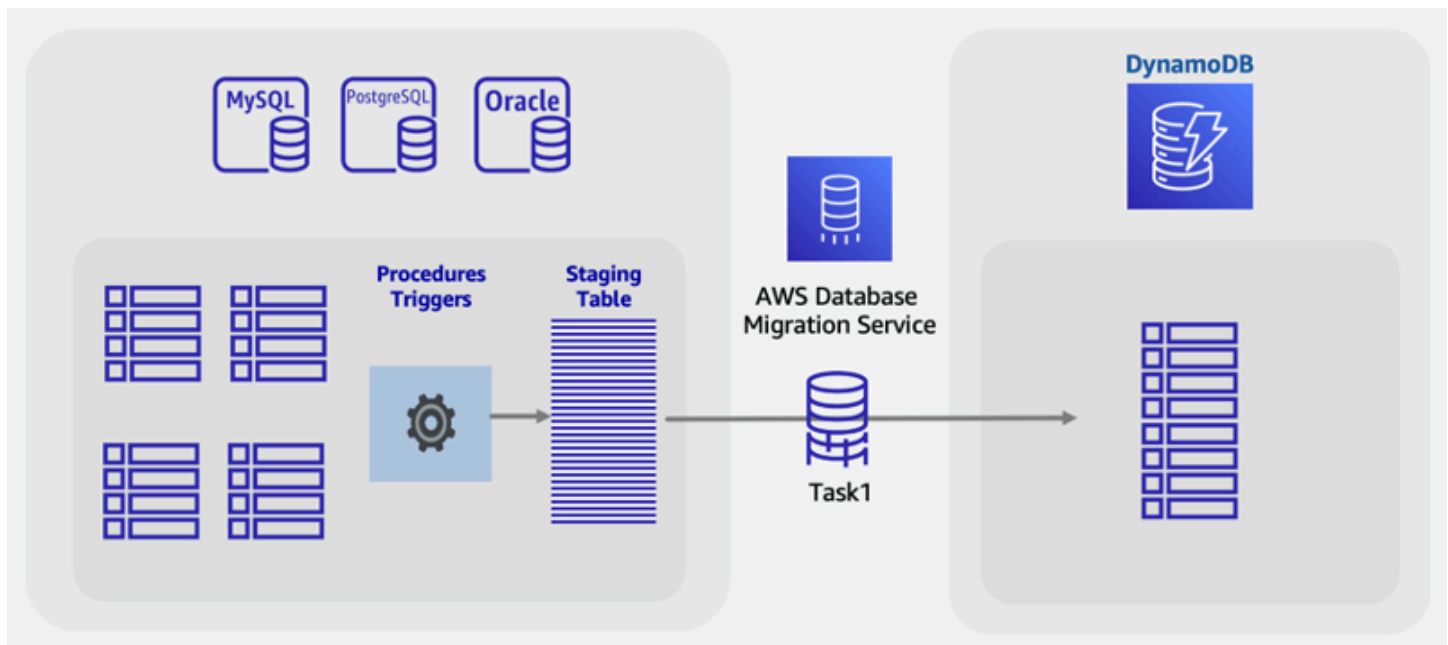
Lakukan migrasi online dengan tabel penahanan SQL menggunakan AWS DMS

Alat

- Prosedur atau pemicu tersimpan SQL khusus
- [AWS Database Migration Service \(DMS\)](#), alat ETL yang dapat memigrasikan tabel penahanan langsung ke DynamoDB

Langkah-langkah migrasi online:

1. Dalam mesin basis data relasional sumber, pastikan ada beberapa kapasitas pemrosesan dan ruang disk tambahan.
2. Buat tabel penahanan baru di basis data SQL, dengan stempel waktu atau fitur CDC diaktifkan
3. Tulis dan jalankan prosedur tersimpan untuk menyalin data tabel relasional yang ada ke dalam tabel penahanan
4. Deploy pemicu atau ubah prosedur yang ada untuk penulisan ganda ke tabel penahanan baru saat melakukan penulisan normal ke tabel yang ada
5. Jalankan AWS DMS untuk memigrasi dan menyinkronkan tabel sumber ini ke tabel DynamoDB target



Panduan ini menyajikan beberapa pertimbangan dan pendekatan untuk memigrasikan data basis data relasional ke DynamoDB, dengan fokus pada meminimalkan waktu henti dan menggunakan alat dan teknik basis data umum. Untuk informasi selengkapnya, lihat hal berikut:

- [AWS DMS Panduan Pengguna](#)
- [AWS Glue Panduan Pengguna](#)
- [Praktik Terbaik untuk Bermigrasi dari RDBMS ke DynamoDB](#)

NoSQL Workbench untuk DynamoDB

NoSQL Workbench untuk Amazon DynamoDB adalah aplikasi GUI sisi klien lintas platform yang dapat Anda gunakan untuk pengembangan dan operasi database modern. Aplikasi ini tersedia untuk Windows, macOS, dan Linux. NoSQL Workbench adalah alat pengembangan visual yang menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri untuk membantu Anda merancang, membuat, melakukan kueri, serta mengelola tabel DynamoDB. NoSQL Workbench kini menyertakan DynamoDB lokal sebagai bagian opsional dari proses instalasi, yang mempermudah Anda untuk memodelkan data Anda di DynamoDB lokal. Untuk mempelajari selengkapnya tentang DynamoDB lokal dan persyaratannya, lihat [Menyiapkan DynamoDB lokal \(versi yang dapat diunduh\)](#).

Pemodelan data

Dengan NoSQL Workbench untuk DynamoDB, Anda dapat membuat model data baru dari, atau mendesain model berdasarkan, model data yang sudah ada yang memenuhi pola akses data aplikasi Anda. Anda juga dapat mengimpor dan mengekspor model data yang didesain pada akhir proses. Untuk informasi selengkapnya, lihat [Membangun model data dengan NoSQL Workbench](#).

Visualisasi data

Pemvisualisasi model data menyediakan kanvas tempat Anda dapat memetakan kueri dan memvisualisasikan pola akses (faset) aplikasi tanpa harus menulis kode. Setiap faset berhubungan dengan pola akses yang berbeda di DynamoDB. Anda dapat membuat data sampel secara otomatis untuk digunakan dalam model data Anda. Untuk informasi selengkapnya, lihat [Memvisualisasikan pola akses data](#).

Pembangunan operasi

NoSQL Workbench menyediakan antarmuka pengguna grafis yang kaya bagi Anda untuk mengembangkan dan menguji kueri. Anda dapat menggunakan pembangun operasi untuk melihat, menjelajahi, dan melakukan kueri set data langsung. Builder operasi terstruktur mendukung ekspresi proyeksi, ekspresi kondisi, dan menghasilkan kode sampel dalam berbagai bahasa. Anda dapat langsung mengkloning tabel dari satu akun Amazon DynamoDB ke akun lain di Wilayah yang berbeda. Anda juga dapat langsung mengkloning tabel antara DynamoDB lokal dan akun Amazon DynamoDB untuk menyalin skema kunci tabel Anda dengan lebih cepat (dan skema dan item GSI opsional) di antara lingkungan pengembangan Anda. Untuk informasi selengkapnya, lihat [Menjelajahi set data dan membangun operasi dengan NoSQL Workbench](#).

Topik

- [Unduh NoSQL Workbench untuk DynamoDB](#)
- [Menginstal NoSQL Workbench untuk DynamoDB](#)
- [Membangun model data dengan NoSQL Workbench](#)
- [Memvisualisasikan pola akses data](#)
- [Menjelajahi set data dan membangun operasi dengan NoSQL Workbench](#)
- [Model data Sampel untuk NoSQL Workbench](#)
- [Riwayat rilis untuk NoSQL Workbench](#)

Unduh NoSQL Workbench untuk DynamoDB

Ikuti petunjuk ini untuk mengunduh NoSQL Workbench dan DynamoDB lokal* untuk Amazon DynamoDB.

Prasyarat

Ada dua bagian prasyarat perangkat lunak yang diperlukan untuk menginstal Ubuntu: libfuse2 dan curl.

libfuse2

Pada Ubuntu 22.04, libfuse2 tidak lagi diinstal secara default. Untuk mengatasi ini, jalankan `sudo add-apt-repository universe && sudo apt install libfuse2` untuk menginstal untuk [versi Ubuntu terbaru](#).

keriting

Perbarui Ubuntu, jalankan `sudo apt update && sudo apt upgrade`

Selanjutnya, instal curl, jalankan: `sudo apt install curl`

Untuk mengunduh NoSQL Workbench dan DynamoDB lokal

1. Unduh versi NoSQL Workbench yang sesuai untuk sistem operasi Anda.

Sistem operasi	Tautan unduhan
macOS (Intel) **	Unduh untuk macOS (Intel)

Sistem operasi	Tautan unduhan
macOS (silikon Apple)	Unduh untuk macOS (Apple silicon)
Windows	Unduh untuk Windows
Linux***	Unduh untuk Linux

* NoSQL Workbench termasuk DynamoDB lokal sebagai bagian opsional dari proses instalasi.

** Jika pesan peringatan muncul saat Anda mencoba membuka NoSQL Workbench yang menyatakan bahwa aplikasi tidak terdaftar di Apple oleh pengembang yang diidentifikasi, lakukan hal berikut:

1. Temukan aplikasi dan kemudian buka.
2. Kontrol+klik ikon aplikasi, lalu pilih Buka dari menu pintasan.

Ini menyimpan aplikasi sebagai pengecualian untuk pengaturan keamanan Anda. Buka aplikasi dengan mengklik dua kali sama seperti Anda dapat membuka aplikasi terdaftar apa pun.

*** NoSQL Workbench mendukung Ubuntu 12.04, Fedora 21, dan Debian 8, atau versi yang lebih baru dari distribusi Linux ini.

2. Mulai aplikasi yang Anda unduh, lalu ikuti langkah-langkah di Menginstal NoSQL Workbench.

Note

Java Runtime Environment (JRE) versi 11.x atau yang lebih baru diperlukan untuk menjalankan DynamoDB lokal.

Menginstal NoSQL Workbench untuk DynamoDB

Ikuti langkah-langkah berikut untuk menginstal NoSQL Workbench dan DynamoDB lokal pada platform yang didukung.

Windows

Untuk menginstal NoSQL Workbench di Windows

1. Jalankan aplikasi penginstal NoSQL Workbench dan pilih bahasa pengaturan. Kemudian pilih OK untuk memulai pengaturan. Untuk informasi selengkapnya tentang mengunduh NoSQL Workbench, lihat [Unduh NoSQL Workbench untuk DynamoDB](#).
2. Pilih Selanjutnya untuk melanjutkan pengaturan, lalu pilih Selanjutnya di layar berikut.
3. Secara default, kotak centang Instal DynamoDB Lokal dipilih untuk menyertakan DynamoDB lokal sebagai bagian dari instalasi. Jika opsi ini tetap dipilih, maka DynamoDB lokal akan diinstal, dan jalur tujuan akan sama dengan jalur instalasi NoSQL Workbench. Mengosongkan kotak centang untuk opsi ini akan melewati instalasi DynamoDB lokal, dan jalur instalasi hanya untuk NoSQL Workbench.

Pilih tujuan di mana Anda ingin perangkat lunak diinstal, dan pilih Selanjutnya.

Note

Jika Anda memilih untuk tidak menyertakan DynamoDB lokal sebagai bagian dari pengaturan, kosongkan kotak centang Install DynamoDB Local, pilih Berikutnya, dan lewati ke langkah 6. Anda dapat mengunduh DynamoDB lokal secara terpisah sebagai instalasi mandiri di lain waktu. Untuk informasi selengkapnya, lihat [Menyiapkan DynamoDB lokal \(versi yang dapat diunduh\)](#).

Konfigurasi jalur instalasi Anda selama langkah ini.

4. Pilih nomor port untuk DynamoDB lokal yang akan digunakan. Port default adalah 8000. Setelah Anda memasukkan nomor port, pilih Selanjutnya.
5. Pilih Selanjutnya untuk memulai pengaturan.
6. Setelah pengaturan selesai, pilih Selesai untuk menutup layar pengaturan.
7. Buka aplikasi di jalur instalasi Anda, seperti `/Programs/DynamoDbWorkBench/`.

macOS

Untuk menginstal NoSQL Workbench di macOS

1. Jalankan aplikasi penginstal NoSQL Workbench dan pilih bahasa pengaturan. Kemudian pilih OK untuk memulai pengaturan. Untuk informasi selengkapnya tentang mengunduh NoSQL Workbench, lihat [Unduh NoSQL Workbench untuk DynamoDB](#).
2. Pilih Selanjutnya untuk melanjutkan pengaturan, lalu pilih Selanjutnya di layar berikut.
3. Secara default, kotak centang Instal DynamoDB lokal dipilih untuk menyertakan DynamoDB lokal sebagai bagian dari instalasi. Jika opsi ini tetap dipilih, maka DynamoDB lokal akan diinstal, dan jalur tujuan akan sama dengan jalur instalasi NoSQL Workbench. Mengosongkan opsi ini akan melewati instalasi DynamoDB lokal, dan jalur instalasi hanya untuk NoSQL Workbench.

Pilih tujuan di mana Anda ingin perangkat lunak diinstal, dan pilih Selanjutnya.

Note

Jika Anda memilih untuk tidak menyertakan DynamoDB lokal sebagai bagian dari pengaturan, kosongkan kotak centang Instal DynamoDB lokal, pilih Berikutnya, dan lewati ke langkah 6. Anda dapat mengunduh DynamoDB lokal secara terpisah sebagai instalasi mandiri di lain waktu. Untuk informasi selengkapnya, lihat [Menyiapkan DynamoDB lokal \(versi yang dapat diunduh\)](#).

Konfigurasi jalur instalasi Anda selama langkah ini.

4. Pilih nomor port untuk DynamoDB lokal yang akan digunakan. Port default adalah 8000. Setelah Anda memasukkan nomor port, pilih Selanjutnya.
5. Pilih Selanjutnya untuk memulai pengaturan.
6. Setelah pengaturan selesai, pilih Selesai untuk menutup layar pengaturan.
7. Buka aplikasi di jalur instalasi Anda, seperti `/Applications/DynamoDBWorkbench/`.

Note

NoSQL Workbench untuk macOS melakukan pembaruan otomatis. Untuk mendapatkan notifikasi tentang pembaruan, aktifkan akses notifikasi ke NoSQL Workbench di Preferensi Sistem > Notifikasi.

Linux

Untuk menginstal NoSQL Workbench di Linux

1. Jalankan aplikasi penginstal NoSQL Workbench dan pilih bahasa pengaturan. Kemudian pilih OK untuk memulai pengaturan. Untuk informasi selengkapnya tentang mengunduh NoSQL Workbench, lihat [Unduh NoSQL Workbench untuk DynamoDB](#).
2. Pilih Teruskan untuk melanjutkan pengaturan, dan pilih Teruskan di layar berikut.
3. Secara default, kotak centang Instal DynamoDB lokal dipilih untuk menyertakan DynamoDB lokal sebagai bagian dari instalasi. Jika opsi ini tetap dipilih, maka DynamoDB lokal akan diinstal, dan jalur tujuan akan sama dengan jalur instalasi NoSQL Workbench. Mengosongkan opsi ini akan melewati instalasi DynamoDB lokal, dan jalur instalasi hanya untuk NoSQL Workbench.

Pilih tujuan di mana Anda ingin perangkat lunak diinstal, dan pilih Teruskan.

Note

Jika Anda memilih untuk tidak menyertakan DynamoDB lokal sebagai bagian dari pengaturan, kosongkan kotak centang Instal DynamoDB lokal, pilih Teruskan, dan lewati ke langkah 6. Anda dapat mengunduh DynamoDB lokal secara terpisah sebagai instalasi mandiri di lain waktu. Untuk informasi selengkapnya, lihat [Menyiapkan DynamoDB lokal \(versi yang dapat diunduh\)](#).

Konfigurasi jalur instalasi Anda selama langkah ini.

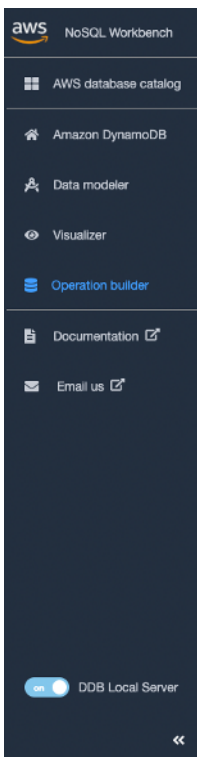
4. Pilih nomor port untuk DynamoDB lokal yang akan digunakan. Port default adalah 8000. Setelah Anda memasukkan nomor port yang dimasukkan, pilih Teruskan.
5. Pilih Teruskan untuk memulai pengaturan.
6. Setelah pengaturan selesai, pilih Selesai untuk menutup layar pengaturan.

7. Buka aplikasi di jalur instalasi Anda, seperti `/usr/local/programs/DynamoDBWorkbench/`.

Note

Jika Anda memilih untuk menginstal DynamoDB lokal sebagai bagian dari instalasi NoSQL Workbench, DynamoDB lokal akan dikonfigurasi sebelumnya dengan opsi default. Untuk mengedit opsi default, ubah LocalStart skrip DDB yang terletak di direktori `/Resources/DDBlocal_scripts/`. Anda dapat menemukannya di jalur yang Anda berikan saat instalasi. Untuk mempelajari selengkapnya tentang opsi lokal DynamoDB, lihat [Catatan penggunaan DynamoDB lokal](#).

Jika Anda memilih untuk menginstal DynamoDB lokal sebagai bagian dari instalasi NoSQL Workbench, Anda akan memiliki akses ke tombol untuk mengaktifkan dan menonaktifkan DynamoDB lokal seperti yang ditunjukkan pada gambar berikut.



Membangun model data dengan NoSQL Workbench

Anda dapat menggunakan alat pemodel data di NoSQL Workbench untuk Amazon DynamoDB untuk membangun model data baru, atau merancang model berdasarkan model data yang ada yang

memenuhi pola akses data aplikasi Anda. Pemodel data mencakup beberapa model data sampel untuk membantu Anda memulai.

Topik

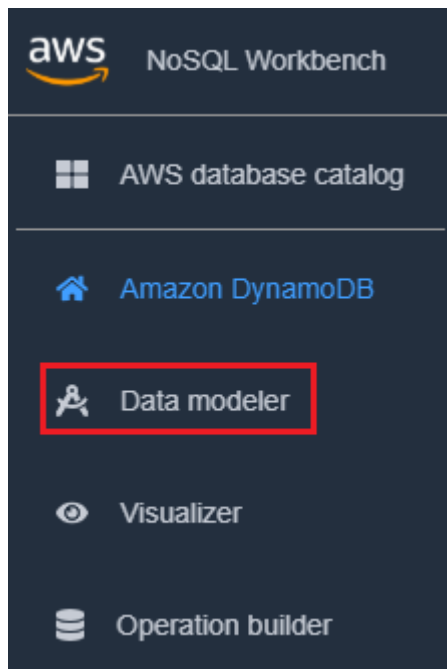
- [Membuat model data baru](#)
- [Mengimpor model data yang ada](#)
- [Mengekspor model data](#)
- [Mengedit model data yang ada](#)

Membuat model data baru

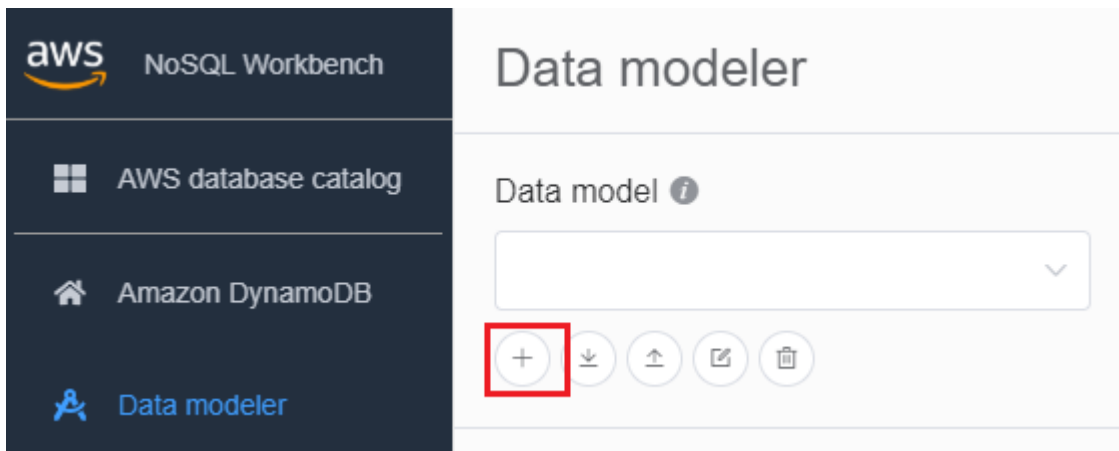
Ikuti langkah-langkah ini untuk membuat model data baru di Amazon DynamoDB menggunakan NoSQL Workbench.

Untuk membuat model data baru

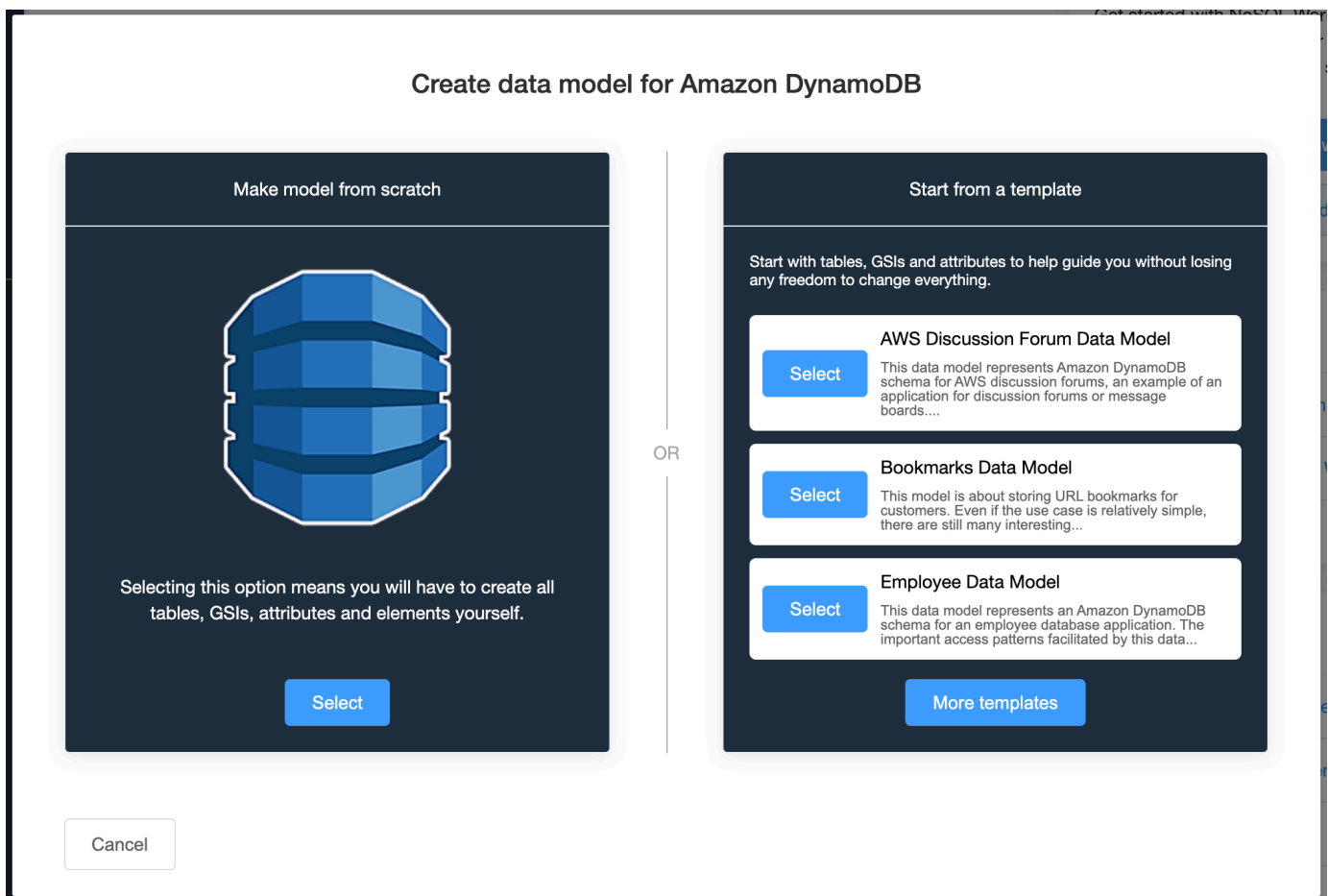
1. Buka NoSQL Workbench, dan di panel navigasi di sisi kiri, pilih ikon Pemodel data.



2. Pilih Buat model data.

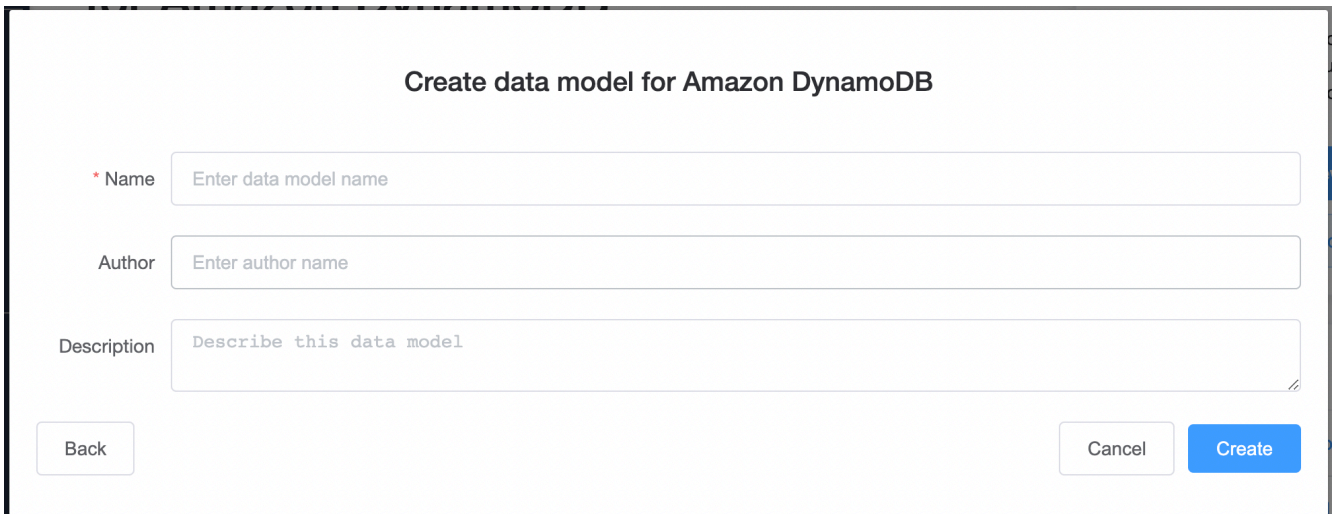


Buat model data memiliki dua pilihan: Membuat model dari awal dan Mulai dari template.



Make model from scratch

Untuk membuat model dari awal, masukkan nama, penulis, dan deskripsi untuk model data. Pilih Buat setelah selesai.



Create data model for Amazon DynamoDB

* Name

Author

Description

Start from a template

Mulai dari template memungkinkan Anda memilih model sampel untuk memulai. Pilih Lebih banyak templat untuk melihat lebih banyak opsi templat. Pilih Pilih untuk template yang ingin Anda gunakan.

Masukkan nama model data, penulis, dan deskripsi untuk template yang Anda pilih. Anda dapat memilih antara Skema saja dan Skema dengan data sampel.

- Skema hanya membuat model data kosong dengan kunci utama (partisi dan kunci sortir) dan atribut lainnya.
- Skema dengan data sampel akan membuat model data lengkap dengan data sampel untuk kunci utama (partisi dan kunci sortir) dan atribut lainnya.

Ketika informasi ini selesai, pilih Buat untuk membuat model.

Create data model for Amazon DynamoDB

Data Model

Template

* Save as

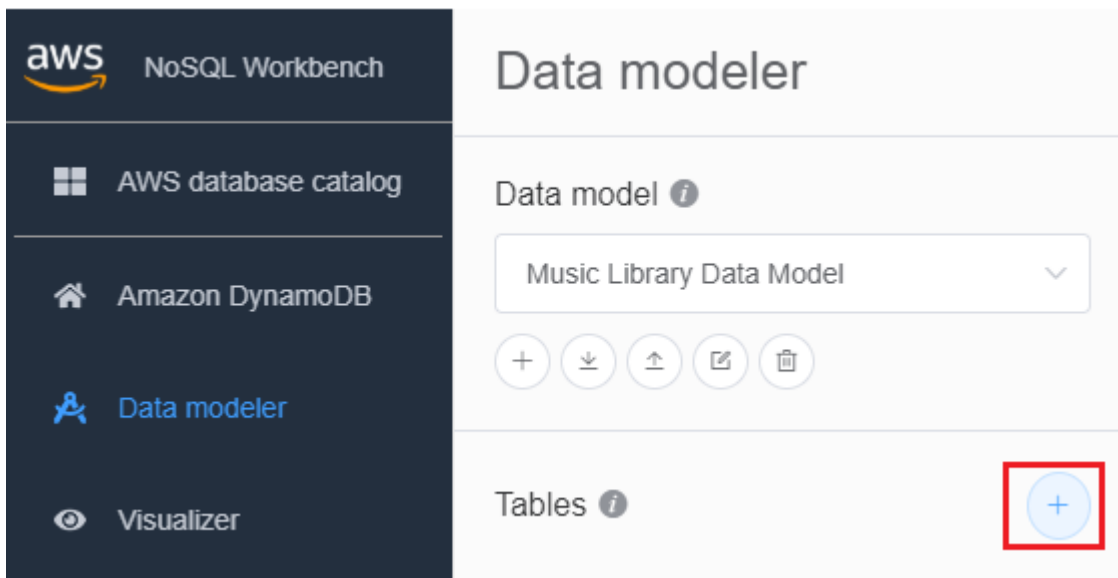
Author

Description

Sample Data

Schema with sample data will create a data model complete with sample data for the primary keys (partition key and/or sort key) and other attributes.

3. Dengan model yang dibuat, pilih Tambahkan tabel.



Untuk informasi selengkapnya tentang tabel, lihat [Bekerja dengan tabel di DynamoDB](#).

4. Tentukan hal berikut:

- Nama tabel - Masukkan nama unik untuk tabel.
- Kunci partisi — Masukkan nama kunci partisi, dan tentukan jenisnya. Secara opsional, Anda juga dapat memilih format tipe data yang lebih terperinci untuk pembuatan data sampel.
- Jika Anda ingin menambahkan kunci pengurutan:
 1. Pilih Tambah kunci pengurutan.
 2. Tentukan nama kunci pengurutan dan tipenya. Secara opsional, Anda dapat memilih format tipe data yang lebih terperinci untuk pembuatan data sampel.

Note

Untuk mempelajari lebih lanjut tentang desain kunci primer, merancang dan menggunakan kunci partisi secara efektif, dan menggunakan tombol sortir, lihat berikut ini:

- [Kunci primer](#)
- [Praktik Terbaik untuk merancang dan menggunakan kunci partisi secara efektif](#)
- [Praktik terbaik untuk menggunakan kunci urutan untuk mengatur data](#)

5. Untuk menambahkan atribut lainnya, lakukan hal berikut untuk setiap atribut:
 1. Pilih Tambahkan atribut.
 2. Tentukan nama atribut dan tipenya. Secara opsional, Anda dapat memilih format tipe data yang lebih terperinci untuk pembuatan data sampel.
6. Tambahkan faset:

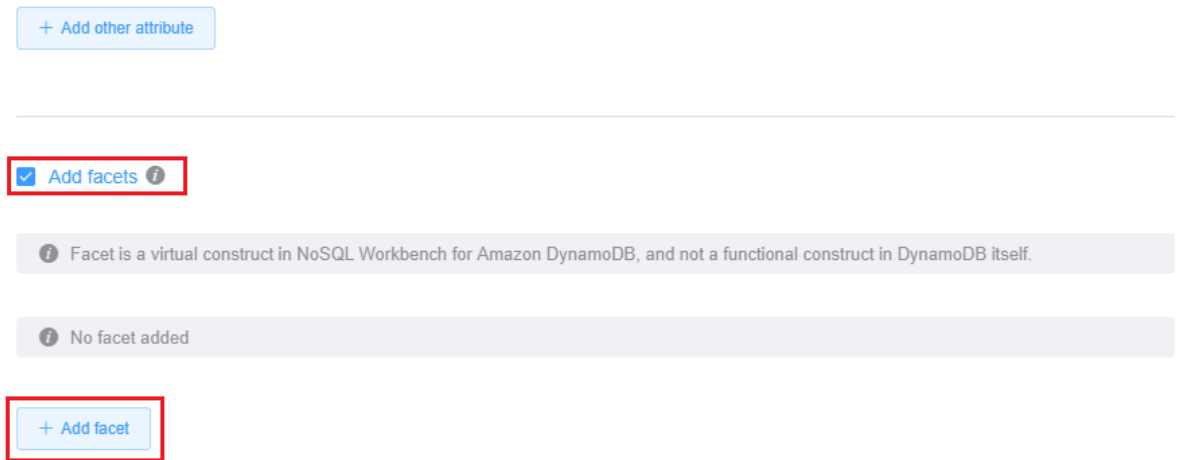
Anda dapat menambahkan facet secara opsional. Sebuah facet adalah konstruksi virtual di NoSQL Workbench. Ini bukan konstruksi fungsional di DynamoDB itu sendiri.

Note

Aspek di NoSQL Workbench membantu Anda memvisualisasikan pola akses data aplikasi yang berbeda untuk Amazon DynamoDB dengan hanya subset data dalam tabel. Untuk mempelajari lebih lanjut tentang aspek, lihat [Melihat pola akses data](#).

Untuk menambahkan facet,

- Pilih Tambahkan faset-faset.
- Pilih Tambahkan faset.



- Tentukan hal berikut:
 - Nama faset.
 - Alias kunci Partisi untuk membantu membedakan tampilan facet ini.
 - Alias kunci pengurutan.
 - Pilih Atribut lainnya yang merupakan bagian dari faset ini.

Pilih Tambahkan faset.

Add facets ?

? Facet is a virtual construct in NoSQL Workbench for Amazon DynamoDB, and not a functional construct in DynamoDB itself.

? No facet added

Add facet

* Facet name

* Partition key alias ?

* Sort key alias ?

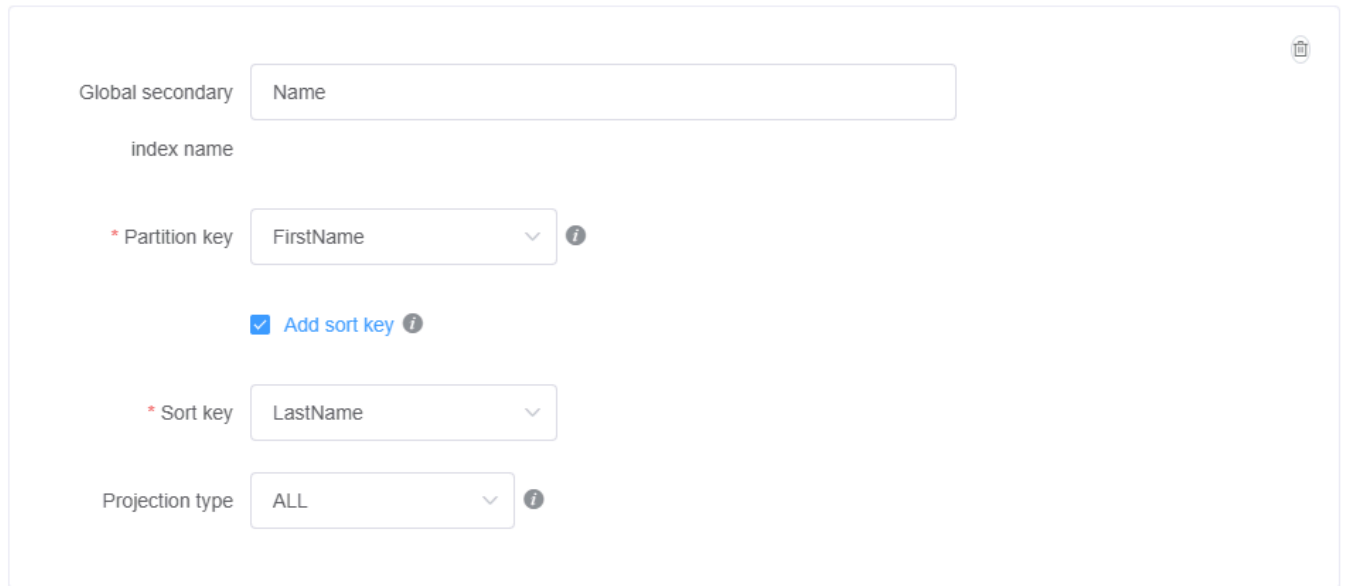
Other attributes ?

Ulangi langkah ini jika Anda ingin menambahkan lebih banyak aspek.

7. Jika Anda ingin menambahkan indeks sekunder global, pilih Tambah indeks sekunder global.

Tentukan Nama indeks sekunder global, atribut Kunci partisi, dan Tipe proyeksi.

Global secondary indexes

[+ Add global secondary index](#)

[Untuk informasi selengkapnya tentang bekerja dengan indeks sekunder global di DynamoDB, lihat Indeks sekunder global.](#)

- Secara default, tabel Anda akan menggunakan mode kapasitas yang disediakan dengan penskalaan otomatis diaktifkan pada kedua kapasitas baca dan tulis. Jika Anda ingin mengubah pengaturan ini, hapus Mewarisi pengaturan kapasitas dari tabel dasar di bawah Pengaturan kapasitas.

Pilih mode kapasitas, kapasitas baca dan tulis, serta IAM role auto scaling (jika ada) yang Anda inginkan.

Untuk informasi selengkapnya tentang pengaturan kapasitas DynamoDB, lihat [Mode kapasitas baca/tulis](#).

- Simpan hasil edit ke pengaturan tabel Anda..

Cancel

Save edits

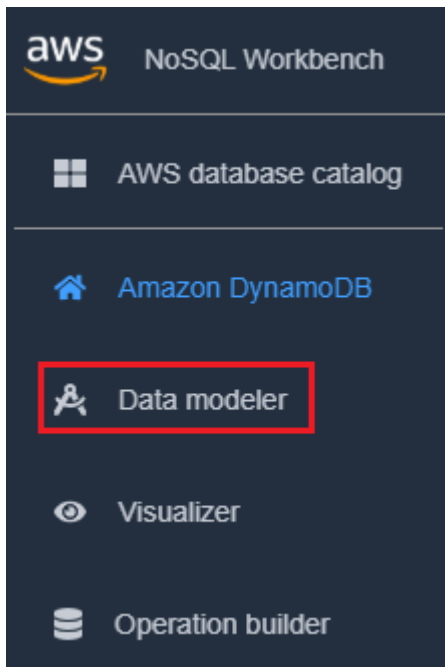
Untuk informasi selengkapnya tentang operasi CreateTable API, lihat [CreateTable](#) di Referensi Amazon DynamoDB API.

Mengimpor model data yang ada

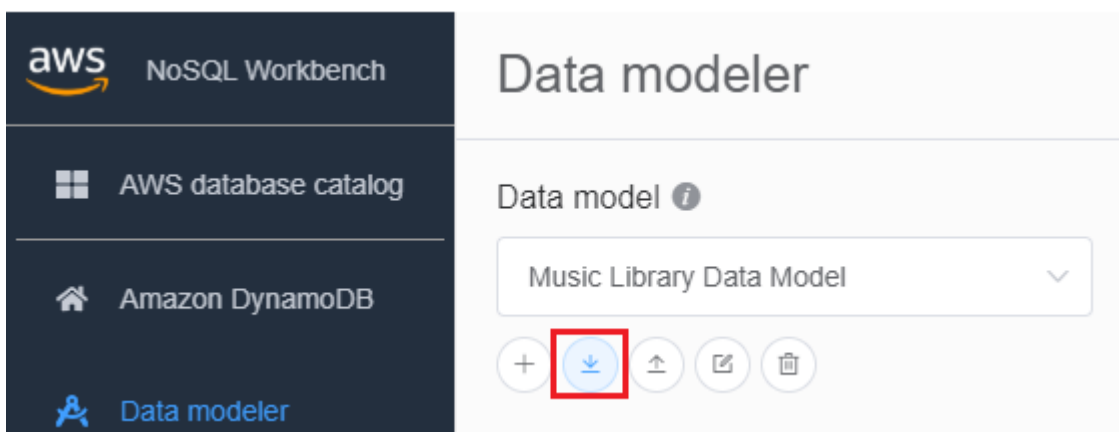
Anda dapat menggunakan NoSQL Workbench untuk Amazon DynamoDB untuk membangun model data dengan mengimpor dan memodifikasi model yang ada. Anda dapat mengimpor model data baik dalam format model NoSQL Workbench atau dalam [AWS CloudFormation Format templat JSON](#).

Untuk mengimpor model data

1. Di NoSQL Workbench, di panel navigasi di sisi kiri, pilih ikon Pemodel data.

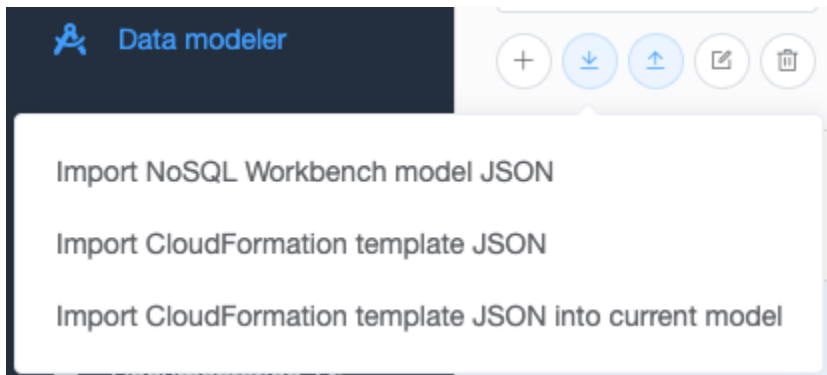


2. Arahkan kursor ke atas model data Impor.

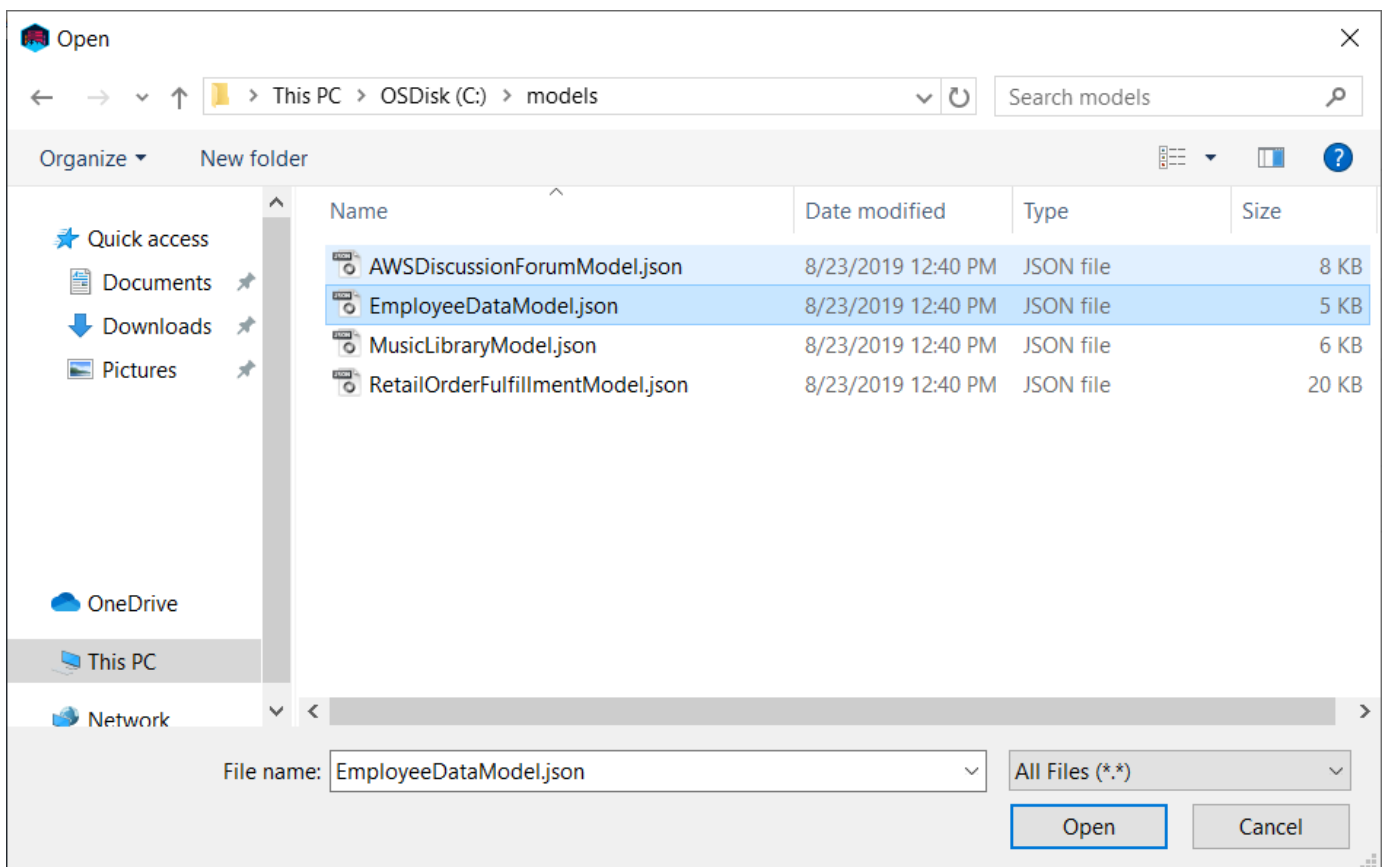


Dalam daftar dropdown, pilih apakah model yang ingin Anda impor dalam format model NoSQL Workbench atau format template JSON. CloudFormation Jika Anda memiliki model data yang

ada terbuka di NoSQL Workbench, Anda akan memiliki opsi untuk mengimpor CloudFormation template ke model saat ini.




3. Pilih model untuk diimpor.



4. Jika model yang Anda impor dalam format CloudFormation templat, Anda akan melihat daftar tabel yang akan diimpor dan memiliki kesempatan untuk menentukan nama model data, penulis, dan deskripsi.

Create data model for Amazon DynamoDB

 Only CloudFormation resources related to DynamoDB: tables and any related application auto scaling, will be imported. Some fields within these resources are not supported by NoSQL Workbench and will also not be imported, including LocalSecondaryIndexes, RoleARN, and PolicyName.

Successfully imported tables (1)

 Employee

Data model information

* Name

Author

Description

Cancel

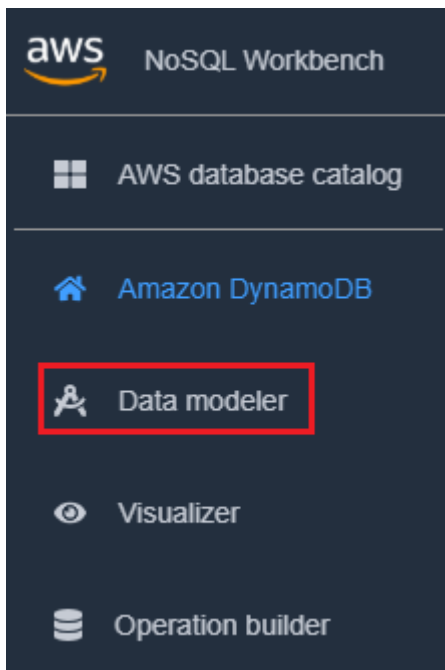
Create

Mengekspor model data

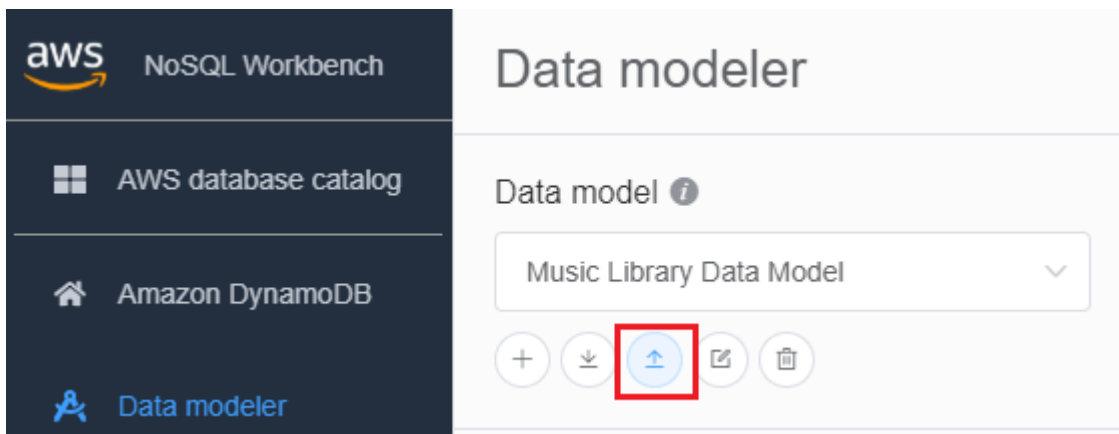
Setelah Anda membuat model data menggunakan NoSQL Workbench untuk Amazon DynamoDB, Anda dapat menyimpan dan mengekspor model dalam format model NoSQL Workbench atau [AWS CloudFormation Format templat JSON](#).

Untuk mengekspor model data

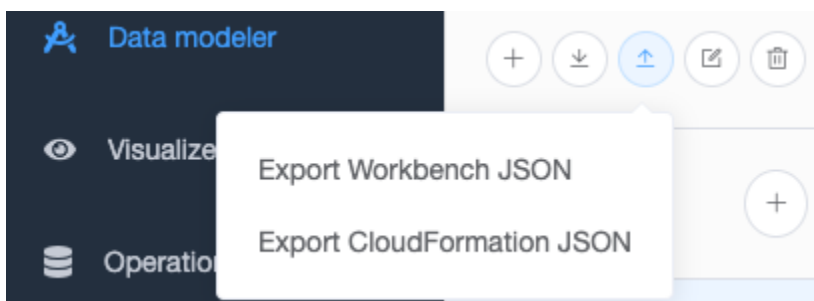
1. Di NoSQL Workbench, di panel navigasi di sisi kiri, pilih ikon Pemodel data.



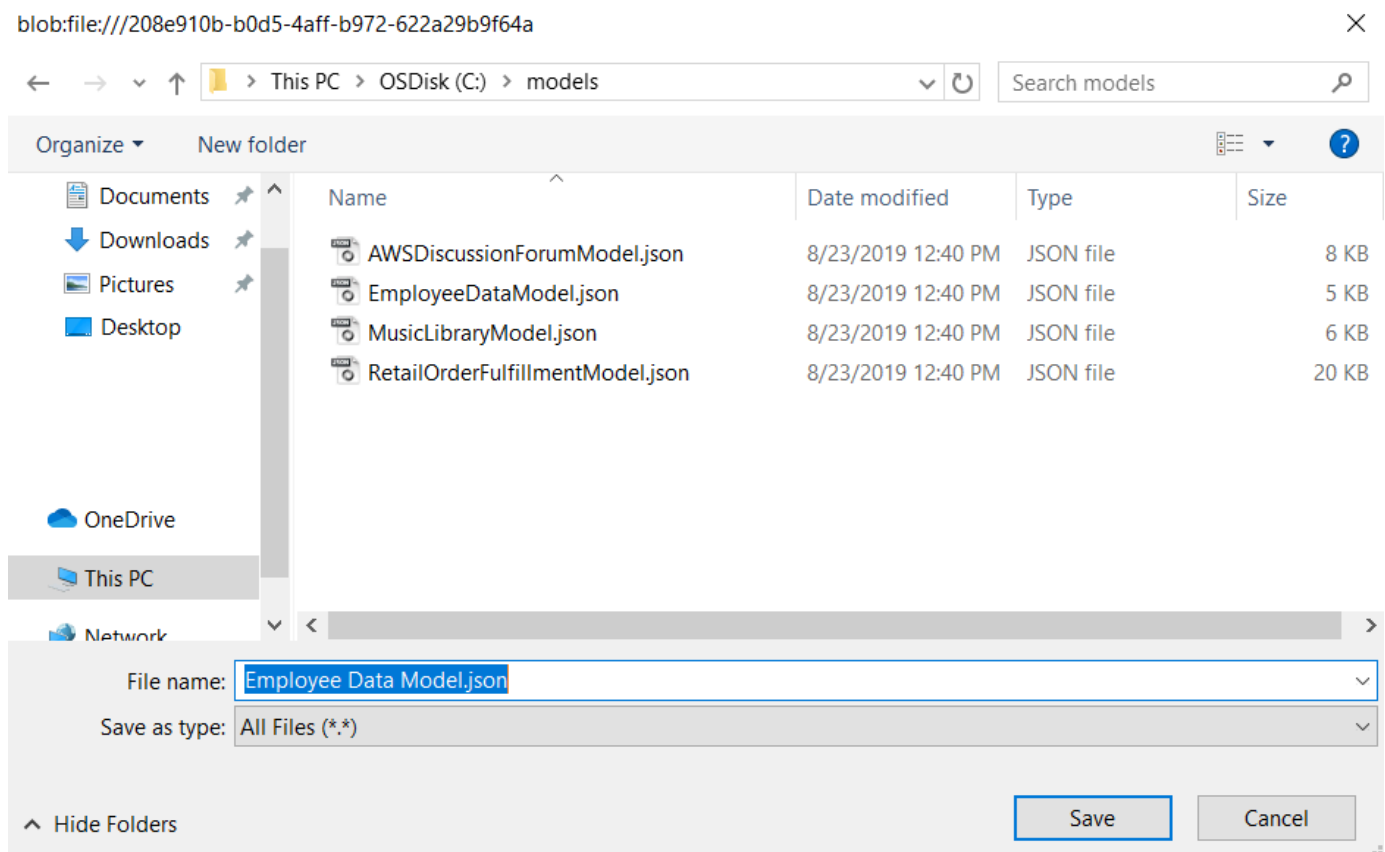
2. Arahkan kursor ke atas model data Ekspor.



Dalam daftar dropdown, pilih apakah akan mengekspor model data Anda dalam format model NoSQL Workbench atau format template JSON. CloudFormation



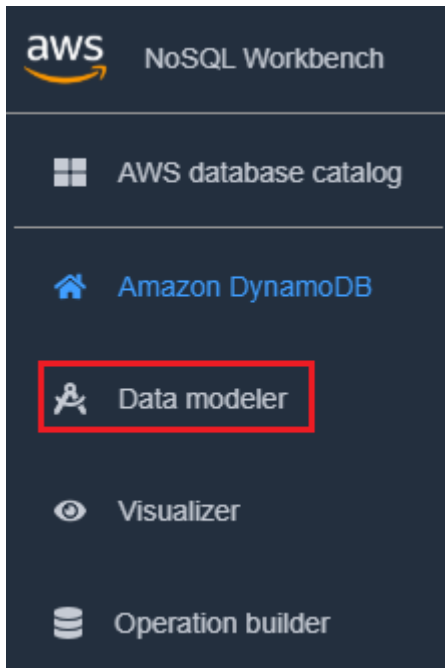
3. Pilih lokasi untuk menyimpan model Anda.



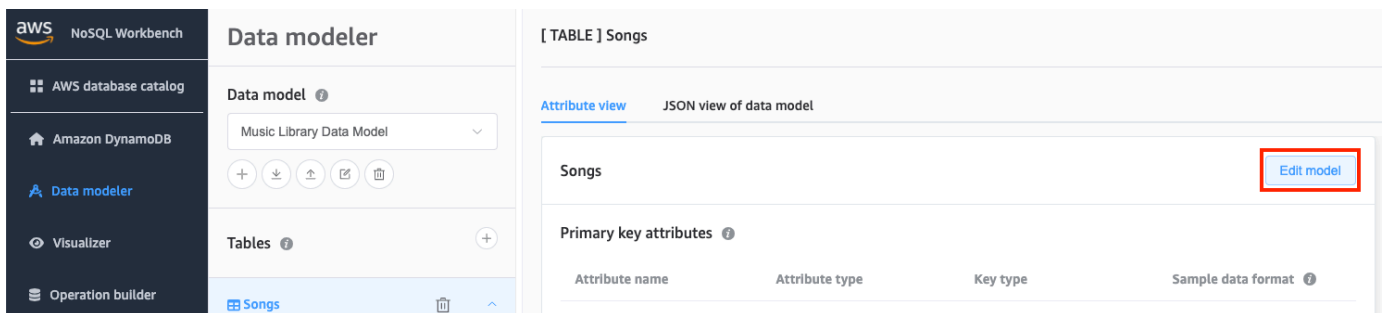
Mengedit model data yang ada

Untuk mengedit model yang ada

1. Di NoSQL Workbench, di panel navigasi di sisi kiri, pilih tombol Pemodel data.



- Pilih model data dan pilih tabel yang ingin Anda edit. Pilih Edit model



- Lakukan pengeditan yang diperlukan, lalu pilih Simpan pengeditan.

Untuk mengedit model yang sudah ada dan menambahkan faset secara manual

- Ekspor model Anda. Untuk informasi selengkapnya, lihat [Mengekspor model data](#).
- Buka file yang diekspor dalam editor.
- Temukan `DataModel` Objek untuk tabel yang ingin Anda buat faset.

Tambahkan susunan `TableFacets` yang mewakili semua faset untuk tabel.

Untuk setiap segi, tambahkan objek ke `TableFacets` array. Setiap elemen susunan memiliki sifat berikut:

- `FacetName`— Nama untuk sisi Anda. Nilai ini harus unik di seluruh model.

- **PartitionKeyAlias**— Nama yang ramah untuk kunci partisi tabel. Alias ini ditampilkan ketika Anda melihat faset di NoSQL Workbench.
- **SortKeyAlias**— Nama yang ramah untuk kunci pengurutan tabel. Alias ini ditampilkan ketika Anda melihat faset di NoSQL Workbench. Properti ini tidak diperlukan jika tabel tidak memiliki kunci pengurutan yang ditentukan.
- **NonKeyAttributes**— Sebuah array nama atribut yang diperlukan untuk pola akses. Nama-nama ini harus dipetakan ke nama atribut yang didefinisikan untuk tabel.

```
{
  "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. Anda sekarang dapat mengimpor model yang dimodifikasi ke NoSQL Workbench. Untuk informasi selengkapnya, lihat [Mengimpor model data yang ada](#).

Memvisualisasikan pola akses data

Anda dapat menggunakan alat pemvisualisasi di NoSQL Workbench untuk Amazon DynamoDB untuk memetakan kueri dan memvisualisasikan pola akses yang berbeda (dikenal sebagai faset) dari aplikasi. Setiap faset sesuai dengan pola akses yang berbeda di DynamoDB. Anda juga dapat secara manual menambahkan data ke model data Anda atau mengimpor data dari MySQL.

Topik

- [Menambahkan data sampel ke model data](#)
- [Mengimpor data sampel dari file CSV](#)
- [Melihat pola akses data](#)
- [Melihat semua tabel dalam model data menggunakan tampilan agregat](#)
- [Melakukan model data untuk DynamoDB](#)

Menambahkan data sampel ke model data

Dengan menambahkan data sampel ke model Anda, Anda dapat menampilkan data ketika memvisualisasikan model dan berbagai pola akses datanya, atau faset.

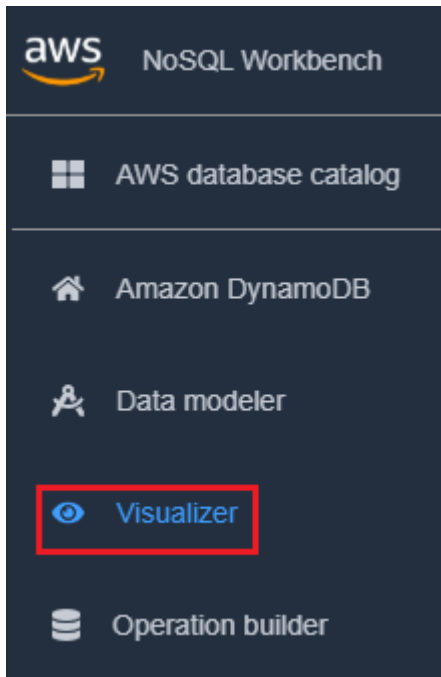
Ada dua cara untuk menambahkan data. Salah satunya adalah menggunakan alat pembuatan auto data sampel kami. Yang lainnya adalah menambahkan data satu per satu.

Ikuti langkah-langkah ini untuk menambahkan data sampel ke model data menggunakan NoSQL Workbench untuk Amazon DynamoDB.

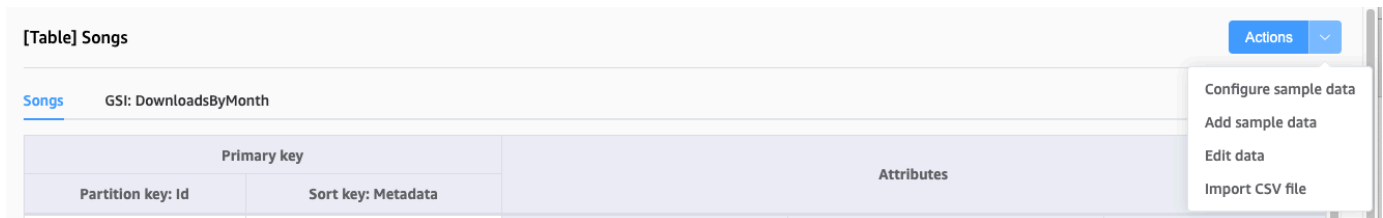
Untuk menghasilkan data sampel secara otomatis

Data sampel yang dihasilkan secara otomatis membantu Anda menghasilkan antara 1 hingga 5000 baris data untuk digunakan segera. Anda dapat menentukan tipe data sampel granular untuk membuat data realistis berdasarkan kebutuhan desain dan pengujian Anda. Untuk memanfaatkan kemampuan menghasilkan data realistis, Anda perlu menentukan format tipe data sampel untuk atribut Anda di pemodel Data. Lihat [Membuat model data baru](#) untuk menentukan format tipe data sampel.

1. Dalam panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



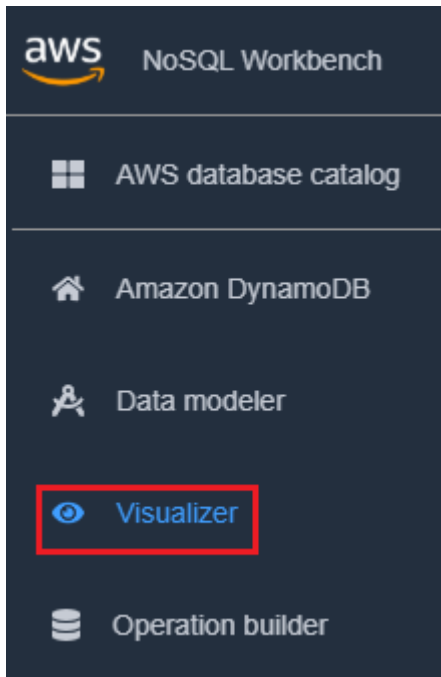
2. Di visualizer, pilih model data dan pilih tabel.
3. Pilih Action drop down, dan pilih Tambahkan data sampel.



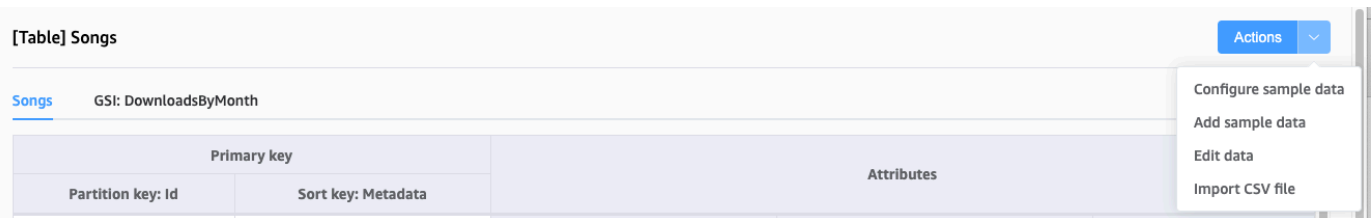
4. Masukkan nomor atau item data sampel yang ingin Anda hasilkan, lalu pilih Konfirmasi.

Untuk menambahkan data sampel satu per satu

1. Dalam panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



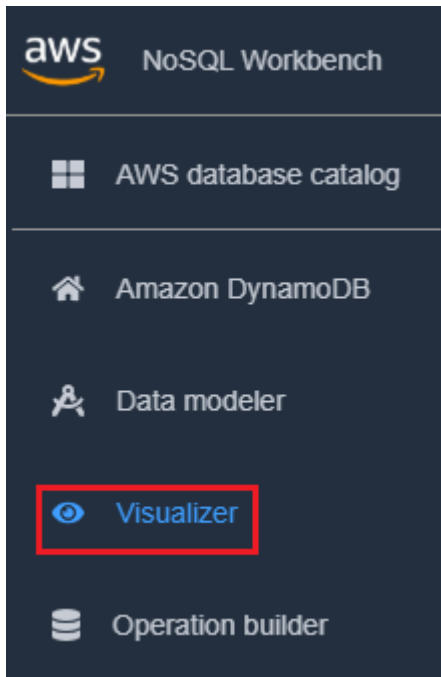
2. Di visualizer, pilih model data dan pilih tabel.
3. Pilih Action drop down, dan pilih Edit data.



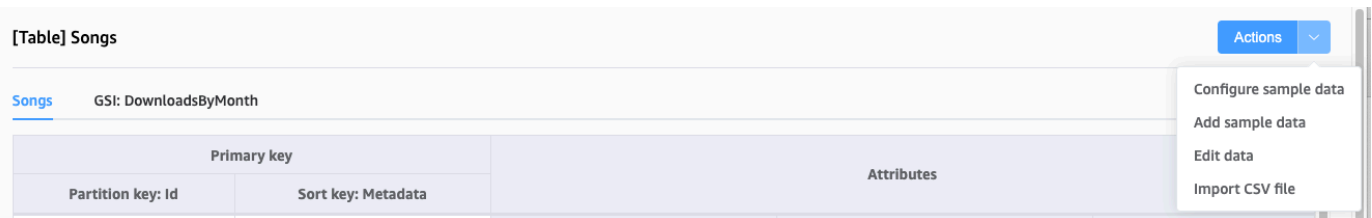
4. Pilih Tambahkan baris baru. Masukkan data sampel ke dalam kotak teks kosong, lalu pilih Tambah baris baru lagi untuk menambahkan baris tambahan. Setelah selesai pilih Simpan perubahan.

Untuk menghapus data

1. Dalam panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



2. Di visualizer, pilih model data dan pilih tabel.
3. Pilih Action drop down, dan pilih Edit data.



4. Pilih ikon hapus di samping setiap baris data yang ingin Anda hapus.

Mengimpor data sampel dari file CSV

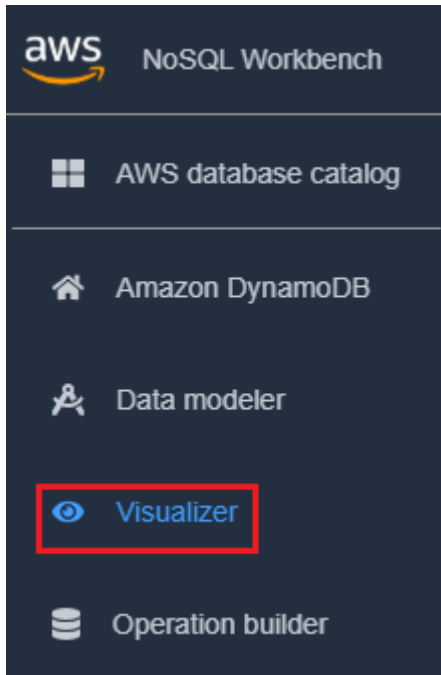
Jika Anda memiliki data sampel yang sudah ada sebelumnya dalam file CSV, Anda dapat mengimpornya ke NoSQL Workbench. Ini memungkinkan Anda untuk dengan cepat mengisi model Anda dengan data sampel tanpa harus memasukkannya baris demi baris.

Nama kolom dalam file CSV harus cocok dengan nama atribut dalam model data Anda, tetapi tidak harus dalam urutan yang sama. Misalnya, jika model data Anda memiliki atribut yang disebut `LoginAlias`, `FirstName`, dan `LastName`, kolom CSV Anda dapat berupa `LastName`, `FirstName`, dan `LoginAlias`.

Impor data dari file CSV dibatasi hingga 150 baris sekaligus.

Untuk mengimpor data dari file CSV ke NoSQL Workbench

1. Dalam panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



2. Di visualizer, pilih model data dan pilih tabel.
3. Pilih Action drop down, dan pilih Edit Data.
4. Pilih Action drop down lagi, dan pilih Impor file CSV.
5. Pilih file CSV Anda dan pilih Buka. Data dalam file CSV akan ditambahkan ke tabel Anda.

Note

Jika file CSV Anda berisi satu atau lebih baris yang memiliki kunci yang sama dengan item yang sudah ada di tabel Anda, Anda akan memiliki opsi untuk menimpa item yang ada atau menambahkannya ke akhir tabel. Jika Anda memilih untuk menambahkan item, akhiran “-Copy” akan ditambahkan ke setiap kunci item duplikat untuk membedakan item duplikat dari item yang sudah ada di tabel.

Melihat pola akses data

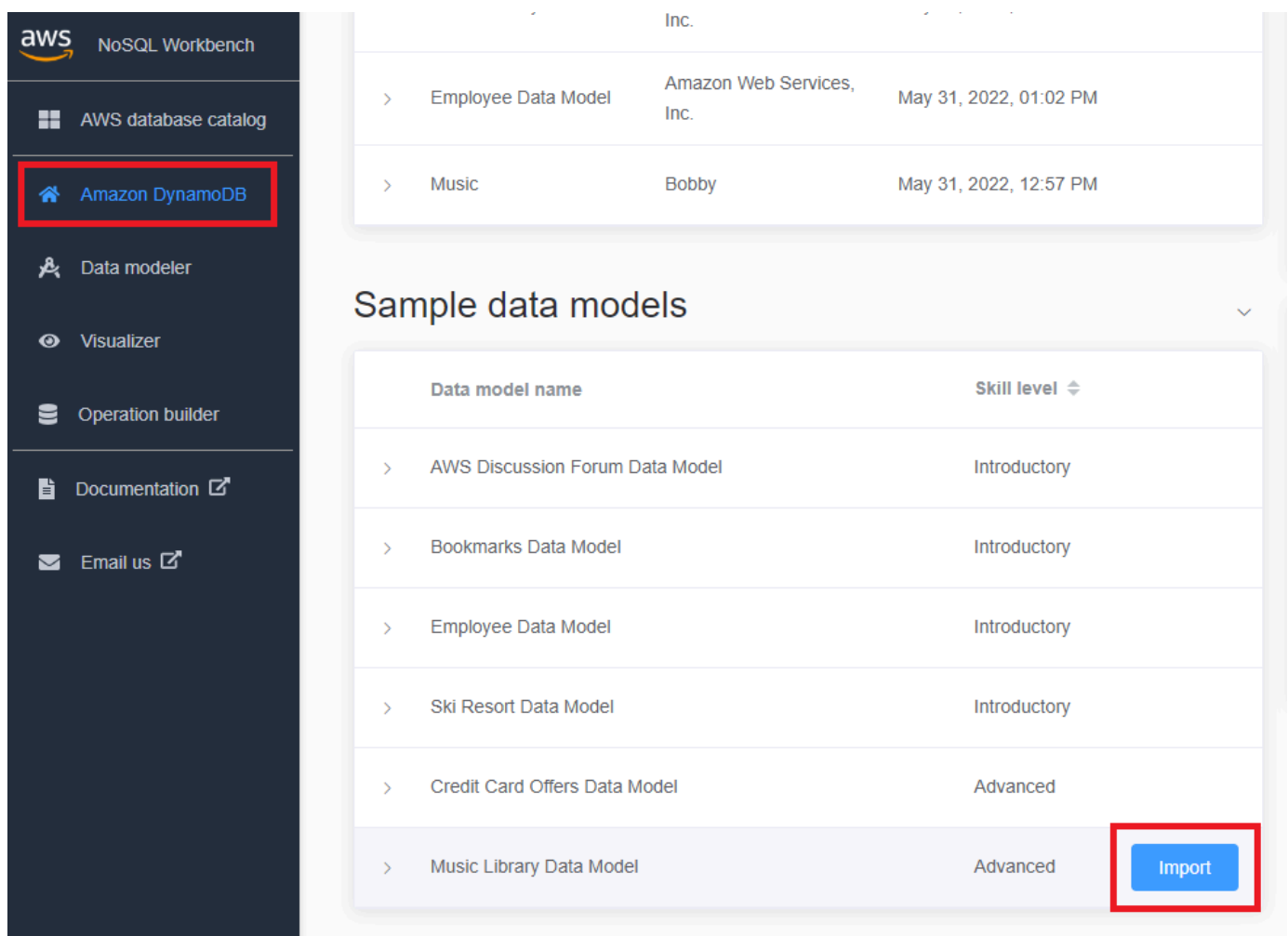
Di NoSQL Workbench, faset mewakili pola akses data aplikasi yang berbeda untuk Amazon DynamoDB. Aspek dapat membantu Anda memvisualisasikan model data Anda ketika beberapa tipe data diwakili oleh kunci pengurutan. Aspek memberi Anda cara untuk melihat subset data dalam

tabel, tanpa harus melihat catatan yang tidak memenuhi batasan aspek. Aspek dianggap sebagai alat pemodelan data visual, dan tidak ada sebagai konstruksi yang dapat digunakan di DynamoDB, karena mereka murni bantuan untuk pemodelan pola akses.

Untuk melihat contoh aspek, Anda dapat mengimpor salah satu model data sampel kami dengan faset sebagai bagian dari templat model data.

Model data impor

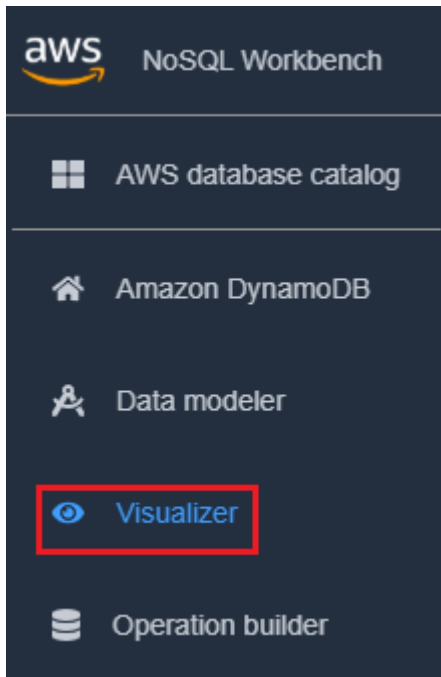
1. Di sebelah kiri, pilih Amazon DynamoDB.
2. Di bagian Model data sampel, arahkan kursor ke Model Data Perpustakaan Musik dan pilih Impor.



The screenshot shows the AWS NoSQL Workbench interface. On the left is a dark navigation sidebar with the following items: 'AWS database catalog', 'Amazon DynamoDB' (highlighted with a red box), 'Data modeler', 'Visualizer', 'Operation builder', 'Documentation', and 'Email us'. The main content area displays a table of sample data models. The table has two columns: 'Data model name' and 'Skill level'. The 'Music Library Data Model' is highlighted in light blue, and its 'Import' button is also highlighted with a red box.

Data model name	Skill level
> Employee Data Model	Amazon Web Services, Inc. May 31, 2022, 01:02 PM
> Music	Bobby May 31, 2022, 12:57 PM
> AWS Discussion Forum Data Model	Introductory
> Bookmarks Data Model	Introductory
> Employee Data Model	Introductory
> Ski Resort Data Model	Introductory
> Credit Card Offers Data Model	Advanced
> Music Library Data Model	Advanced

3. Dalam panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



4. Pilih tabel Lagu untuk memperluasnya. Anda akan ditampilkan tampilan agregat data Anda.

Primary key		Attributes		
Partition key: Id	Sort key: Metadata	Title	Artist	TotalDownloads
	Details	Wild Love	Argyboots	3
Did-9349823681	DownloadTimestamp	2018-01-01T00:00:07		
Did-9349823682	DownloadTimestamp	2018-01-01T00:01:08		

5. Pilih panah drop-down Facets untuk memperluas aspek yang tersedia.
6. Pilih SongDetails faset untuk memvisualisasikan data dengan SongDetails faset yang diterapkan.

SongId (Partition key) : String	Metadata (Sort key) : String	Title : String	Artist : String	TotalDownloads : String
1	Details	Wild Love	Argyboots	3
2	Details	Example Song Title	Jorge Souza	4
12	ACME Album	ACME Best Song	ACME	4

Anda juga dapat mengedit definisi faset menggunakan Pemodel Data. Untuk informasi selengkapnya, lihat [Mengedit model data yang ada](#).

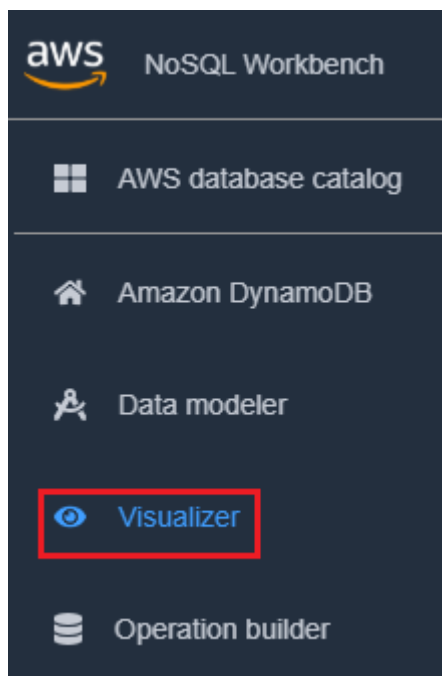
Melihat semua tabel dalam model data menggunakan tampilan agregat

Tampilan agregat di NoSQL Workbench untuk Amazon DynamoDB mewakili semua tabel dalam model data. Untuk setiap tabel, informasi berikut muncul:

- Nama kolom tabel
- Data sampel
- Semua indeks sekunder global yang terkait dengan tabel. Informasi berikut ini ditampilkan untuk masing-masing indeks:
 - Nama kolom indeks
 - Data sampel

Untuk melihat semua informasi tabel

1. Dalam panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



2. Dalam pemvisualisasi, pilih Tampilan agregat.

The screenshot shows the AWS NoSQL Workbench Visualizer interface. On the left is a navigation sidebar with options like 'AWS database catalog', 'Amazon DynamoDB', 'Data modeler', 'Visualizer', 'Operation builder', 'Documentation', and 'Share feedback'. The main area is titled 'Visualizer' and shows a 'Data model' for 'Discussion Forum'. Below this, there are sections for 'Forum', 'Thread', and 'Reply'. An 'Aggregate view' button is visible. The 'Aggregate view' section displays a table with the following data:

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	
Amazon MQ	Category	Threads	Messages	Views	
	Amazon Web Services	22	7	1400	
Amazon EMR	Category	Threads	Messages	Views	
	Amazon Web Services	15	8	600	
AWS Data Pipeline	Category	Threads	Messages	Views	
	Amazon Web Services	19	9	500	
Amazon Athena	Category	Threads	Messages	Views	
	Amazon Web Services	43	10	55	
AWS Step Functions	Category	Threads	Messages	Views	
	Amazon Web Services	30	11	99	

Melakukan model data untuk DynamoDB

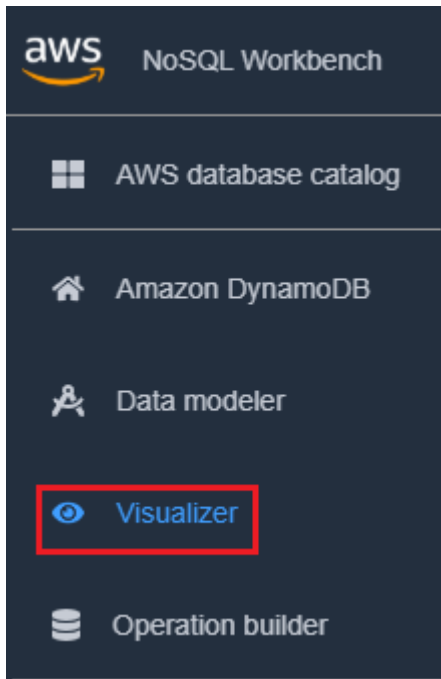
Bila Anda puas dengan model data Anda, Anda dapat melakukan model pada Amazon DynamoDB.

Note

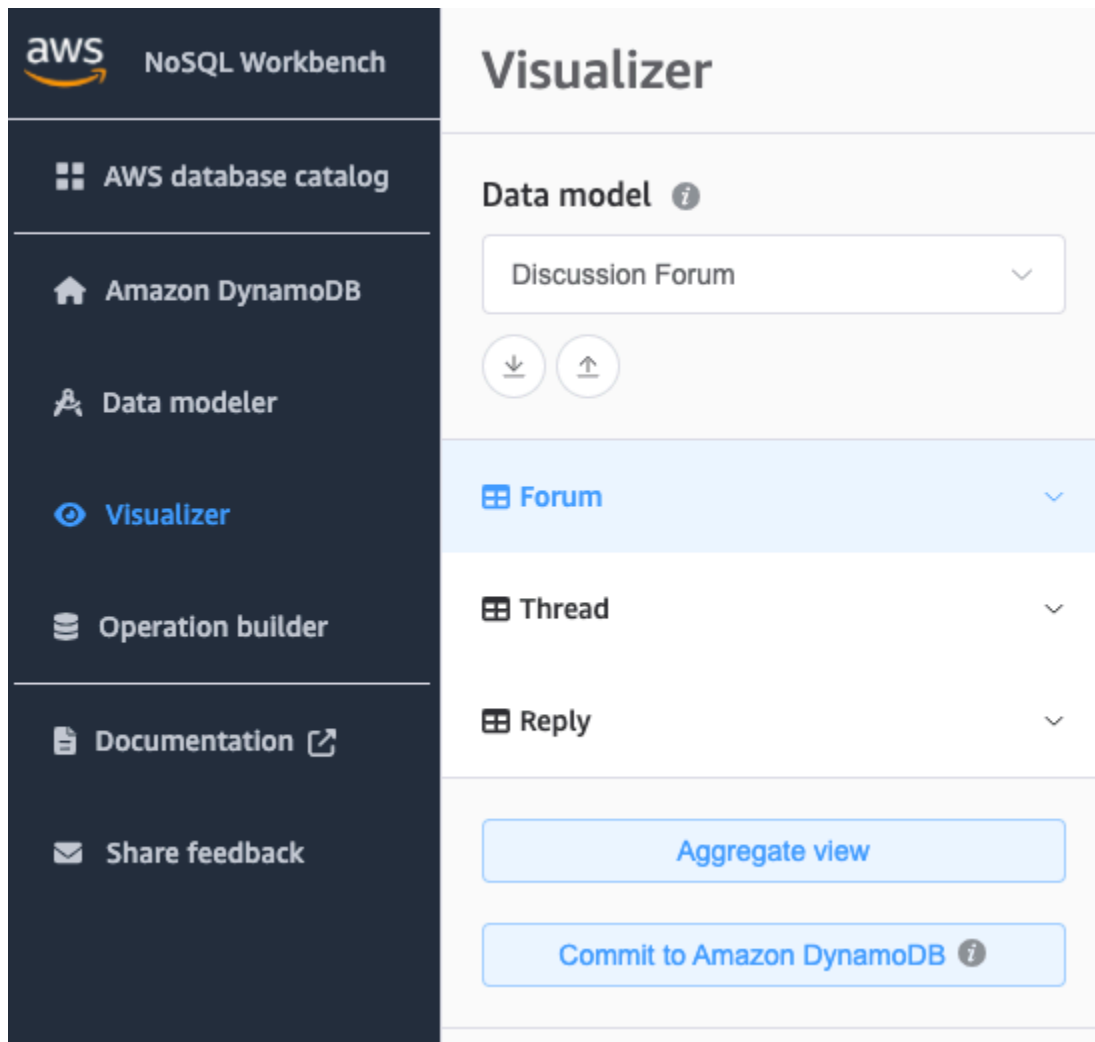
- Tindakan ini menghasilkan pembuatan sumber daya sisi server di AWS untuk tabel dan indeks sekunder global yang diwakilkan dalam model data.
- Tabel dibuat dengan karakteristik sebagai berikut:
 - Auto scaling diatur ke 70 persen pemanfaatan target.
 - Kapasitas yang disediakan diatur ke 5 unit kapasitas baca dan 5 unit kapasitas tulis.
- Indeks sekunder global dibuat dengan kapasitas yang disediakan dari 10 unit kapasitas baca dan 5 unit kapasitas tulis.

Untuk melakukan model data pada DynamoDB

1. Dalam panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



2. Pilih Melakukan pada DynamoDB.



3. Pilih koneksi yang sudah ada, atau buat koneksi baru dengan memilih tab Menambahkan koneksi jarak jauh baru.

- Untuk menambahkan koneksi baru, tentukan informasi berikut:
 - Akun Alias
 - AWSWilayah
 - Access key
 - Secret access key

Untuk informasi selengkapnya tentang cara mendapatkan access key, lihat [Mendapatkan AWS access key](#).

- Anda juga dapat menentukan hal berikut:
 - [Token sesi](#)

- [Peran IAM ARN](#)
- Jika Anda tidak ingin mendaftar untuk akun tingkat gratis, dan memilih untuk menggunakan [DynamoDB lokal \(versi dapat diunduh\)](#):
 1. Pilih tab Menambahkan koneksi lokal DynamoDB baru.
 2. Tentukan Nama koneksi dan Port.
- 4. Pilih Lakukan.

Note

Jika Anda menginstal DynamoDB lokal sebagai bagian dari pengaturan NoSQL Workbench, Anda harus mengaktifkan DynamoDB lokal dengan menggunakan sakelar Server lokal DynamoDB di kiri bawah layar NoSQL Workbench. Lihat [Menginstal NoSQL Workbench untuk DynamoDB](#) untuk informasi lebih lanjut tentang sakelar ini.

Menjelajahi set data dan membangun operasi dengan NoSQL Workbench

NoSQL Workbench for Amazon DynamoDB menyediakan antarmuka pengguna grafis yang kaya bagi Anda untuk mengembangkan dan menguji kueri. Anda dapat menggunakan pembangun operasi di NoSQL Workbench untuk melihat, menjelajahi, dan mengkueri dataset. Builder operasi terstruktur mendukung ekspresi proyeksi, ekspresi kondisi, dan menghasilkan kode sampel dalam berbagai bahasa. Anda dapat langsung mengkloning tabel dari satu akun Amazon DynamoDB ke akun lain di Wilayah yang berbeda. Anda juga dapat langsung mengkloning tabel antara DynamoDB lokal dan akun Amazon DynamoDB untuk menyalin skema kunci tabel Anda dengan lebih cepat (dan skema dan item GSI opsional) di antara lingkungan pengembangan Anda. Anda dapat menyimpan sebanyak 50 operasi data DynamoDB di pembuat operasi.

Topik

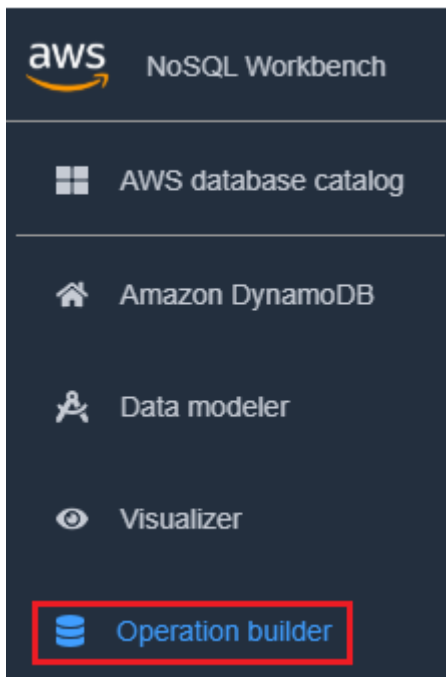
- [Menghubungkan ke kumpulan data langsung](#)
- [Membangun operasi kompleks](#)
- [Kloning tabel dengan NoSQL Workbench](#)
- [Mengekspor data ke file CSV](#)

Menghubungkan ke kumpulan data langsung

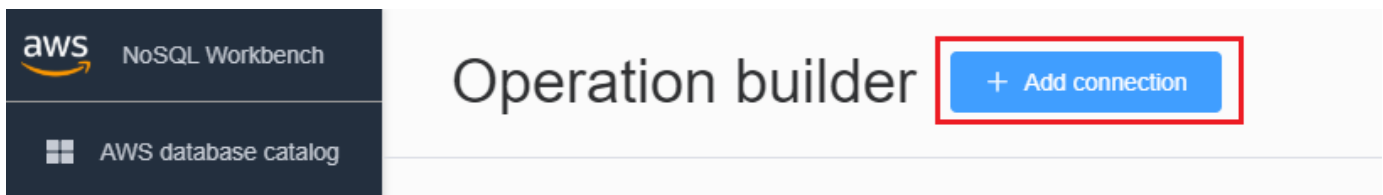
Untuk terhubung ke tabel Amazon DynamoDB Anda dengan NoSQL Workbench, Anda harus terlebih dahulu terhubung ke akun Anda. AWS

Untuk menambahkan koneksi ke database Anda

1. Di NoSQL Workbench, di panel navigasi di sisi kiri, pilih ikon Pembangun operasi.



2. Pilih Tambahkan koneksi.



3. Tentukan informasi berikut:

- Alias akun
- AWS Wilayah
- ID kunci akses
- Kunci akses rahasia

Untuk informasi selengkapnya tentang cara mendapatkan kunci akses, lihat [Mendapatkan kunci AWS akses](#).

Anda dapat menentukan hal berikut secara opsional:

- [Token sesi](#)
- ARN peran IAM

4. Pilih Hubungkan.

Add a new database connection



A remote connection lets you access the DynamoDB web service in different AWS Regions.

A DynamoDB local connection lets you access the DynamoDB local server running on your computer.

Remote

DynamoDB local

* Connection name

Connection 3

* Default AWS Region

Default AWS Region



* Access key ID

AWS Access key ID

* Secret access key

AWS secret access key

Session token

AWS session token



IAM role ARN

IAM role ARN



Persist connection

If you select this check box, AWS connection secrets will be persisted in

C:\Users\mon...m\aws\credentials

Cancel

Reset

Connect

Jika Anda tidak ingin mendaftar akun tingkat gratis, dan lebih memilih menggunakan [DynamoDB lokal \(versi yang dapat diunduh\)](#):

- a. Pilih tab Lokal pada layar koneksi
- b. Tentukan informasi berikut:
 - Nama koneksi
 - Port
- c. Pilih tombol hubungkan.

Add a new database connection



A remote connection can let you interact with DynamoDB servers in different regions.
A local connection can let you interact with the DynamoDB Local server on your machine.

Remote

Local



Please setup the dynamoDB local server on your machine before adding a connection.

[Setting Up DynamoDB Local](#)

* Connection name

Connection 1

* Hostname

localhost

* Port

8000

Cancel

Reset

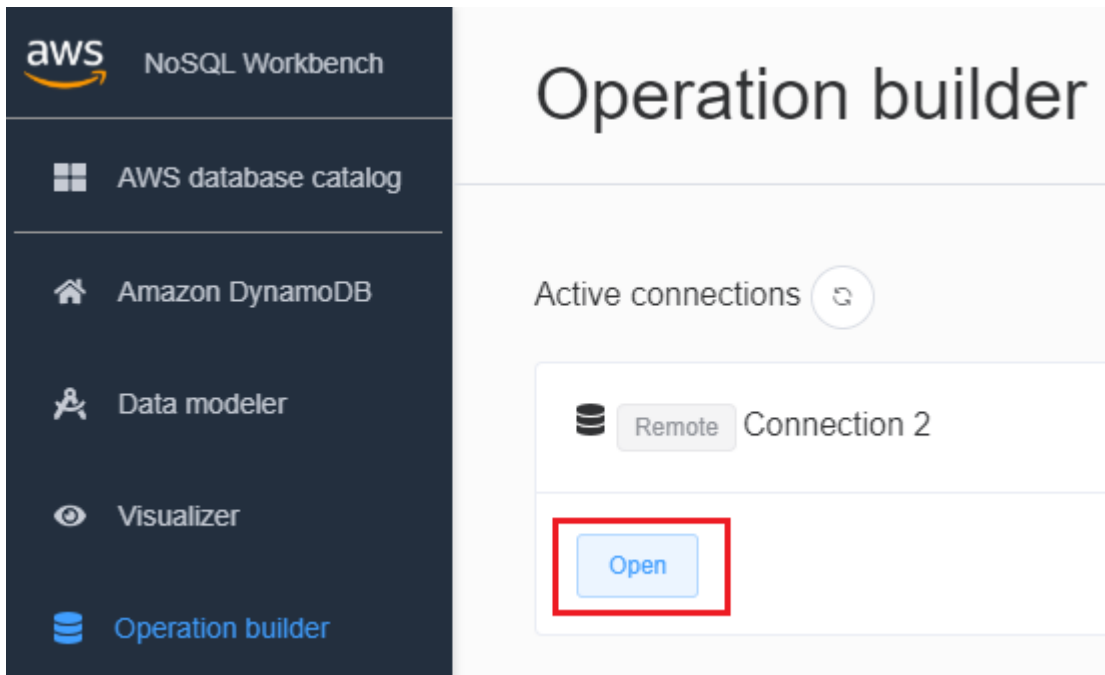
Connect

Note

Untuk terhubung ke DynamoDB lokal, jalankan DynamoDB lokal secara manual menggunakan terminal Anda (lihat [menerapkan DynamoDB lokal di komputer Anda](#)) atau [luncurkan DynamoDB lokal secara langsung menggunakan sakelar lokal DDB di menu](#)

navigasi NoSQL Workbench. Pastikan port koneksi sama dengan port lokal DynamoDB Anda.

5. Pada koneksi yang dibuat, pilih Buka.



Setelah menghubungkan ke database DynamoDB Anda, daftar tabel yang tersedia muncul di panel kiri. Pilih salah satu tabel untuk mengembalikan sampel data yang tersimpan dalam tabel.

Anda sekarang dapat menjalankan kueri terhadap tabel yang dipilih.

Untuk menjalankan kueri pada tabel

1. Di daftar Nama atribut, pilih atribut yang ingin Anda kueri.
2. Tentukan operator perbandingan.
3. Tentukan tipe daya dari nilai.
4. Tentukan nilai untuk dilakukan kueri.
5. Pilih Pindai.

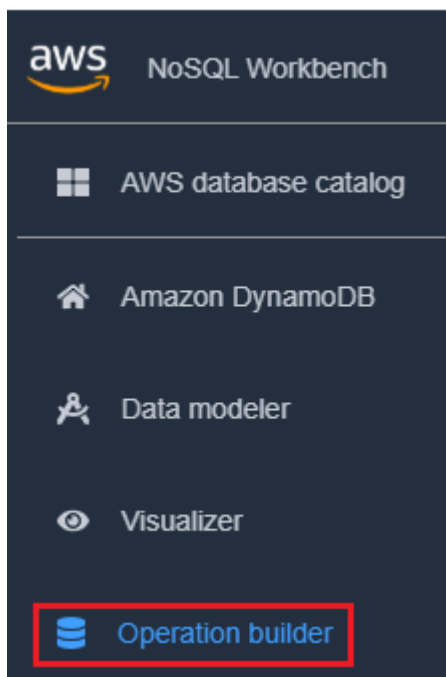
Untuk informasi selengkapnya tentang operasi ini, lihat [Pindai](#) dalam Referensi API Amazon DynamoDB.

Membangun operasi kompleks

Pembangun operasi di NoSQL Workbench untuk Amazon DynamoDB menyediakan antarmuka visual di mana Anda dapat melakukan operasi bidang data yang kompleks. Ini termasuk dukungan untuk ekspresi proyeksi dan ekspresi syarat. Setelah Anda membangun operasi, Anda dapat menyimpannya untuk digunakan nanti (hingga 50 operasi dapat disimpan). Anda kemudian dapat menelusuri daftar operasi bidang data yang sering digunakan di menu Operasi Tersimpan, dan menggunakannya untuk mengisi dan membangun operasi baru secara otomatis. Anda juga dapat menghasilkan kode sampel untuk operasi ini, dalam beberapa bahasa.

NoSQL Workbench mendukung pembangunan [PartiQL](#) untuk pernyataan DynamoDB, yang memungkinkan Anda untuk berinteraksi dengan DynamoDB menggunakan bahasa kueri yang kompatibel dengan SQL. NoSQL Workbench juga mendukung membangun operasi API CRUD DynamoDB.

Untuk menggunakan NoSQL Workbench untuk membangun operasi, di panel navigasi di sisi kiri, pilih ikon Pembangun operasi.



Topik

- [Membangun pernyataan PartiQL](#)
- [Membangun operasi API](#)

Membangun pernyataan PartiQL

Untuk menggunakan NoSQL Workbench untuk membangun pernyataan [PartiQL untuk DynamoDB](#) pilih operasi PartiQL di sudut kanan atas NoSQL Workbench.

Anda dapat membangun jenis pernyataan PartiQL berikut dalam pembangun operasi.

Topik

- [Pernyataan singleton](#)
- [Transaksi](#)
- [Batch](#)

Pernyataan singleton

Untuk menjalankan atau menghasilkan kode untuk pernyataan PartiQL, lakukan hal berikut.

PartiQL statement PartiQL transaction PartiQL batch

1

Statement

```
1 SELECT *
2 FROM Music
3 WHERE Artist=? and SongTitle=?
```

2

Optional request parameters **3.a**

Enable strongly consistent reads ⓘ

Parameters ⓘ

Attribute type	Attribute value 3.c
String	Acme Band
Attribute type	Attribute value
String	PartiQL Rocks

3.b + Add new parameter

5 **4** **6**

Clear form Run Generate code Save operation

▲ Hide operation

1. Pilih pernyataan PartiQL.
2. Masukkan [pernyataan PartiQL](#) yang valid.
3. Jika pernyataan Anda menggunakan parameter:
 - a. Pilih Parameter permintaan opsional.
 - b. Pilih Tambahkan parameter baru.
 - c. Masukkan jenis dan nilai atribut.
 - d. Jika Anda ingin menambahkan parameter tambahan, ulangi langkah b dan c.
4. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih bahasa yang Anda inginkan dari tab yang ditampilkan. Anda sekarang dapat menyalin kode ini dan menggunakannya di aplikasi Anda.

5. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
6. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi. Kemudian masukkan nama untuk operasi Anda dan pilih Simpan.

Transaksi

Untuk menjalankan atau menghasilkan kode untuk transaksi PartiQL, lakukan hal berikut.

PartiQL statement **PartiQL transaction** PartiQL batch

1

Delete all statements Collapse all statements Run Generate code Save operation

7 6 8

i Read and write operations are not supported in the same PartiQL transaction request. A SELECT statement cannot be in the same request with INSERT, UPDATE, and DELETE statements.

+ 1. Choose and then enter a title

Statement

```
1 SELECT * FROM Music WHERE Artist=? AND SongTitle=?
```

3

Optional request parameters 4.a

Parameters **i**

Attribute type	Attribute value
String	Acme Band
String	PartiQL Rocks

+ Add new parameter 4.b

Clear Delete

+ Add a new statement 2

Hide operation

1. Pilih Transaksi PartiQL.
2. Pilih Tambahkan pernyataan baru.
3. Masukkan [pernyataan PartiQL](#) yang valid.

 Note

Operasi baca dan tulis tidak didukung dalam permintaan transaksi PartiQL yang sama. Pernyataan PILIH tidak dapat berada dalam permintaan yang sama dengan pernyataan SISIPKAN, PERBARUI, dan HAPUS. Lihat [Melakukan transaksi dengan PartiQL untuk DynamoDB](#) untuk detail selengkapnya.

4. Jika pernyataan Anda menggunakan parameter
 - a. Pilih Parameter permintaan opsional.
 - b. Pilih Tambahkan parameter baru.
 - c. Masukkan jenis dan nilai atribut.
 - d. Jika Anda ingin menambahkan parameter tambahan, ulangi langkah b dan c.
5. Jika Anda ingin menambahkan lebih banyak pernyataan, ulangi langkah 2 sampai 4.
6. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih bahasa yang Anda inginkan dari tab yang ditampilkan. Anda sekarang dapat menyalin kode ini dan menggunakannya di aplikasi Anda.

7. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
8. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi. Kemudian masukkan nama untuk operasi Anda dan pilih Simpan.

Batch

Untuk menjalankan atau menghasilkan kode untuk batch PartiQL, lakukan hal berikut.

PartiQL statement PartiQL transaction PartiQL batch 1

Delete all statements Collapse all statements Run Generate code Save operation

7 6 8

i Read and write operations are not supported in the same PartiQL batch request, which means a SELECT statement cannot be in the same request with INSERT, UPDATE, and DELETE statements. Write operations to the same item are not allowed. As with the BatchGetItem operation, only singleton read operations are supported. Scan and query operations are not supported.

✦ 1. Choose and then enter a title

Statement

```
1 SELECT * FROM Music WHERE Artist=? AND SongTitle=? 3
```

Optional request parameters 4.a

Enable strongly consistent reads **i**

Parameters **i**

Attribute type	Attribute value 4.c
String	Acme Band 🗑️
String	PartiQL Rocks 🗑️

+ Add new parameter 4.b


Clear Delete

+ Add a new statement 2

▲ Hide operation

1. Pilih Batch PartiQL.
2. Pilih Tambahkan pernyataan baru.

3. Masukkan [pernyataan PartiQL](#) yang valid.

 Note

Operasi baca dan tulis tidak didukung dalam permintaan batch PartiQL yang sama, yang berarti pernyataan PILIH tidak dapat berada dalam permintaan yang sama dengan pernyataan SISIPKAN, PERBARUI, dan HAPUS. Operasi tulis untuk item yang sama tidak diperbolehkan. Seperti halnya BatchGetItem operasi, hanya operasi baca tunggal yang didukung. Operasi pemindaian dan kueri tidak didukung. Lihat [Menjalankan operasi batch dengan PartiQL untuk DynamoDB](#) untuk detail selengkapnya.

4. Jika pernyataan Anda menggunakan parameter:
 - a. Pilih Parameter permintaan opsional.
 - b. Pilih Tambahkan parameter baru.
 - c. Masukkan jenis dan nilai atribut.
 - d. Jika Anda ingin menambahkan parameter tambahan, ulangi langkah b dan c.
5. Jika Anda ingin menambahkan lebih banyak pernyataan, ulangi langkah 2 sampai 4.
6. Jika Anda ingin membuat kode, pilih Buat kode.

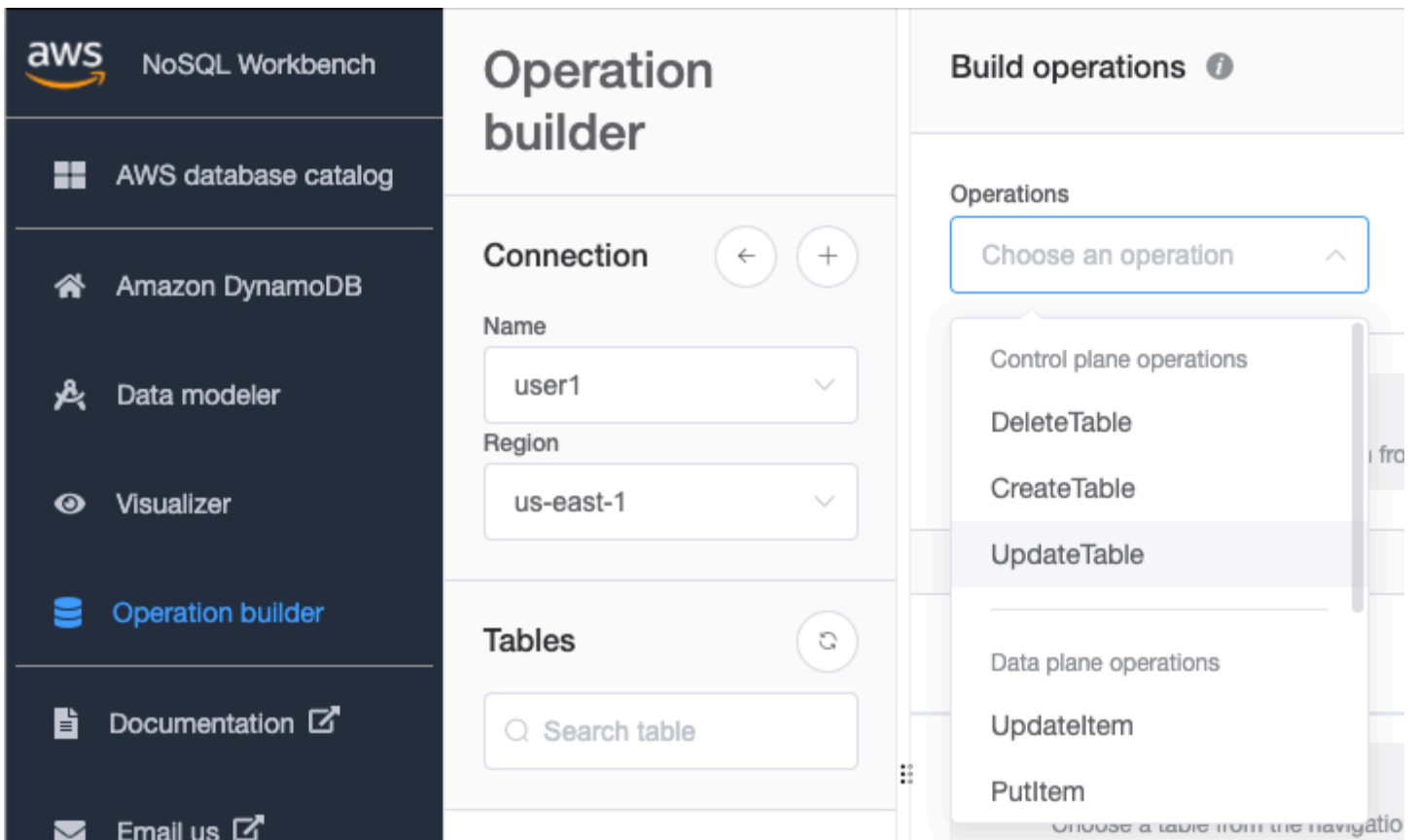
Pilih bahasa yang Anda inginkan dari tab yang ditampilkan. Anda sekarang dapat menyalin kode ini dan menggunakannya di aplikasi Anda.

7. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
8. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi. Kemudian masukkan nama untuk operasi Anda dan pilih Simpan.

Membangun operasi API

Untuk menggunakan NoSQL Workbench untuk membangun API CRUD DynamoDB, pilih Operasi berbasis antarmuka di sudut kanan atas NoSQL Workbench.

Lalu buka daftar menurun Operasi dan pilih operasi yang ingin Anda buat.



Anda dapat melakukan operasi berikut di pembangun operasi.

Operasi tabel

- [Hapus Tabel](#)
- [Buat Tabel](#)
- [Perbarui Tabel](#)

Operasi item

- [Get Item](#)
- [Put Item](#)
- [Perbarui Item](#)
- [Hapus Item](#)
- [Kueri](#)
- [Pemindaian](#)
- [Transaksi Get Item](#)

- [Transaksi Write Item](#)

Operasi tabel

Hapus tabel

Untuk menjalankan atau menghasilkan kode untuk suatu operasi Delete Table, lakukan hal berikut.

1. Pilih Hapus Tabel dari daftar menurun Operasi.
2. Pilih tabel dari daftar menurun Nama tabel.
3. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

4. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.

Di jendela konfirmasi yang muncul, konfirmasi dengan memasukkan nama tabel dan memilih Hapus.

The screenshot shows the Amazon DynamoDB console's 'Operation Builder' interface. On the left is a sidebar with 'Operation builder' title, 'Connection' (user1), 'Region' (us-east-1), and 'Tables' (Search table). The main area is 'Build operations' with 'Interface-based operations' selected. It shows 'DeleteTable' operation on 'NoSQLTest1' table. 'Run' and 'Generate code' buttons are present. Below is a 'Results' section with a message: 'No result to display. Choose a table from the navigation pane to view all table data, or build an operation and view operation results.'

Hasil operasi akan muncul di tab hasil di bagian bawah layar.

Untuk informasi selengkapnya tentang operasi ini, lihat [Menghapus tabel di Referensi API Amazon DynamoDB](#).

Membuat tabel

Untuk menjalankan atau menghasilkan kode untuk suatu operasi `Create Table`, lakukan hal berikut.

1. Pilih `Buat Tabel` dari daftar menurun `Operasi`.
2. Masukkan nama tabel yang diinginkan.
3. Buat kunci partisi.
4. Konfigurasi tabel sesuai keinginan. Anda dapat membuat `Indeks Sekunder Global`, mengaktifkan penskalaan otomatis, dan opsi lainnya.
5. Untuk menyesuaikan pengaturan kapasitas, buka `Pengaturan Kapasitas` dan hapus centang pada kotak di samping `Pengaturan default`.

- Anda sekarang dapat memilih `Kapasitas` sesuai permintaan atau `Yang disediakan`.

Dengan `Provisioned` dipilih, Anda dapat mengatur unit kapasitas baca dan tulis minimum dan maksimum. Anda juga dapat mengaktifkan atau menonaktifkan penskalaan otomatis.

- Jika tabel saat ini disetel ke `OnDemand`, Anda tidak akan dapat menentukan throughput yang disediakan.
 - Jika Anda beralih dari `OnDemand` ke `throughput Provisioned`, maka `Autoscaling` akan secara otomatis diterapkan ke semua `GSI` dengan: `min: 1, max: 10; target: 70%`.
6. Jika Anda ingin menghapus semua pengaturan baru yang Anda buat dan memulai dari awal, pilih `Hapus formulir`.
 7. Jika Anda ingin membuat kode, pilih `Buat kode`.

Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

8. Jika Anda ingin operasi segera dijalankan, pilih `Jalankan`.

Hasil operasi akan muncul di tab hasil di bagian bawah layar.

9. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih `Simpan operasi`, lalu masukkan nama untuk operasi Anda dan pilih `Simpan`.

The screenshot shows the 'Build operations' interface in the Amazon DynamoDB console. On the left is the 'Operation builder' sidebar with sections for 'Connection' (Name: user1, Region: us-east-1), 'Tables' (DeleteMe1, NoSQLTest1), and 'Saved operations' (No saved operations found). The main area is titled 'Build operations' and has two tabs: ' PartiQL operations ' and ' Interface-based operations ' (selected). Under 'Interface-based operations', the 'Operations' dropdown is set to 'CreateTable'. The 'Table name' field contains 'Music'. Under 'Primary key attributes', the 'Partition key' is 'pk1' with a data type of 'String'. There is an unchecked checkbox for 'Add sort key'. Under 'Global secondary indexes', there is a '+ Add global secondary index' button. Under 'Capacity settings', the 'Default settings' checkbox is checked, with a note: 'The default settings use provisioned capacity mode with auto scaling enabled on both read and write capacity.' At the bottom are buttons for 'Clear form', 'Run', 'Generate code', and 'Save operation'.

Untuk informasi selengkapnya tentang operasi ini, lihat [Membuat tabel](#) dalam Referensi API Amazon DynamoDB.

Perbarui Tabel

Untuk menjalankan atau menghasilkan kode untuk suatu operasi `Update Table`, lakukan hal berikut.

1. Pilih Perbarui Tabel dari daftar menurun Operasi.
2. Pilih operasi yang diinginkan dari daftar menurun Perbarui operasi.
3. Pilih tabel dari daftar menurun Nama tabel.

4. Perbarui tabel yang dipilih sesuai keinginan. Anda dapat membuat GSI, memperbarui throughput yang disediakan, dan opsi lainnya.
5. Jika Anda memilih Buat GSI, Perbarui Throughput yang Disediakan atau Perbarui Throughput yang Disediakan untuk GSI, opsi pengaturan Kapasitas akan muncul.
6. Untuk menyesuaikan pengaturan kapasitas, buka Pengaturan Kapasitas. Kemudian hapus centang opsi untuk Mewarisi pengaturan kapasitas dari tabel dasar atau Sesuai permintaan.
 - Anda sekarang dapat memilih Kapasitas sesuai permintaan atau Yang disediakan.

Dengan Provisioned dipilih, Anda dapat mengatur unit kapasitas baca dan tulis minimum dan maksimum. Anda juga dapat mengaktifkan atau menonaktifkan penskalaan otomatis.

- Jika tabel saat ini disetel ke OnDemand, Anda tidak akan dapat menentukan throughput yang disediakan.
 - Jika Anda beralih dari OnDemand ke throughput Provisioned, maka Autoscaling akan secara otomatis diterapkan ke semua GSI dengan: min: 1, max: 10; target: 70%.
7. Jika Anda ingin menghapus pengaturan yang baru saja Anda masukkan dan mulai dari awal, Hapus formulir.
 8. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

9. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.

Hasil operasi akan muncul di tab hasil di bagian bawah layar.

10. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

Operation builder

Connection ← +

Name
user1

Region
us-east-1

Tables ↻

Search table

DeleteMe1

NoSQLTest1

Saved operations 🗑️

Search saved op

No saved operations found

Build operations ⓘ

PartiQL operations
Interface-based operations

Operations
UpdateTable

Table name
NoSQLTest1

Update operations
Create GSI

Global secondary indexes ⓘ

* Global secondary index name

* Partition key

Add sort key ⓘ

Projection type

Capacity settings ⓘ

Inherit capacity settings from base table

By default, global secondary indexes inherit the capacity settings of the base table. However, if the table is using provisioned capacity mode, you can establish the capacity settings of each global secondary index individually.

Clear form
Run
Generate code
Save operation

Untuk informasi selengkapnya tentang operasi ini, lihat [Memperbarui tabel](#) di Referensi API Amazon DynamoDB.

Operasi item

Get item

Untuk menjalankan atau menghasilkan kode untuk suatu operasi Get Item, lakukan hal berikut.

1. Tentukan nilai kunci partisi.
2. Jika Anda ingin menambahkan ekspresi proyeksi, lakukan hal berikut:

- a. Pilih Ekspresi proyeksi.
- b. Pilih + (tanda tambah) di samping Ekspresi proyeksi.
- c. Tentukan Nama atribut.

Data plane operations: GetItem

Table: Music

* Partition key: Attribute value for partition key: Artist

* Sort key: Attribute value for sort key: SongTitle

Projection expression

* Projected attribute: Attribute name

Other parameters

Clear form Run Generate code Save operation

3. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih bahasa yang Anda inginkan dari tab yang ditampilkan. Anda sekarang dapat menyalin kode ini dan menggunakannya di aplikasi Anda.

4. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
5. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

Untuk informasi selengkapnya tentang operasi ini, lihat [GetItem](#) di Referensi Amazon DynamoDB API.

Put item

Untuk menjalankan atau menghasilkan kode untuk suatu operasi Put Item, lakukan hal berikut.

1. Tentukan nilai kunci partisi.

2. Tentukan nilai kunci urutan, jika ada.
3. Jika Anda ingin menambahkan atribut non-kunci, lakukan hal berikut:
 - a. Pilih + (tanda tambah) di samping Atribut lain.
 - b. Tentukan Nama atribut, Jenis, dan Nilai.
4. Jika ekspresi kondisi harus dipenuhi agar operasi Put Item berhasil, lakukan hal berikut:
 - a. Pilih Kondisi.
 - b. Tentukan nama atribut, operator perbandingan, jenis atribut, dan nilai atribut.
 - c. Jika kondisi lain dibutuhkan, pilih Kondisi lagi.

Untuk informasi selengkapnya, lihat [Ekspresi kondisi](#).

Data plane operations
Table

PutItem

Music

* Partition key i

* Sort key i

Other attributes + i

Attribute name	Attribute type	Attribute value
Attribute name	String	<empty>

Condition expression + Condition + Child expression i

And
Or
Negate

* Condition Negate

Attribute name
=
Attribute type
Attribute value
i

Clear form
Run
Generate code
Save operation

5. Jika Anda ingin membuat kode, pilih **Buat kode**.

Pilih bahasa yang Anda inginkan dari tab yang ditampilkan. Anda sekarang dapat menyalin kode ini dan menggunakannya di aplikasi Anda.

6. Jika Anda ingin operasi segera dijalankan, pilih **Jalankan**.

7. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih **Simpan operasi**, lalu masukkan nama untuk operasi Anda dan pilih **Simpan**.

Untuk informasi selengkapnya tentang operasi ini, lihat [PutItem](#) di Referensi Amazon DynamoDB API.

Perbarui item

Untuk menjalankan atau menghasilkan kode untuk operasi `Update Item`, lakukan hal berikut:

1. Masukkan nilai kunci partisi.
2. Masukkan nilai kunci urutan, jika ada.
3. Dalam `Perbarui ekspresi`, pilih ekspresi dalam daftar.
4. Pilih **+** (tanda tambah) untuk ekspresi.
5. Masukkan nama atribut dan nilai atribut untuk ekspresi yang dipilih.
6. Jika Anda ingin menambahkan lebih banyak ekspresi, pilih ekspresi lain di daftar `Perbarui Ekspresi`, lalu pilih **+**.
7. Jika ekspresi kondisi harus dipenuhi agar operasi `Update Item` berhasil, lakukan hal berikut:
 - a. Pilih `Kondisi`.
 - b. Tentukan nama atribut, operator perbandingan, jenis atribut, dan nilai atribut.
 - c. Jika kondisi lain dibutuhkan, pilih `Kondisi` lagi.

Untuk informasi selengkapnya, lihat [Ekspresi kondisi](#).

Data plane operations

UpdateItem
▼

Table

Music
▼

* Partition key

Attribute value for partition key: Artist
i

* Sort key

Attribute value for sort key: SongTitle
i

Update expression

Select operation
▼
+
i

Condition expression

+ Condition

+ Child expression

i

And

Or

Negate

* Condition

Negate

Attribute nan
=
▼

Attribute t
▼

Attribute val
i

Clear form

Run

Generate code

Save operation

8. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

9. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.

10. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

Untuk informasi selengkapnya tentang operasi ini, lihat [UpdateItem](#) di Referensi Amazon DynamoDB API.

Hapus item

Untuk menjalankan atau menghasilkan kode untuk suatu operasi Delete Item, lakukan hal berikut.

1. Masukkan nilai kunci partisi.

2. Masukkan nilai kunci urutan, jika ada.
3. Jika ekspresi kondisi harus dipenuhi agar operasi Delete Item berhasil, lakukan hal berikut:
 - a. Pilih Kondisi.
 - b. Tentukan nama atribut, operator perbandingan, jenis atribut, dan nilai atribut.
 - c. Jika kondisi lain dibutuhkan, pilih Kondisi lagi.

Untuk informasi selengkapnya, lihat [Ekspresi kondisi](#).

Data plane operations

Table

DeleteItem

Music

* Partition key

* Sort key

Condition expression

+ Condition
+ Child expression
i

And
Or
Negate

* Condition Negate

Attribute nan

=

Attribute t

Attribute val

i

Clear form

Run

Generate code

Save operation

4. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

5. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
6. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

Untuk informasi selengkapnya tentang operasi ini, lihat [DeleteItem](#) di Referensi Amazon DynamoDB API.

Query

Untuk menjalankan atau menghasilkan kode untuk suatu operasi Query, lakukan hal berikut.

1. Tentukan nilai kunci partisi.
2. Jika kunci urutan diperlukan untuk operasi Query:
 - a. Pilih Kunci urutan.
 - b. Tentukan, operator perbandingan, jenis atribut, dan nilai atribut.
3. Jika tidak semua atribut harus dikembalikan bersama hasil operasi, pilih Ekspresi proyeksi.
4. Pilih + (tanda tambah).
5. Masukkan atribut yang akan dikembalikan dengan hasil kueri.
6. Jika diperlukan lebih banyak atribut, pilih + .
7. Jika ekspresi kondisi harus dipenuhi agar operasi Query berhasil, lakukan hal berikut:
 - a. Pilih Kondisi.
 - b. Tentukan nama atribut, operator perbandingan, jenis atribut, dan nilai atribut.
 - c. Jika kondisi lain dibutuhkan, pilih Kondisi lagi.

Untuk informasi selengkapnya, lihat [Ekspresi kondisi](#).

Data plane operations Table

Query Music

* Partition key Attribute value for partition key: Artist i

Sort key i

Projection expression i +

Filter expression + Condition + Child expression i

And Or Negate

* Condition Negate

Attribute nan = Attribute t Attribute val i

Other parameters >

Clear form Run Generate code Save operation

8. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

9. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.

10. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

Untuk informasi selengkapnya tentang operasi ini, lihat [Kueri](#) dalam Referensi API Amazon DynamoDB.

Pemindaian

Untuk menjalankan atau menghasilkan kode untuk suatu operasi Scan, lakukan hal berikut.

1. Jika tidak semua atribut harus dikembalikan bersama hasil operasi, pilih Ekspresi proyeksi.
2. Pilih + (tanda tambah).
3. Tentukan atribut yang akan dikembalikan dengan hasil kueri.
4. Jika diperlukan lebih banyak atribut, pilih + lagi.
5. Jika ekspresi kondisi harus dipenuhi agar operasi pemindaian berhasil, lakukan hal berikut:
 - a. Pilih Kondisi.
 - b. Tentukan nama atribut, operator perbandingan, jenis atribut, dan nilai atribut.
 - c. Jika kondisi lain dibutuhkan, pilih Kondisi lagi.

Untuk informasi selengkapnya, lihat [Ekspresi kondisi](#).

6. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

7. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
8. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

TransactGetItems

Untuk menjalankan atau menghasilkan kode untuk suatu operasi `TransactGetItems`, lakukan hal berikut.

1. Pilih + (tanda tambah).
2. Ikuti petunjuk untuk operasi [Get item](#).

Setelah Anda selesai menentukan detail operasi, pilih + (tanda tambah) jika Anda ingin menambahkan operasi lainnya.

Data plane operations

TransactGetItems

TransactGetItem (+)



1. GetItem

1. GetItem

Table Name Music

* Partition key Attribute value for partition key: Artist

* Sort key Attribute value for sort key: SongTitle

 Projection expression (+)

* Projected attribute Attribute name

Clear item

Run

Generate code

Save operation

Untuk mengubah urutan tindakan, pilih tindakan dalam daftar di sisi kiri, lalu pilih panah atas atau bawah untuk memindahkannya ke atas atau bawah dalam daftar.

Untuk menghapus tindakan, pilih tindakan dalam daftar, kemudian pilih ikon Hapus (tempat sampah).

3. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

4. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
5. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

Untuk informasi selengkapnya tentang transaksi, lihat [Amazon DynamoDB Transactions](#).

TransactWriteItems

Untuk menjalankan atau menghasilkan kode untuk operasi `TransactWriteItems`, lakukan hal berikut.

1. Dalam daftar menurun Tindakan, pilih operasi yang Anda inginkan.
 - Untuk `DeleteItem`, ikuti petunjuk untuk operasi [Hapus item](#).
 - Untuk `PutItem`, ikuti petunjuk untuk operasi [Put item](#).
 - Untuk `UpdateItem`, ikuti petunjuk untuk operasi [Perbarui item](#).

Setelah selesai menentukan detail operasi, pilih tombol +.

Data plane operations

TransactWriteItems

Actions

PutItem ^ +

DeleteItem

PutItem

UpdateItem

ConditionCheck

1. PutItem

PutItem

Table name Music

* Partition key Attribute value for partition key: Artist

* Sort key Attribute value for sort key: SongTitle

Other attributes +

Condition expression + Condition + Child expression

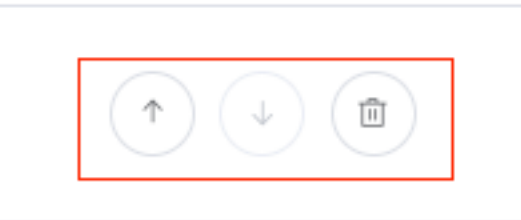
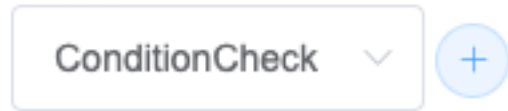
Clear

Run Generate code Save operation

Untuk mengubah urutan tindakan, pilih tindakan dalam daftar di sisi kiri, lalu pilih panah atas atau bawah untuk memindahkannya ke atas atau bawah dalam daftar.

Untuk menghapus tindakan, pilih tindakan dalam daftar, kemudian pilih ikon Hapus (tempat sampah).

Actions



1. DeleteItem

2. PutItem

3. ConditionCheck

2. Jika Anda ingin membuat kode, pilih Buat kode.

Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

3. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.

4. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

Untuk informasi selengkapnya tentang transaksi, lihat [Amazon DynamoDB Transactions](#).

Kloning tabel dengan NoSQL Workbench

Tabel kloning akan menyalin skema kunci tabel (dan skema dan item GSI opsional) di antara lingkungan pengembangan Anda. Anda dapat mengkloning tabel antara DynamoDB lokal ke akun

Amazon DynamoDB, dan bahkan mengkloning tabel dari satu akun ke akun lain di Wilayah yang berbeda untuk eksperimen yang lebih cepat.

Untuk mengkloning tabel

1. Di Operation Builder, pilih koneksi Anda dan Wilayah (Pilihan wilayah tidak tersedia untuk DynamoDB lokal).
2. Setelah Anda terhubung ke DynamoDB, telusuri tabel Anda dan pilih tabel yang ingin Anda kloning.
3. Pilih tombol Clone.

Items returned (15)
Consumed 1 RCUs.

Clone

Export to CSV

#	PrimaryKey ⓘ	Attribute1	Attribute2
1	Item_39	ggdqztcaiw	61
2	Item_32	wljiouytoj	42
3	Item_3	aniqhpnzsb	86
4	Item_18	esettqagqv	67

4. Masukkan detail tujuan klon Anda:
 - a. Pilih koneksi.
 - b. Pilih Wilayah (Wilayah tidak tersedia untuk DynamoDB lokal).
 - c. Masukkan nama tabel baru.
 - d. Pilih opsi klon:
 - i. Skema kunci dipilih secara default dan tidak dapat dibatalkan. Secara default, kloning tabel akan menyalin kunci utama Anda dan mengurutkan kunci jika tersedia.
 - ii. Skema GSI dipilih secara default jika tabel Anda yang akan dikloning memiliki GSI. Mengkloning tabel akan menyalin kunci utama GSI Anda dan kunci pengurutan jika tersedia. Anda memiliki opsi untuk membatalkan pilihan skema GSI untuk melewati kloning skema GSI. Mengkloning tabel akan menyalin pengaturan kapasitas tabel dasar Anda sebagai pengaturan kapasitas GSI. Anda dapat menggunakan `UpdateTable` operasi di Operation Builder untuk memperbarui pengaturan kapasitas GSI tabel setelah kloning selesai.

5. Masukkan jumlah item yang akan dikloning. Untuk hanya mengkloning skema kunci dan secara opsional skema GSI, Anda dapat menyimpan Item untuk mengkloning nilai pada 0. Jumlah maksimum item yang dapat dikloning adalah 5000.
6. Pilih mode kapasitas:
 - a. Mode sesuai permintaan dipilih secara default. DynamoDB on-demand pay-per-request menawarkan harga untuk permintaan baca dan tulis sehingga Anda hanya membayar untuk apa yang Anda gunakan. Untuk mempelajari lebih lanjut, lihat Mode [DynamoDB On-Demand](#).
 - b. Mode yang disediakan memungkinkan Anda menentukan jumlah pembacaan dan penulisan per detik yang Anda perlukan untuk aplikasi Anda. Anda dapat menggunakan penskalaan otomatis untuk menyesuaikan kapasitas yang disediakan tabel Anda secara otomatis sebagai respons terhadap perubahan lalu lintas. Untuk mempelajari selengkapnya, lihat [DynamoDB Provisioned mode](#).
7. Pilih Clone untuk memulai kloning.

Clone table smallTable

Cloning the table **smallTable** will copy the table's key schema and optionally the GSI schema and data to your specified destination. If you exit NoSQL Workbench before cloning is complete, the cloning process will be interrupted. Some items might fail to copy due to your table and capacity settings.

Connection * default

Region * us-east-1

New table name * smallTableClone

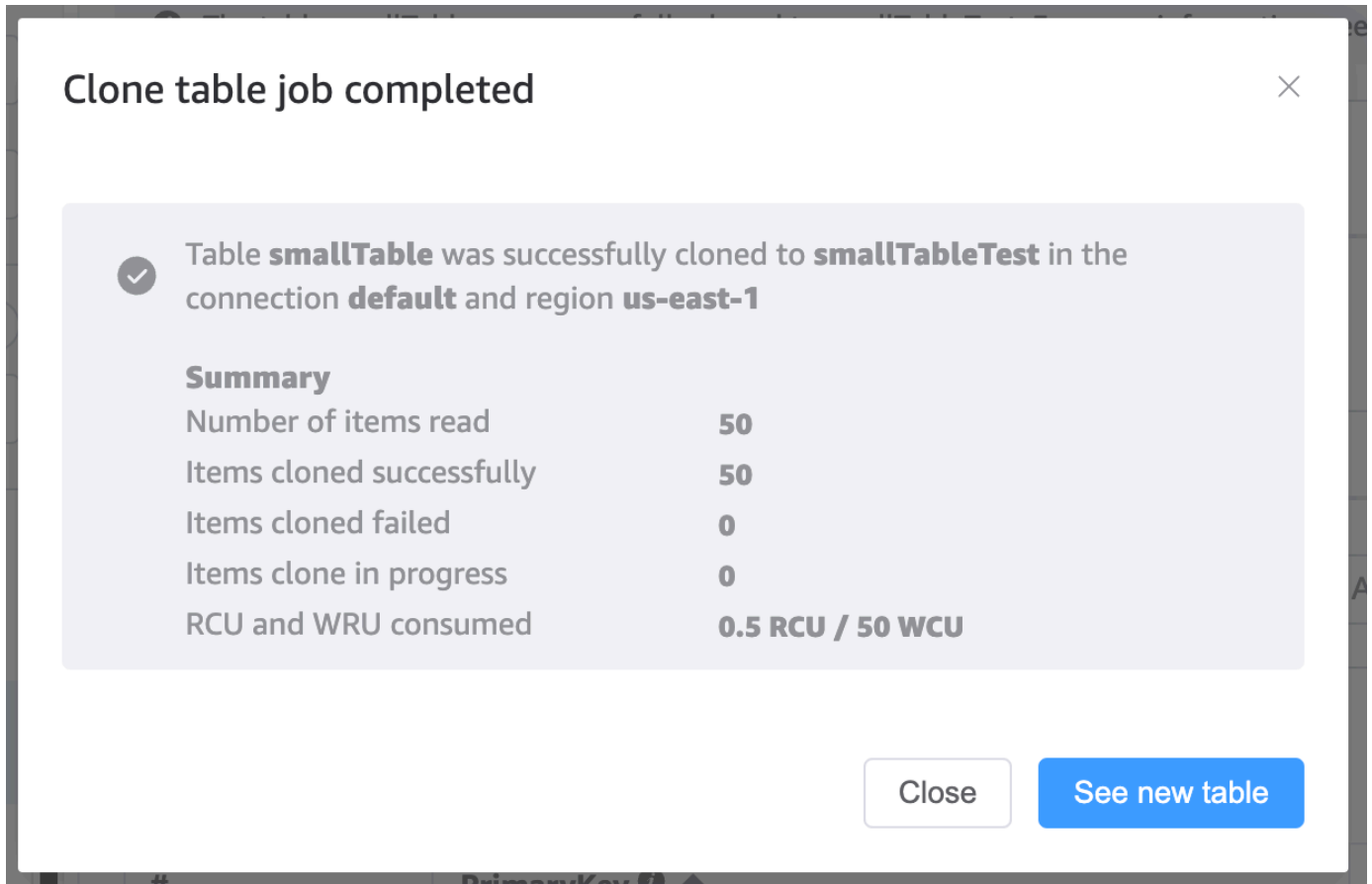
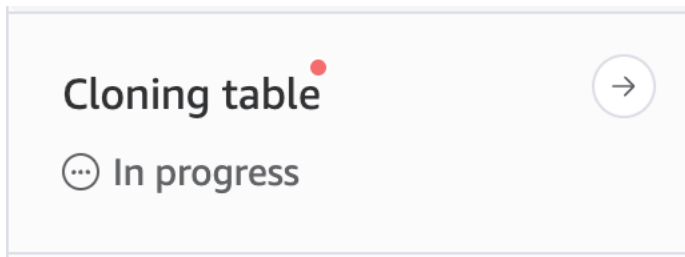
Clone *i* Key schema GSI schema

Items to clone *i* - 0 +

Capacity mode *i* On-demand Provisioned

Cancel Clone

8. Proses kloning akan berjalan di latar belakang. Tab Operation builder akan menampilkan pemberitahuan ketika ada perubahan dalam status tabel kloning. Anda dapat mengakses status ini dengan memilih tab Operation builder dan kemudian memilih tombol panah. Tombol panah terletak di widget status tabel kloning yang terletak di dekat bagian bawah bilah sisi menu.



Mengekspor data ke file CSV

Anda dapat mengekspor hasil kueri dari Pembangun Operasi ke file CSV. Tindakan ini memungkinkan Anda untuk memuat data ke dalam spreadsheet atau memprosesnya menggunakan bahasa pemrograman pilihan Anda.

Mengekspor ke CSV

1. Di Pembangun Operasi, jalankan operasi pilihan Anda, seperti Pindai atau Kueri.

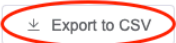
Note

- Anda hanya dapat mengekspor hasil dari operasi API baca dan pernyataan PartiQL ke file CSV. Anda tidak dapat mengekspor hasil dari pernyataan baca transaksi.
- Saat ini, Anda dapat mengekspor hasil satu halaman pada satu waktu ke file CSV. Jika ada beberapa halaman hasil, Anda harus mengekspor setiap halaman satu per satu.

2. Di panel Hasil, pilih Ekspor ke CSV.

Results

Items returned (2)

A button with a downward arrow icon and the text "Export to CSV". The button is highlighted with a red oval.

#	Awards ⇅	AlbumTitle ⇅	Artist ⇅	SongTitle ⇅
1	1	Somewhat Famous	No One You Know	Call My Today
2	10	Updated Album Title 2	Acme Band	Happy Day

3. Pilih nama file dan lokasi untuk file ZIP, lalu pilih Simpan.

Model data Sampel untuk NoSQL Workbench

Beranda untuk pemodel dan pemvisualisasi menampilkan sejumlah model sampel yang dikirimkan dengan NoSQL Workbench. Bagian ini menjelaskan model tersebut dan potensi penggunaannya.

Topik

- [Model data karyawan](#)
- [Model data diskusi](#)
- [Model data musik](#)
- [Model data ski](#)
- [Kartu kredit menawarkan model data](#)
- [Model data bookmark](#)

Model data karyawan

Model data ini adalah model pengenalan. Ini melambangkan detail dasar karyawan seperti alias unik, nama depan, nama belakang, penunjukan, manajer, dan keterampilan.

Model data ini menggambarkan beberapa teknik seperti penanganan atribut kompleks seperti memiliki lebih dari satu keterampilan. Model ini juga merupakan contoh dari one-to-many hubungan melalui manajer dan karyawan pelaporan mereka yang telah dicapai oleh indeks sekunder `DirectReports`.

Pola akses yang difasilitasi oleh model data ini adalah:

- Pengambilan catatan karyawan menggunakan alias login karyawan, difasilitasi oleh tabel yang disebut `Employee`.
- Mencari karyawan berdasarkan nama, difasilitasi oleh indeks sekunder global tabel Karyawan yang disebut `Name`.
- Pengambilan semua laporan langsung dari manajer menggunakan alias login manajer, difasilitasi oleh indeks sekunder global tabel Karyawan yang disebut `DirectReports`.

Model data diskusi

Model data ini melambangkan forum diskusi. Menggunakan model ini pelanggan dapat terlibat dengan komunitas developer, mengajukan pertanyaan, dan menanggapi posting pelanggan lain. Setiap layanan AWS memiliki forum khusus. Siapa pun dapat memulai utas diskusi baru dengan memposting pesan di forum, dan setiap utas menerima sejumlah balasan.

Pola akses yang difasilitasi oleh model data ini adalah:

- Pengambilan catatan forum menggunakan nama forum, difasilitasi oleh tabel yang disebut `Forum`.
- Pengambilan utas tertentu atau semua utas untuk forum, difasilitasi oleh tabel yang disebut `Thread`.
- Cari balasan menggunakan alamat email pengguna yang memposting, difasilitasi oleh indeks sekunder global tabel Balas yang disebut `PostedBy-Message-Index`.

Model data musik

Model data ini mewakili pustaka musik yang memiliki banyak koleksi lagu dan menampilkan lagu yang paling banyak diunduh dalam mendekati waktu nyata.

Pola akses yang difasilitasi oleh model data ini adalah:

- Pengambilan catatan lagu, difasilitasi oleh sebuah tabel yang disebut `Songs`.

- Pengambilan catatan unduhan tertentu atau semua catatan unduhan untuk sebuah lagu, yang difasilitasi oleh tabel yang disebut Songs.
- Pengambilan catatan hitungan unduhan bulanan tertentu atau semua catatan hitungan unduhan bulanan untuk sebuah lagu, yang difasilitasi oleh tabel yang disebut Song.
- Pengambilan semua catatan (termasuk catatan lagu, catatan unduhan, dan catatan hitungan unduhan bulanan) untuk lagu, difasilitasi oleh tabel yang disebut Songs.
- Cari lagu yang paling banyak diunduh, yang difasilitasi oleh indeks sekunder global tabel Lagu yang disebut DownloadsByMonth.

Model data ski

Model data ini mewakili resor ski yang memiliki koleksi data yang besar untuk setiap lift ski yang dikumpulkan setiap hari.

Pola akses yang difasilitasi oleh model data ini adalah:

- Pengambilan semua data untuk lift ski tertentu atau resor secara keseluruhan, dinamis dan statis, difasilitasi oleh tabel yang disebut SkiLifts.
- Pengambilan semua data dinamis (termasuk penunggang lift unik, cakupan salju, bahaya longsor salju, dan status lift) untuk lift ski atau resor secara keseluruhan pada tanggal tertentu, difasilitasi oleh tabel yang disebut SkiLifts.
- Pengambilan semua data statis (termasuk jika lift hanya untuk pengendara berpengalaman, kaki vertikal lift naik, dan waktu berkendara lift) untuk lift ski tertentu, difasilitasi oleh tabel yang disebut SkiLifts.
- Pengambilan tanggal data yang direkam untuk lift ski tertentu atau resor secara keseluruhan diurutkan berdasarkan total pengendara unik, difasilitasi oleh indeks sekunder global SkiLifts tabel yang disebut SkiLiftsByRiders.

Kartu kredit menawarkan model data

Model data ini digunakan oleh Aplikasi Penawaran Kartu Kredit.

Penyedia kartu kredit menghasilkan penawaran dari waktu ke waktu. Penawaran ini termasuk transfer saldo tanpa biaya, peningkatan batas kredit, suku bunga lebih rendah, uang kembali, dan miles perusahaan penerbangan. Setelah pelanggan menerima atau menolak penawaran ini, status penawaran masing-masing diperbarui.

Pola akses yang difasilitasi oleh model data ini adalah:

- Pengambilan catatan akun menggunakan `AccountId`, seperti yang difasilitasi oleh tabel utama.
- Pengambilan semua akun dengan beberapa item yang diproyeksikan, seperti yang difasilitasi oleh indeks sekunder `AccountIndex`.
- Pengambilan akun dan semua catatan penawaran yang terkait dengan akun tersebut dengan menggunakan `AccountId`, seperti yang difasilitasi oleh tabel utama.
- Pengambilan akun dan catatan penawaran yang terkait dengan akun tersebut dengan menggunakan `AccountId` dan `OfferId`, seperti yang difasilitasi oleh tabel utama.
- Pengambilan semua catatan penawaran `ACCEPTED/DECLINED` dari `OfferType` spesifik terkait dengan akun yang menggunakan `AccountId`, `OfferType`, dan `Status`, seperti yang difasilitasi oleh indeks sekunder `GSI1`.
- Pengambilan penawaran dan catatan item penawaran terkait menggunakan `OfferId`, seperti yang difasilitasi oleh tabel utama.

Model data bookmark

Model data ini digunakan untuk menyimpan bookmark bagi pelanggan.

Pelanggan dapat memiliki banyak bookmark dan bookmark dapat menjadi milik banyak pelanggan. Model data ini mewakili many-to-many hubungan.

Pola akses yang difasilitasi oleh model data ini adalah:

- Kueri tunggal oleh `customerId` sekarang dapat mengembalikan data pelanggan serta bookmark.
- Indeks `ByEmail` kueri mengembalikan data pelanggan melalui alamat email. Perhatikan bahwa bookmark tidak diambil oleh indeks ini.
- Indeks `ByUrl` kueri mendapat data bookmark dengan URL. Perhatikan bahwa kita memiliki `customerId` sebagai kunci pengurutan untuk indeks karena URL yang sama dapat di-bookmark oleh beberapa pelanggan.
- Indeks `ByCustomerFolder` kueri mendapat bookmark dengan folder untuk setiap pelanggan.

Riwayat rilis untuk NoSQL Workbench

Tabel berikut menjelaskan perubahan penting dalam setiap rilis alat klien NoSQL Workbench.

Versi	Perubahan	Deskripsi	Tanggal
3.12.0	Kloning tabel dengan NoSQL Workbench dan mengembalikan kapasitas yang dikonsumsi	Anda sekarang dapat mengkloning tabel antara DynamoDB lokal dan akun layanan web DynamoDB atau antara akun layanan web DynamoDB untuk iterasi pengembangan yang lebih cepat. Lihat RCU atau WCU yang dikonsumsi setelah menjalankan operasi menggunakan Operations Builder. Kami memperbaiki masalah data tumpang tindih saat mengimpor dari file CSV.	Februari 26, 2024
3.11.0	DynamoDB perbaikan lokal	Anda sekarang dapat menentukan port saat meluncurkan instance lokal DynamoDB bawaan. NoSQL Workbench sekarang dapat diinstal pada Windows tanpa hak admin. Kami telah memperbarui template model data.	Januari 17, 2024
3.10.0	Dukungan asli untuk silikon Apple	NoSQL Workbench sekarang menyertakan dukungan asli	5 Desember 2023

Versi	Perubahan	Deskripsi	Tanggal
		untuk Mac dengan silikon Apple. Anda sekarang dapat mengkonfigurasi format pembuatan data sampel untuk atribut tipeNumber.	
3.9.0	Perbaikan pemodel data	Pemvisualisasi sekarang mendukung melakukan model data ke DynamoDB lokal dengan opsi untuk menimpa tabel yang ada.	3 November 2023
3.8.0	Pembuatan data sampel	NoSQL Workbench kini mendukung pembuatan data sampel untuk model data DynamoDB Anda.	25 September 2023
3.6.0	Peningkatan dalam Pembangun operasi	Peningkatan manajemen koneksi di pembuat Operasi. Nama atribut di Pemodel Data sekarang dapat diubah tanpa menghapus data. Perbaikan bug lainnya.	11 April 2023

Versi	Perubahan	Deskripsi	Tanggal
3.5.0	Support untuk AWS Wilayah baru	NoSQL Workbench sekarang mendukung wilayah ap-south-2, ap-southeast-3, ap-southeast-4, eu-central-2, eu-south-2, me-central-1, and me-west-1.	23 Februari 2023
3.4.0	Dukungan untuk DynamoDB lokal	NoSQL Workbench sekarang mendukung instalasi DynamoDB lokal sebagai bagian dari proses instalasi.	6 Desember 2022
3.3.0	Dukungan untuk operasi bidang kontrol	Pembangun Operasi sekarang mendukung operasi bidang kontrol.	1 Juni 2022
3.2.0	Impor dan ekspor CSV	Anda sekarang dapat mengimpor data sampel dari file CSV di alat Visualizer, dan juga mengekspor hasil kueri Operation Builder ke file CSV.	11 Oktober 2021
3.1.0	Simpan operasi	Operation Builder di NoSQL Workbench sekarang mendukung operasi penyimpanan untuk digunakan nanti.	12 Juli 2021

Versi	Perubahan	Deskripsi	Tanggal
3.0.0	Pengaturan kapasitas dan CloudFormation impor/ekspor	NoSQL Workbench untuk Amazon DynamoDB sekarang mendukung penentuan mode kapasitas baca/tulis untuk tabel serta dapat mengimpor dan mengekspor model data dalam format AWS CloudFormation .	21 April 2021
2.2.0	Dukungan untuk PartiQL	NoSQL Workbench untuk Amazon DynamoDB menambahkan dukungan untuk membangun pernyataan PartiQL untuk DynamoDB.	4 Desember 2020
1.1.0	Dukungan untuk Linux.	NoSQL Workbench untuk Amazon DynamoDB didukung pada Linux—Ubuntu, Fedora dan Debian.	4 Mei 2020
1.0.0	NoSQL Workbench untuk Amazon DynamoDB — GA.	NoSQL Workbench untuk Amazon DynamoDB tersedia secara umum.	2 Maret 2020

Versi	Perubahan	Deskripsi	Tanggal
0.4.1	Dukungan untuk peran IAM dan kredensial keamanan sementara.	NoSQL Workbench untuk Amazon DynamoDB menambahkan dukungan untuk peran AWS Identity and Access Management (IAM) dan kredensial keamanan sementara.	19 Desember 2019
0.3.1	Dukungan untuk DynamoDB lokal (Versi Dapat Diunduh) .	NoSQL Workbench kini mendukung koneksi ke DynamoDB lokal (Versi Dapat Diunduh) untuk merancang, membuat, mengkueri, dan mengelola tabel DynamoDB.	8 November, 2019
0.2.1	Pratinjau NoSQL Workbench dirilis.	Ini adalah rilis awal NoSQL Workbench untuk DynamoDB. Gunakan NoSQL Workbench untuk merancang, membuat, mengkueri, dan mengelola tabel DynamoDB.	16 September 2019

Contoh kode untuk DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menggunakan DynamoDB dengan kit pengembangan perangkat lunak (AWS SDK).

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Contoh lintas layanan adalah contoh aplikasi yang bekerja di beberapa Layanan AWS.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga berisi informasi tentang cara memulai dan detail tentang versi SDK sebelumnya.

Memulai

Halo DynamoDB

Contoh kode berikut ini menunjukkan cara untuk mulai menggunakan DynamoDB.

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace DynamoDB_Actions;

public static class HelloDynamoDB
```

```
{
    static async Task Main(string[] args)
    {
        var dynamoDbClient = new AmazonDynamoDBClient();

        Console.WriteLine($"Hello Amazon Dynamo DB! Following are some of your
tables:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five tables.
        var response = await dynamoDbClient.ListTablesAsync(
            new ListTablesRequest()
            {
                Limit = 5
            });

        foreach (var table in response.TableNames)
        {
            Console.WriteLine($"\\tTable: {table}");
            Console.WriteLine();
        }
    }
}
```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for .NET API.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kode untuk file CMake MakeLists C.txt.

```
# 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)
  # 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})
```

Kode untuk file sumber hello_dynamodb.cpp.

```
#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;
            }
        } while (true);
    }
}
```

```
        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;
}
```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for C++ API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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!");
}
```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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;
};
```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for JavaScript API.

Contoh kode

- [Tindakan untuk DynamoDB menggunakan SDK AWS](#)
 - [Membuat tabel DynamoDB menggunakan SDK AWS](#)
 - [Menghapus tabel DynamoDB menggunakan SDK AWS](#)
 - [Menghapus item dari tabel DynamoDB menggunakan SDK AWS](#)
 - [Dapatkan sekumpulan item DynamoDB menggunakan SDK AWS](#)
 - [Mendapatkan item dari tabel DynamoDB menggunakan SDK AWS](#)
 - [Dapatkan informasi tentang tabel DynamoDB](#)
 - [Daftar tabel DynamoDB menggunakan SDK AWS](#)
 - [Menempatkan item dalam tabel DynamoDB menggunakan SDK AWS](#)
 - [Kueri tabel DynamoDB menggunakan SDK AWS](#)
 - [Jalankan pernyataan PartiQL pada tabel DynamoDB menggunakan SDK AWS](#)
 - [Jalankan batch pernyataan PartiQL pada tabel DynamoDB menggunakan SDK AWS](#)
 - [Memindai tabel DynamoDB menggunakan SDK AWS](#)
 - [Memperbarui item dalam tabel DynamoDB menggunakan SDK AWS](#)
 - [Menulis sekumpulan item DynamoDB menggunakan SDK AWS](#)
- [Skenario untuk DynamoDB menggunakan AWS SDK](#)
 - [Mempercepat pembacaan DynamoDB dengan DAX menggunakan SDK AWS](#)
 - [Memulai tabel, item, dan kueri DynamoDB menggunakan SDK AWS](#)
 - [Kueri tabel DynamoDB dengan menggunakan kumpulan pernyataan PartiQL dan SDK AWS](#)
 - [Kueri tabel DynamoDB menggunakan PartiQL dan SDK AWS](#)
 - [Menggunakan model dokumen untuk DynamoDB menggunakan SDK AWS](#)
 - [Menggunakan model persistensi objek tingkat tinggi untuk DynamoDB menggunakan SDK AWS](#)
- [Contoh lintas layanan untuk DynamoDB menggunakan SDK AWS](#)
 - [Membangun aplikasi untuk mengirimkan data ke tabel DynamoDB](#)
 - [Membuat API REST Gateway API untuk melacak data COVID-19](#)
 - [Membuat aplikasi messenger dengan Step Functions](#)
 - [Membuat aplikasi manajemen aset foto yang memungkinkan pengguna mengelola foto menggunakan label](#)
 - [Membuat aplikasi web untuk melacak data DynamoDB](#)
 - [Membuat aplikasi obrolan websocket dengan API Gateway](#)

- [Mendeteksi APD dalam gambar dengan Amazon AWS Rekognition menggunakan SDK](#)
- [Menginvokasi fungsi Lambda dari browser](#)
- [Simpan EXIF dan informasi gambar lainnya menggunakan SDK AWS](#)
- [Menggunakan API Gateway untuk menginvokasi fungsi Lambda](#)
- [Menggunakan Step Functions untuk menginvokasi fungsi Lambda](#)
- [Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda](#)

Tindakan untuk DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara melakukan tindakan DynamoDB individual dengan SDK. AWS Kutipan ini memanggil API DynamoDB dan merupakan kutipan kode dari program yang lebih besar yang harus dijalankan dalam konteks. Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan instruksi untuk mengatur dan menjalankan kode.

Contoh berikut hanya mencakup tindakan yang paling umum digunakan. Untuk daftar lengkapnya, lihat [Referensi API Amazon DynamoDB](#).

Contoh-contoh

- [Membuat tabel DynamoDB menggunakan SDK AWS](#)
- [Menghapus tabel DynamoDB menggunakan SDK AWS](#)
- [Menghapus item dari tabel DynamoDB menggunakan SDK AWS](#)
- [Dapatkan sekumpulan item DynamoDB menggunakan SDK AWS](#)
- [Mendapatkan item dari tabel DynamoDB menggunakan SDK AWS](#)
- [Dapatkan informasi tentang tabel DynamoDB](#)
- [Daftar tabel DynamoDB menggunakan SDK AWS](#)
- [Menempatkan item dalam tabel DynamoDB menggunakan SDK AWS](#)
- [Kueri tabel DynamoDB menggunakan SDK AWS](#)
- [Jalankan pernyataan PartiQL pada tabel DynamoDB menggunakan SDK AWS](#)
- [Jalankan batch pernyataan PartiQL pada tabel DynamoDB menggunakan SDK AWS](#)
- [Memindai tabel DynamoDB menggunakan SDK AWS](#)
- [Memperbarui item dalam tabel DynamoDB menggunakan SDK AWS](#)
- [Menulis sekumpulan item DynamoDB menggunakan SDK AWS](#)

Membuat tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara membuat tabel DynamoDB.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- [Mempercepat pembacaan dengan DAX](#)
- [Memulai tabel, item, dan kueri](#)

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
client, string tableName)
    {
        var response = await client.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,
    },
},
ProvisionedThroughput = new ProvisionedThroughput
{
    ReadCapacityUnits = 5,
    WriteCapacityUnits = 5,
},
});

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

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

TableStatus status;

int sleepDuration = 2000;

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

```

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

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

    return status == TableStatus.ACTIVE;
}

```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

#####
# 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.
#     -p provisioned_throughput -- Provisioned throughput settings for the
table.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.

```

```
#####  
function dynamodb_create_table() {  
    local table_name attribute_definitions key_schema provisioned_throughput  
    response  
    local option OPTARG # Required to use getopt command in a function.  
  
    #####  
    # Function usage explanation  
    #####  
    function usage() {  
        echo "function dynamodb_create_table"  
        echo "Creates an Amazon DynamoDB table."  
        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 " -p provisioned_throughput -- Provisioned throughput settings for the  
table."  
        echo ""  
    }  
  
    # Retrieve the calling parameters.  
    while getopt "n:a:k:p:h" option; do  
        case "${option}" in  
            n) table_name="${OPTARG}" ;;  
            a) attribute_definitions="${OPTARG}" ;;  
            k) key_schema="${OPTARG}" ;;  
            p) provisioned_throughput="${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."  
    fi  
}
```

```
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

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

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

response=$(aws dynamodb create-table \
  --table-name "$table_name" \
  --attribute-definitions file://"${attribute_definitions}" \
  --key-schema file://"${key_schema}" \
  --provisioned-throughput "${provisioned_throughput}")

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
```



```
fi

return 0
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
#####
# 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
}

```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#!/ 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 outcome.IsSuccess();
}
```

```
}
```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Contoh 1: Untuk membuat tabel dengan tag

`create-table` Contoh berikut menggunakan atribut tertentu dan skema kunci untuk membuat tabel bernama `MusicCollection`. Tabel ini menggunakan throughput yang disediakan dan dienkripsi saat istirahat menggunakan CMK yang dimiliki default. AWS Perintah ini juga menerapkan tag ke tabel, dengan kunci dari `Owner` dan nilai `blueTeam`.

```
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
```

Output:

```
{  
  "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"
}
}
```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 2: Untuk membuat tabel dalam Mode On-Demand

Contoh berikut membuat tabel yang disebut MusicCollection menggunakan mode on-demand, bukan mode throughput yang disediakan. Ini berguna untuk tabel dengan beban kerja yang tidak terduga.

```
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
```

Output:

```
{
  "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"
    }
  }
}
```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 3: Untuk membuat tabel dan mengenkripsi dengan Customer Managed CMK

Contoh berikut membuat tabel bernama MusicCollection dan mengenkripsi menggunakan CMK yang dikelola pelanggan.

```
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
```

Output:

```
{  
  "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"
    }
}
}
}

```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 4: Untuk membuat tabel dengan Indeks Sekunder Lokal

Contoh berikut menggunakan atribut tertentu dan skema kunci untuk membuat tabel bernama `MusicCollection` dengan Indeks Sekunder Lokal bernama `AlbumTitleIndex`.

```

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 \
  --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\"]
    }
  ]"
```

Output:

```
{
  "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"
      }
    ]
  }
}

```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 5: Untuk membuat tabel dengan Indeks Sekunder Global

Contoh berikut membuat tabel bernama GameScores dengan Global Secondary Index disebut GameTitleIndex. Tabel dasar memiliki kunci partisi UserId dan kunci urutan GameTitle, sehingga Anda dapat menemukan skor terbaik pengguna individu untuk game tertentu secara efisien, sedangkan GSI memiliki kunci partisi GameTitle dan kunci urutan

TopScore, memungkinkan Anda untuk cepat menemukan skor tertinggi secara keseluruhan untuk game tertentu.

```
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
        }
      }
    ]"
```

Output:

```
{
  "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"
  }
]
}
}

```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 6: Untuk membuat tabel dengan beberapa Indeks Sekunder Global sekaligus

Contoh berikut membuat tabel bernama GameScores dengan dua Global Secondary Indexes. Skema GSI diteruskan melalui file, bukan pada baris perintah.

```

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

```

Isi dari gsi.json:

```

[
  {
    "IndexName": "GameTitleIndex",
    "KeySchema": [
      {
        "AttributeName": "GameTitle",

```

```

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

```

Output:

```

{
  "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": "GameDataIndex",
    "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"
    }
]
}
}

```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 7: Untuk membuat tabel dengan Streams diaktifkan

Contoh berikut membuat tabel yang disebut GameScores dengan DynamoDB Streams diaktifkan. Gambar baru dan lama dari setiap item akan ditulis ke aliran.

```

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

```

Output:

```

{
  "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"
}
}
```

Untuk informasi selengkapnya, lihat [Operasi Dasar untuk Tabel](#) dalam Panduan Developer Amazon DynamoDB.

Contoh 8: Untuk membuat tabel dengan Keys-Only Stream diaktifkan

Contoh berikut membuat tabel yang disebut GameScores dengan DynamoDB Streams diaktifkan. Hanya atribut kunci dari item yang dimodifikasi yang ditulis ke aliran.

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

Output:

```
{
  "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
  }
}
```

```
}  
}
```

Untuk informasi selengkapnya, lihat [Mengubah pengambilan data untuk DynamoDB Streams di Panduan Pengembang Amazon DynamoDB](#).

Contoh 9: Untuk membuat tabel dengan kelas Standard Infrequent Access

Contoh berikut membuat tabel yang disebut GameScores dan menetapkan kelas tabel Standard-Infrequent Access (DynamoDB Standard-IA). Kelas tabel ini dioptimalkan untuk penyimpanan menjadi biaya dominan.

```
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
```

Output:

```
{  
  "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
}
}

```

Untuk informasi selengkapnya, lihat [Kelas tabel](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 10: Membuat tabel dengan Hapus Perlindungan diaktifkan

Contoh berikut membuat tabel yang disebut GameScores dan memungkinkan perlindungan penghapusan.

```

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

```

Output:

```
{
```

```
"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
}
}
```

Untuk informasi selengkapnya, lihat [Menggunakan perlindungan penghapusan](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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() (*types.TableDescription, error) {
    var tableDesc *types.TableDescription
    table, err := basics.DynamoDbClient.CreateTable(context.TODO(),
        &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),
  ProvisionedThroughput: &types.ProvisionedThroughput{
    ReadCapacityUnits:  aws.Int64(10),
    WriteCapacityUnits: aws.Int64(10),
  },
})
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(context.TODO(), &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
}
return tableDesc, err
}
```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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.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.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.
 *
 * For more information, see the following documentation topic:
 *
 * 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())
        .provisionedThroughput(ProvisionedThroughput.builder()
            .readCapacityUnits(10L)
            .writeCapacityUnits(10L)
            .build())
        .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 "";
  }
}
```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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",
      },
    ],
  });
};
```

```
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

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

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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 provisionedVal = ProvisionedThroughput {
        readCapacityUnits = 10
        writeCapacityUnits = 10
    }

    val request = CreateTableRequest {
        attributeDefinitions = listOf(attDef)
        keySchema = listOf(keySchemaVal)
        provisionedThroughput = provisionedVal
        tableName = tableNameVal
    }


    DynamoDbClient { 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
    }
}
```

- Untuk detail API, lihat [CreateTable](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat tabel.

```
$tableName = "ddb_demo_table_${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],
    ]);
}
```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat tabel untuk menyimpan data film.

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

    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=[
```



```

key
    {"AttributeName": "year", "KeyType": "HASH"}, # Partition
    {"AttributeName": "title", "KeyType": "RANGE"}, # Sort key
  ],
  AttributeDefinitions=[
    {"AttributeName": "year", "AttributeType": "N"},
    {"AttributeName": "title", "AttributeType": "S"},
  ],
  ProvisionedThroughput={
    "ReadCapacityUnits": 10,
    "WriteCapacityUnits": 10,
  },
)
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

```

- Untuk detail API, lihat [CreateTable](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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

```

```
attr_reader :table_name
attr_reader :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"}
    ],
    provisioned_throughput: {read_capacity_units: 10, write_capacity_units:
10})
  @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
```

- Untuk detail API, lihat [CreateTable](#) di Referensi AWS SDK for Ruby API.

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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 pt = ProvisionedThroughput::builder()
        .read_capacity_units(10)
        .write_capacity_units(5)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .provisioned_throughput(pt)
```

```

        .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))
        }
    }
}

```

- Untuk detail API, lihat [CreateTable](#) referensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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.

```

- Untuk detail API, lihat [CreateTable](#) di AWS SDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
///
/// Create a movie table in the Amazon DynamoDB data store.
///
private func createTable() async throws {
    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),
        ],
        keySchema: [
            DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
            DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
        ],
        provisionedThroughput: DynamoDBClientTypes.ProvisionedThroughput(
            readCapacityUnits: 10,
            writeCapacityUnits: 10
        ),
        tableName: self.tableName
    )
    let output = try await client.createTable(input: input)
    if output.tableDescription == nil {
        throw MoviesError.TableNotFound
    }
}
```

- Untuk detail API, lihat referensi [CreateTable AWS SDK](#) untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menghapus tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menghapus tabel DynamoDB.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- [Mempercepat pembacaan dengan DAX](#)
- [Memulai tabel, item, dan kueri](#)

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient
client, string tableName)
{
    var request = new DeleteTableRequest
    {
        TableName = tableName,
    };

    var response = await client.DeleteTableAsync(request);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
        return true;
    }
    else
    {
        Console.WriteLine("Could not delete table.");
        return false;
    }
}
```

- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#####
# 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 getopt 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 getopt "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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}

```

- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

//! 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();
}
```

- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Untuk menghapus tabel

`delete-table` Contoh berikut menghapus `MusicCollection` tabel.

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

Output:

```
{  
  "TableDescription": {  
    "TableStatus": "DELETING",  
    "TableSizeBytes": 0,  
    "ItemCount": 0,  
    "TableName": "MusicCollection",  
    "ProvisionedThroughput": {  
      "NumberOfDecreasesToday": 0,  
      "WriteCapacityUnits": 5,  
      "ReadCapacityUnits": 5  
    }  
  }  
}
```


```
}
```

Untuk informasi selengkapnya, lihat [Menghapus Tabel di Panduan](#) Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh selengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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() error {
    _, err := basics.DynamoDbClient.DeleteTable(context.TODO(),
        &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
}
```

- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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!");
}
}
```

- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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;
};
```

- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request = DeleteTableRequest {
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- Untuk detail API, lihat [DeleteTable](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
public function deleteTable(string $TableName)
{
    $this->customWaiter(function () use ($TableName) {
        return $this->dynamoDbClient->deleteTable([
            'TableName' => $TableName,
        ]);
    });
}
```

- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

```
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
```

- Untuk detail API, lihat [DeleteTable](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
```

```

attr_reader :dynamo_resource
attr_reader :table_name
attr_reader :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

```

- Untuk detail API, lihat [DeleteTable](#) di Referensi AWS SDK for Ruby API.

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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())),
}
}
```

- Untuk detail API, lihat [DeleteTable](#) referensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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_dynresourceindex.
  MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
CATCH /aws1/cx_dynresourceinuse.
  MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
ENDTRY.
```

- Untuk detail API, lihat [DeleteTable](#) di AWS SDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
///  
/// Deletes the table from Amazon DynamoDB.  
///  
func deleteTable() async throws {  
    guard let client = self.ddbClient else {  
        throw MoviesError.UninitializedClient  
    }  
  
    let input = DeleteTableInput(  
        tableName: self.tableName  
    )  
    _ = try await client.deleteTable(input: input)  
}
```

- Untuk detail API, lihat referensi [DeleteTable AWSSDK](#) untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menghapus item dari tabel DynamoDB menggunakan SDK AWS


Contoh kode berikut menunjukkan cara menghapus item dari tabel DynamoDB.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- [Memulai tabel, item, dan kueri](#)

.NET

AWS SDK for .NET

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// <summary>
/// Deletes a single item from a DynamoDB table.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <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 static async Task<bool> DeleteItemAsync(
    AmazonDynamoDBClient client,
    string tableName,
    Movie movieToDelete)
{
    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,
    };

    var response = await client.DeleteItemAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

#####
# 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 getopt 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

}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}
```

- Untuk detail API, lihat [Deleteltem](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#!/ 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 outcome.IsSuccess();  
}
```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Contoh 1: Untuk menghapus item

`delete-item` Contoh berikut menghapus item dari `MusicCollection` tabel dan meminta rincian tentang item yang telah dihapus dan kapasitas yang digunakan oleh permintaan.

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

Isi dari `key.json`:

```
{  
  "Artist": {"S": "No One You Know"},  
  "SongTitle": {"S": "Scared of My Shadow"}  
}
```

Output:

```
{  
  "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
    ]
  }
}

```

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk menghapus item secara kondisional

Contoh berikut menghapus item dari ProductCatalog tabel hanya jika salah satu Sporting Goods atau Gardening Supplies dan harganya antara 500 dan 600. ProductCategory la mengembalikan rincian tentang item yang telah dihapus.

```

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

```

Isi dari names.json:

```
{
```

```
"#P": "Price"
}
```

Isi dari `values.json`:

```
{
  ":cat1": {"S": "Sporting Goods"},
  ":cat2": {"S": "Gardening Supplies"},
  ":lo": {"N": "500"},
  ":hi": {"N": "600"}
}
```

Output:

```
{
  "Attributes": {
    "Id": {
      "N": "456"
    },
    "Price": {
      "N": "550"
    },
    "ProductCategory": {
      "S": "Sporting Goods"
    }
  }
}
```

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(movie Movie) error {
    _, err := basics.DynamoDbClient.DeleteItem(context.TODO(),
        &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
}

// 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"])
}
```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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);
}
}
```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [DeleteCommand](#).

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Hapus item dalam tabel.

```
// 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);
  }
});
```

```
}  
});
```

Hapus item dari tabel menggunakan klien dokumen DynamoDB.

```
// 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);  
  }  
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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 { region = "us-east-1" }.use { ddb ->
        ddb.deleteItem(request)
        println("Item with key matching $keyVal was deleted")
    }
}
```

- Untuk detail API, lihat [DeleteItem](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
$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,
    ]);
}
```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

    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

```

Anda dapat menentukan kondisi sehingga item dihapus hanya ketika memenuhi kriteria tertentu.

```

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
```

- Untuk detail API, lihat [DeleteItem](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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.
end
```

```
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
```

- Untuk detail API, lihat [DeleteItem](#) di Referensi AWS SDK for Ruby API.

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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)),
  }
}
```

- Untuk detail API, lihat [DeleteItem](#) referensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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.
```

- Untuk detail API, lihat [DeleteItem](#) di AWS SDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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 {
    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)
}
```

- Untuk detail API, lihat referensi [DeleteItem AWSSDK](#) untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Dapatkan sekumpulan item DynamoDB menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara untuk mendapatkan batch item DynamoDB.

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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);
    }
}

```

- Untuk detail API, lihat [BatchGetItem](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#####
```



```
# 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 getopt 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 getopt "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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}

```

- Untuk detail API, lihat [BatchGetItem](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

//! 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;
}

```

- Untuk detail API, lihat [BatchGetItem](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Untuk mengambil beberapa item dari tabel

`batch-get-items` Contoh berikut membaca beberapa item dari `MusicCollection` tabel menggunakan batch tiga `GetItem` permintaan, dan meminta jumlah unit kapasitas baca yang dikonsumsi oleh operasi. Perintah hanya mengembalikan `AlbumTitle` atribut.

```
aws dynamodb batch-get-item \  
  --request-items file://request-items.json \  
  --return-consumed-capacity TOTAL
```

Isi dari request-items.json:

```
{  
  "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"  
  }  
}
```

Output:

```
{  
  "Responses": {  
    "MusicCollection": [  
      {  
        "AlbumTitle": {  
          "S": "Somewhat Famous"  
        }  
      },  
      {  
        "AlbumTitle": {  
          "S": "Blue Sky Blues"  
        }  
      },  
      {
```

```
        "AlbumTitle": {
            "S": "Louder Than Ever"
        }
    ],
    "UnprocessedKeys": {},
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 1.5
        }
    ]
}
```

Untuk informasi selengkapnya, lihat [Operasi Batch](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [BatchGetItem](#) di Referensi AWS CLI Perintah.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

menunjukkan cara mendapatkan item batch menggunakan klien layanan.

```
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.");
}
}
```

menunjukkan cara mendapatkan item batch menggunakan klien layanan dan paginator.

```
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());
        });
}
```

```
}  
}
```

- Untuk detail API, lihat [BatchGetItem](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [BatchGet](#).

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [BatchGetItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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);
    });
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [BatchGetItem](#) di Referensi AWS SDK for JavaScript API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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

```

- Untuk detail API, lihat [BatchGetItem](#) di AWS SDK for Python (Boto3) Referensi API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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] {
    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 {
    movieList.append(try Movie(withItem: response))
}

return movieList
}
```

- Untuk detail API, lihat referensi [BatchGetItem AWSSDK](#) untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Mendapatkan item dari tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara mendapatkan item dari tabel DynamoDB.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- [Mempercepat pembacaan dengan DAX](#)
- [Memulai tabel, item, dan kueri](#)

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <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 static async Task<Dictionary<string, AttributeValue>>
    GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
    {
        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 client.GetItemAsync(request);
        return response.Item;
    }
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#####
# 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 getopt 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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}

```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

//! 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();  
}
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Contoh 1: Untuk membaca item dalam tabel

`get-item` Contoh berikut mengambil item dari `MusicCollection` tabel. Tabel memiliki kunci `hash-and-range` utama (`Artist` dan `SongTitle`), jadi Anda harus menentukan kedua atribut ini. Perintah tersebut juga meminta informasi tentang kapasitas baca yang dikonsumsi oleh operasi.

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

Isi dari `key.json`:

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

Output:

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

```
    "Artist": {
      "S": "Acme Band"
    },
    "ConsumedCapacity": {
      "TableName": "MusicCollection",
      "CapacityUnits": 0.5
    }
  }
}
```

Untuk informasi selengkapnya, lihat [Membaca Item](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk membaca item menggunakan pembacaan yang konsisten

Contoh berikut mengambil item dari MusicCollection tabel menggunakan pembacaan yang sangat konsisten.

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

Isi dari key.json:

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

Output:

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

```
        "S": "Acme Band"
    }
},
"ConsumedCapacity": {
    "TableName": "MusicCollection",
    "CapacityUnits": 1.0
}
}
```

Untuk informasi selengkapnya, lihat [Membaca Item](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 3: Untuk mengambil atribut tertentu dari suatu item

Contoh berikut menggunakan ekspresi proyeksi untuk mengambil hanya tiga atribut dari item yang diinginkan.

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

Isi dari `names.json`:

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

Output:

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

```
        "S": "Book"
    }
}
}
```

Untuk informasi selengkapnya, lihat [Membaca Item](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [GetItem](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(title string, year int) (Movie, error) {
    movie := Movie{Title: title, Year: year}
    response, err := basics.DynamoDbClient.GetItem(context.TODO(),
        &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
}

// 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"])
}
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mendapat item dari tabel dengan menggunakan DynamoDbClient.

```
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);
}
}
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [GetCommand](#).

```
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;
};
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Dapatkan item dari tabel.

```
// 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);
  }
});
```

Dapatkan item dari tabel menggunakan klien dokumen DynamoDB.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

- Untuk detail API, lihat [GetItem](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
$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,
    ]);
}
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

    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"]
```

- Untuk detail API, lihat [GetItem](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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
end
```

- Untuk detail API, lihat [GetItem](#) di Referensi AWS SDK for Ruby API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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.
```

- Untuk detail API, lihat [GetItem](#) di AWS SDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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 {
    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
}
```

- Untuk detail API, lihat referensi [GetItem AWSSDK](#) untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Dapatkan informasi tentang tabel DynamoDB

Contoh kode berikut ini menunjukkan cara mendapatkan informasi tentang tabel DynamoDB.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- [Memulai tabel, item, dan kueri](#)

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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}");
    Console.WriteLine($"Provision Throughput (reads/sec): " +
        $"{table.ProvisionedThroughput.ReadCapacityUnits}");
    Console.WriteLine($"Provision Throughput (writes/sec): " +
        $"{table.ProvisionedThroughput.WriteCapacityUnits}");
}
```

- Untuk detail API, lihat [DescribeTable](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#####
# 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 getopt 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 getopt "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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
#####
# 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
}
```

- Untuk detail API, lihat [DescribeTable](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#!/ 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();
}

```

- Untuk detail API, lihat [DescribeTable](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Untuk menggambarkan tabel

`describe-table` Contoh berikut menjelaskan `MusicCollection` tabel.

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

Output:


```
{  
  "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  
  }  
}
```

Untuk informasi selengkapnya, lihat [Menjelaskan Tabel di Panduan](#) Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [DescribeTable](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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() (bool, error) {
    exists := true
    _, err := basics.DynamoDbClient.DescribeTable(
        context.TODO(), &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
}
```

- Untuk detail API, lihat [DescribeTable](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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();

    describeDynamoDBTable(ddb, tableName);
    ddb.close();
}

public static void describeDynamoDBTable(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!");
}
}
```

- Untuk detail API, lihat [DescribeTable](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DescribeTable](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [DescribeTable](#) di Referensi AWS SDK for JavaScript API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

    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

```

- Untuk detail API, lihat [DescribeTable](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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

```

- Untuk detail API, lihat [DescribeTable](#) di Referensi AWS SDK for Ruby API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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.

```

- Untuk detail API, lihat [DescribeTable](#) di AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Daftar tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara mencantumkan tabel DynamoDB.

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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);
}
```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#####
# 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
}
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
#####
# 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
}
```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#!/ 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;  
}
```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Contoh 1: Untuk daftar tabel

`list-tables` Contoh berikut mencantumkan semua tabel yang terkait dengan AWS akun saat ini dan Wilayah.

```
aws dynamodb list-tables
```

Output:

```
{  
  "TableNames": [  
    "Forum",  
    "ProductCatalog",  
    "Reply",  
    "Thread"  
  ]  
}
```

Untuk informasi selengkapnya, lihat [Daftar Nama Tabel](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk membatasi ukuran halaman

Contoh berikut mengembalikan daftar semua tabel yang ada, tetapi mengambil hanya satu item dalam setiap panggilan, melakukan beberapa panggilan jika perlu untuk mendapatkan seluruh daftar. Membatasi ukuran halaman berguna saat menjalankan perintah daftar pada sejumlah besar sumber daya, yang dapat mengakibatkan kesalahan “waktu habis” saat menggunakan ukuran halaman default 1000.

```
aws dynamodb list-tables \  
  --page-size 1
```

Output:

```
{  
  "TableNames": [  
    "Forum",  
    "ProductCatalog",  
    "Reply",  
    "Thread"  
  ]  
}
```

Untuk informasi selengkapnya, lihat [Daftar Nama Tabel](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 3: Untuk membatasi jumlah item yang dikembalikan

Contoh berikut membatasi jumlah item yang dikembalikan ke 2. Respons mencakup NextToken nilai yang dapat digunakan untuk mengambil halaman hasil berikutnya.

```
aws dynamodb list-tables \  
  --max-items 2
```

Output:

```
{  
  "TableNames": [  
    "Forum",  
    "ProductCatalog"  
  ],  
  "NextToken":  
  "abCDeFGhiJKlMnOPqrSTuvwxYZ1aBCdEFghijK7LM51n0ppqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9"  
}
```

Untuk informasi selengkapnya, lihat [Daftar Nama Tabel](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 4: Untuk mengambil halaman berikutnya dari hasil

Perintah berikut menggunakan NextToken nilai dari panggilan sebelumnya ke `list-tables` perintah untuk mengambil halaman lain dari hasil. Karena respons dalam kasus ini tidak termasuk NextToken nilai, kami tahu bahwa kami telah mencapai akhir hasil.

```
aws dynamodb list-tables \  
  --starting-token  
  abCDeFGhiJKlMnOPqrSTuvwxYZ1aBCdEFghijK7LM51n0pqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9
```

Output:

```
{  
  "TableNames": [  
    "Reply",  
    "Thread"  
  ]  
}
```

Untuk informasi selengkapnya, lihat [Daftar Nama Tabel](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [ListTables](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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() ([]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(context.TODO())
        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
}
```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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!");
}
}
```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
suspend fun listAllTables() {
    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        val response = ddb.listTables(ListTablesRequest {})
        response.tableNames?.forEach { tableName ->
            println("Table name is $tableName")
        }
    }
}
```

- Untuk detail API, lihat [ListTables](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
public function listTables($exclusiveStartTableName = "", $limit = 100)
{
    $this->dynamoDbClient->listTables([
        'ExclusiveStartTableName' => $exclusiveStartTableName,
        'Limit' => $limit,
    ]);
}
```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

    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
```

- Untuk detail API, lihat [ListTables](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Tentukan apakah tabel ada.

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

```

- Untuk detail API, lihat [ListTables](#) di Referensi AWS SDK for Ruby API.

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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::

```

Tentukan apakah tabel ada.

```

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()),
}
}
```

- Untuk detail API, lihat [ListTables](#) referensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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( ).
    ENDLLOOP.
    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.
```

- Untuk detail API, lihat [ListTables](#) di AWS SDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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] {
    var tableList: [String] = []
    var lastEvaluated: String? = nil

    // Iterate over the list of tables, 25 at a time, until we have the
    // names of every table. Add each group to the `tableList` array.
    // Iteration is complete when `output.lastEvaluatedTableName` is `nil`.

    repeat {
        let input = ListTablesInput(
            exclusiveStartTableName: lastEvaluated,
            limit: 25
        )
        let output = try await self.session.listTables(input: input)
        guard let tableNames = output.tableNames else {
            return tableList
        }
        tableList.append(contentsOf: tableNames)
        lastEvaluated = output.lastEvaluatedTableName
    } while lastEvaluated != nil
}
```



```
    return tableList
}
```

- Untuk detail API, lihat referensi [ListTables AWSSDK](#) untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menempatkan item dalam tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menempatkan item dalam tabel DynamoDB.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- [Mempercepat pembacaan dengan DAX](#)
- [Memulai tabel, item, dan kueri](#)

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <param name="newMovie">A Movie object containing information 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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
    Movie newMovie, string tableName)
    {
        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,
        };

        var response = await client.PutItemAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

#####
# 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 getopt 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 getopt "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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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  
}
```

```
}
```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#!/ 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 outcome.IsSuccess();
}
```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Contoh 1: Untuk menambahkan item ke tabel

`put-item` Contoh berikut menambahkan item baru ke MusicCollection tabel.

```
aws dynamodb put-item \  
    --table-name MusicCollection \  
    --item-key   
    --item-value
```

```
--item file://item.json \  
--return-consumed-capacity TOTAL \  
--return-item-collection-metrics SIZE
```

Isi dari item.json:

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

Output:

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

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk menimpa item secara kondisional dalam tabel

put-itemContoh berikut menimpa item yang ada dalam MusicCollection tabel hanya jika item yang ada memiliki AlbumTitle atribut dengan nilai Greatest Hits Perintah mengembalikan nilai item sebelumnya.

```
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
```

Isi dari item.json:

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

Isi dari names.json:

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

Isi dari values.json:

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

Output:

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

```
}
```

Jika kunci sudah ada, Anda akan melihat output berikut:

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

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [PutItem](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(movie Movie) error {
    item, err := attributevalue.MarshalMap(movie)
    if err != nil {
        panic(err)
    }
    _, err = basics.DynamoDbClient.PutItem(context.TODO(), &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
}

// 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"])
}
```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menempatkan item ke dalam tabel menggunakan [DynamoDbClient](#).

```
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);
        }
    }
}
```

```
    }  
  }  
}
```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [PutCommand](#).

```
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;  
};
```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Masukkan item ke dalam tabel.

```
// 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);
  }
});
```

Masukkan item ke dalam tabel menggunakan klien dokumen DynamoDB.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

```

- Untuk detail API, lihat [PutItem](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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);
    }

```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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

    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
```

- Untuk detail API, lihat [PutItem](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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
end
```

- Untuk detail API, lihat [PutItem](#) di Referensi AWS SDK for Ruby API.

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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,  
})  
}
```

- Untuk detail API, lihat [PutItem](#) referensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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.
```

- Untuk detail API, lihat [PutItem](#) di AWS SDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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 {
    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)
}

///
/// 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
}
```

- Untuk detail API, lihat referensi [PutItem AWSSDK](#) untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Kueri tabel DynamoDB menggunakan SDK AWS


Contoh kode berikut menunjukkan cara melakukan kueri tabel DynamoDB.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- [Mempercepat pembacaan dengan DAX](#)
- [Memulai tabel, item, dan kueri](#)

.NET

AWS SDK for .NET

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient
client, string tableName, int year)
{
    var movieTable = Table.LoadTable(client, tableName);
    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;

Search 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;
}
```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for .NET .

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#####
# 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 getopt 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 getopt "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}" ;;
        esac
    done
}

```

```
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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}

```

- Untuk detail API, lihat [Query](#) in Referensi Perintah AWS CLI .

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

//! 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;  
}
```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for C++ .

CLI

AWS CLI

Contoh 1: Untuk menanyakan tabel

queryContoh berikut query item dalam MusicCollection tabel. Tabel memiliki kunci hash-and-range utama (ArtistdanSongTitle), tetapi kueri ini hanya menentukan nilai kunci hash. Ini mengembalikan judul lagu oleh artis bernama “No One You Know”.

```
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
```

Isi dari expression-attributes.json:

```
{  
  ":v1": {"S": "No One You Know"}  
}
```

Output:

```
{  
  "Items": [  
    {  
      "SongTitle": {  
        "S": "Call Me Today"  
      },  
      "SongTitle": {  
        "S": "Scared of My Shadow"  
      }  
    }  
  ]  
}
```

```

    }
  }
],
"Count": 2,
"ScannedCount": 2,
"ConsumedCapacity": {
  "TableName": "MusicCollection",
  "CapacityUnits": 0.5
}
}

```

Untuk informasi selengkapnya, lihat [Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

Contoh 2: Untuk menanyakan tabel menggunakan pembacaan yang sangat konsisten dan melintasi indeks dalam urutan menurun

Contoh berikut melakukan kueri yang sama dengan contoh pertama, tetapi mengembalikan hasil dalam urutan terbalik dan menggunakan pembacaan yang sangat konsisten.

```

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

```

Isi dari `expression-attributes.json`:

```

{
  ":v1": {"S": "No One You Know"}
}

```

Output:

```

{
  "Items": [
    {
      "SongTitle": {
        "S": "Scared of My Shadow"
      }
    }
  ]
}

```

```

    }
  },
  {
    "SongTitle": {
      "S": "Call Me Today"
    }
  }
],
"Count": 2,
"ScannedCount": 2,
"ConsumedCapacity": {
  "TableName": "MusicCollection",
  "CapacityUnits": 1.0
}
}

```

Untuk informasi selengkapnya, lihat [Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

Contoh 3: Untuk menyaring hasil tertentu

Contoh berikut query MusicCollection tetapi mengecualikan hasil dengan nilai-nilai tertentu dalam atribut. AlbumTitle Perhatikan bahwa ini tidak mempengaruhi ScannedCount atau ConsumedCapacity, karena filter diterapkan setelah item telah dibaca.

```

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

```

Isi dari values.json:

```

{
  ":v1": {"S": "No One You Know"},
  ":v2": {"S": "Blue Sky Blues"},
  ":v3": {"S": "Greatest Hits"}
}

```

Isi dari names.json:

```
{
  "#n1": "Artist",
  "#n2": "AlbumTitle"
}
```

Output:

```
{
  "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
  }
}
```

Untuk informasi selengkapnya, lihat [Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

Contoh 4: Untuk mengambil hanya jumlah item

Contoh berikut mengambil hitungan item yang cocok dengan query, tetapi tidak mengambil salah satu item itu sendiri.

```
aws dynamodb query \
  --table-name MusicCollection \
  --select COUNT \
  --key-condition-expression "Artist = :v1" \
  --expression-attribute-values file://expression-attributes.json
```

Isi dari `expression-attributes.json`:

```
{
  ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
  "Count": 2,
  "ScannedCount": 2,
  "ConsumedCapacity": null
}
```

Untuk informasi selengkapnya, lihat [Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

Contoh 5: Untuk menanyakan indeks

Contoh berikut query indeks `AlbumTitleIndex` sekunder lokal. Query mengembalikan semua atribut dari tabel dasar yang telah diproyeksikan ke indeks sekunder lokal. Perhatikan bahwa saat menanyakan indeks sekunder lokal atau indeks sekunder global, Anda juga harus memberikan nama tabel dasar menggunakan `table-name` parameter.

```
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
```

Isi dari `expression-attributes.json`:

```
{
  ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
```

```
"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
    }
  }
}
```

Untuk informasi selengkapnya, lihat [Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

- Untuk detail API, lihat [Query](#) in Referensi Perintah AWS CLI .

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(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(context.TODO())
        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
}

// 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"])
}
```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for Go .

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kueri tabel dengan menggunakan [DynamoDbClient](#).

```
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;
    }
}
```

Mengkueri tabel menggunakan `DynamoDbClient` dan indeks sekunder.

```
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);
        }
    }
}
```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [QueryCommand](#).

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for JavaScript .

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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);
  }
});

```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for JavaScript .

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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 { region = "us-east-1" }.use { ddb ->
    val response = ddb.query(request)
    return response.count
}
}

```

- Untuk detail API, lihat [Query](#) di Referensi API SDK for Kotlin AWS .

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

$birthKey = [
    'Key' => [
        'year' => [
            'N' => "$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);
    }

```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for PHP .

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kueri item menggunakan ekspresi kondisi kunci.

```

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

    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"]
```

Kueri dan proyeksikan item untuk mengembalikan subset data.

```
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"]
```

- Untuk detail API, lihat [Query](#) di Referensi API SDK for Python (Boto3)AWS .

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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
  end
end
```

- Untuk detail API, lihat [Query](#) di Referensi API AWS SDK for Ruby .

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Temukan film yang dibuat pada tahun tertentu.

```
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![])
    }
}
```

- Untuk detail API, lihat [Query](#) di Referensi API SDK for Rust AWS .

SAP ABAP

SDK untuk SAP ABAP

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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.

```

- Untuk detail API, lihat [Query](#) di Referensi API SDK for SAP ABAP AWS .

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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] {
    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
    )
    let output = try await client.query(input: input)

    guard let items = output.items else {
        throw MoviesError.ItemNotFound
    }
}
```



```
    }

    // Convert the found movies into `Movie` objects and return an array
    // of them.

    var movieList: [Movie] = []
    for item in items {
        let movie = try Movie(withItem: item)
        movieList.append(movie)
    }
    return movieList
}
```

- Untuk detail API, lihat [Query](#) di Referensi API SDK for Swift AWS .

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Jalankan pernyataan PartiQL pada tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara menjalankan pernyataan PartiQL pada tabel DynamoDB.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- [Melakukan kueri tabel menggunakan PartiQL](#)

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Gunakan pernyataan INSERT untuk menambahkan item.

```
/// <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;
}
```

Gunakan pernyataan SELECT untuk mendapatkan item.

```
/// <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;
    }
}
```

Gunakan pernyataan SELECT untuk mendapatkan daftar item.

```
    /// <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;
}

```

Gunakan pernyataan UPDATE untuk memperbarui item.

```

/// <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;
}

```

Gunakan pernyataan DELETE untuk menghapus satu film.

```
/// <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;
}
```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for .NET API.

C++

SDK for C++

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Gunakan pernyataan INSERT untuk menambahkan item.

```
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;
        }
    }
}

```

Gunakan pernyataan SELECT untuk mendapatkan item.

```

// 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;
        }
    }
}
}

```

Gunakan pernyataan UPDATE untuk memperbarui item.

```

// 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;
}
}

```

Gunakan pernyataan DELETE untuk menghapus sebuah item.

```

// 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;
    }
}

```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for C++ API.

Go

SDK for Go V2

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Gunakan pernyataan INSERT untuk menambahkan item.

```

// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(movie Movie) error {
    params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
    movie.Info})
    if err != nil {
        panic(err)
    }
    _, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
    &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
}

```

```
}
```

Gunakan pernyataan SELECT untuk mendapatkan item.

```
// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
// table by
// title and year.
func (runner PartiQLRunner) GetMovie(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(context.TODO(),
        &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
}
```

Gunakan pernyataan SELECT untuk mendapatkan daftar item dan memproyeksikan hasilnya.

```
// 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(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(context.TODO(),
&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
}
```

Gunakan pernyataan UPDATE untuk memperbarui item.

```
// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(movie Movie, rating float64) error {
```

```
params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
movie.Year})
if err != nil {
    panic(err)
}
_, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
&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
}
```

Gunakan pernyataan DELETE untuk menghapus sebuah item.

```
// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(movie Movie) error {
    params, err := attributevalue.MarshalList([]interface{}{movie.Title,
movie.Year})
    if err != nil {
        panic(err)
    }
    _, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
&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
}
```

```
}
```

Tentukan struct `Movie` yang digunakan dalam contoh ini.

```
// 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"])
}
```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for Go API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat item menggunakan PartiQL.

```
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;
};
```

Dapatkan item menggunakan PartiQL.

```
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;
};
```

Perbarui item menggunakan PartiQL.

```
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;
};
```

Hapus item menggunakan PartiQL.


```
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;
};
```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for JavaScript API.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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,
    ]);
}
```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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
```

- Untuk detail API, lihat [ExecuteStatement](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Pilih satu item menggunakan PartiQL.

```
class DynamoDBPartiQLSingle

  attr_reader :dynamo_resource
  attr_reader :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
```

Perbarui satu item menggunakan PartiQL.

```
class DynamoDBPartiQLSingle

  attr_reader :dynamo_resource
  attr_reader :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
end
```

Tambahkan satu item menggunakan PartiQL.

```
class DynamoDBPartiQLSingle

  attr_reader :dynamo_resource
  attr_reader :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

```

Hapus satu item menggunakan PartiQL.

```

class DynamoDBPartiQLSingle

  attr_reader :dynamo_resource
  attr_reader :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]
    }
  end
end

```

```
}  
  @dynamodb.client.execute_statement(request)  
end
```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for Ruby API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Jalankan batch pernyataan PartiQL pada tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara menjalankan batch pernyataan PartiQL pada tabel DynamoDB.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- [Melakukan kueri pada tabel menggunakan batch pernyataan PartiQL](#)

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Gunakan batch pernyataan INSERT untuk menambahkan item.

```
/// <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!;
        }
    }
}

```

Gunakan batch pernyataan SELECT untuk mendapatkan item.

```

/// <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 =>
    {
        Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
    });
    return true;
}
else
{
    Console.WriteLine($"Couldn't find either {title1} or {title2}.");
    return false;
}
}

```

Gunakan batch pernyataan UPDATE untuk memperbarui item.

```

/// <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;
}

```

Gunakan batch DELETE untuk menghapus item.

```

/// <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;
}

```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for .NET API.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Gunakan batch pernyataan INSERT untuk menambahkan item.

```

// 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.AddEntry(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.AddEntry(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;
    }
}

```

Gunakan batch pernyataan SELECT untuk mendapatkan item.

```

// 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;
}
}
```

Gunakan batch pernyataan UPDATE untuk memperbarui item.

```
// 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;
    }
}

```

Gunakan batch DELETE untuk menghapus item.

```

// 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;
    }
}
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for C++ API.

Go

SDK for Go V2

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Gunakan batch pernyataan INSERT untuk menambahkan item.

```
// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
// to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(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(context.TODO(),
&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
}

```

Gunakan batch pernyataan SELECT untuk mendapatkan item.

```

// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
// from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(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(context.TODO(),
&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
}

```

Gunakan batch pernyataan UPDATE untuk memperbaiki item.

```

// 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(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(context.TODO(),
&dynamodb.BatchExecuteStatementInput{
        Statements: statementRequests,
    })
}

```

```
if err != nil {
    log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
}
return err
}
```

Gunakan batch DELETE untuk menghapus item.

```
// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
// movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(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(context.TODO(),
&dynamodb.BatchExecuteStatementInput{
    Statements: statementRequests,
})
    if err != nil {
        log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
    }
    return err
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
// 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"])
}
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for Go API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat batch item menggunakan PartiQL.

```
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;
};
```

Dapatkan batch item menggunakan PartiQL.

```
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;  
};
```

Perbarui batch item menggunakan PartiQL.

```
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;
};
```

Hapus batch item menggunakan PartiQL.

```
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;
};
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for JavaScript API.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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,
            ],
        ],
    ],
]);
}
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Baca batch item menggunakan PartiQL.

```
class DynamoDBPartiQLBatch

  attr_reader :dynamo_resource
  attr_reader :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
```

Hapus batch item menggunakan PartiQL.

```
class DynamoDBPartiQLBatch

  attr_reader :dynamo_resource
  attr_reader :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
end
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for Ruby API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Memindai tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara memindai tabel DynamoDB.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- [Mempercepat pembacaan dengan DAX](#)
- [Memulai tabel, item, dan kueri](#)

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
public static async Task<int> ScanTableAsync(
    AmazonDynamoDBClient client,
    string tableName,
    int startYear,
    int endYear)
{
    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 client.ScanAsync(request);
        foundCount += response.Items.Count;
        response.Items.ForEach(i => DisplayItem(i));
        request.ExclusiveStartKey = response.LastEvaluatedKey;
    }
    while (response.LastEvaluatedKey.Count > 0);
    return foundCount;
}

```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for .NET .

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

#####
# 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 getopt 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 getopt "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}" ;;
        esac
    done
}

```

```
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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}
```

- Untuk detail API, lihat [Scan](#) in Referensi Perintah AWS CLI .

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

//! 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);

    // Perform scan on table.
    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();
        if (!items.empty()) {
            std::cout << "Number of items retrieved from scan: " << items.size()
                << std::endl;
            // Iterate each item and print.
            for (const Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue> &itemMap: 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 item found in table: " << tableName << std::endl;
    }
}
else {
    std::cerr << "Failed to Scan items: " << outcome.GetError().GetMessage()
        << std::endl;
}

return outcome.IsSuccess();
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for C++ .

CLI

AWS CLI

Untuk memindai tabel

scanContoh berikut memindai seluruh MusicCollection tabel, dan kemudian mempersempit hasilnya menjadi lagu-lagu oleh artis "No One You Know". Untuk setiap item, hanya judul album dan judul lagu yang dikembalikan.

```
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
```

Isi dari expression-attribute-names.json:

```
{
  "#ST": "SongTitle",
  "#AT": "AlbumTitle"
}
```


Isi dari `expression-attribute-values.json`:

```
{
  ":a": {"S": "No One You Know"}
}
```

Output:

```
{
  "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
}
```

Untuk informasi selengkapnya, lihat [Bekerja dengan Pemindaian di DynamoDB di Panduan Pengembang Amazon DynamoDB](#).

- Untuk detail API, lihat [Scan](#) in Referensi Perintah AWS CLI .

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(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(context.TODO())
    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
}

// 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"])
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for Go .

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Memindai tabel Amazon [DynamoDbClient](#)DynamoDB menggunakan.

```
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);
    }
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [ScanCommand](#).

```
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;
};
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for JavaScript .

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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 + ")"
      );
    });
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for JavaScript .

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
suspend fun scanItems(tableNameVal: String) {
    val request = ScanRequest {
        tableName = tableNameVal
```



```

    }

    DynamoDbClient { 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]}")
            }
        }
    }
}
}
}

```

- Untuk detail API, lihat [Scan](#) di Referensi API SDK for Kotlin AWS .

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

$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);
    }
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for PHP .

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

    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
```

- Untuk detail API, lihat [Scan](#) di Referensi API SDK for Python (Boto3)AWS .

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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
  end
end
```

```
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
```

- Untuk detail API, lihat [Scan](#) di Referensi API AWS SDK for Ruby .

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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(())
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API SDK for Rust AWS .

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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.
  ENDLLOOP.
  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.

```

- Untuk detail API, lihat [Scan](#) di Referensi API SDK for SAP ABAP AWS .

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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] {
    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 output = try await client.scan(input: input)

    guard let items = output.items else {
        return movieList
    }

    // Build an array of `Movie` objects for the returned items.

    for item in items {
        let movie = try Movie(withItem: item)
        movieList.append(movie)
    }

    // Call this function recursively to continue collecting matching
    // movies, if necessary.

    if output.lastEvaluatedKey != nil {
        let movies = try await self.getMovies(firstYear: firstYear, lastYear:
lastYear,
                                           startKey: output.lastEvaluatedKey)
        movieList += movies
    }
    return movieList
}
```

- Untuk detail API, lihat [Scan](#) di Referensi API SDK for Swift AWS .

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Memperbarui item dalam tabel DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara memperbarui item dalam tabel DynamoDB.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- [Memulai tabel, item, dan kueri](#)

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <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 static async Task<bool> UpdateItemAsync(
        AmazonDynamoDBClient client,
        Movie newMovie,
        MovieInfo newInfo,
```

```
string tableName)
{
    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,
    };

    var response = await client.UpdateItemAsync(request);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#####
# 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 getopt 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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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  
}
```

```
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#!/ 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 outcome.IsSuccess();
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Contoh 1: Untuk memperbarui item dalam tabel

Contoh `update-item` berikut memperbarui item dalam tabel `MusicCollection`. Ia menambahkan atribut baru (`Year`) dan memodifikasi `AlbumTitle` atribut. Semua atribut dalam item, seperti yang muncul setelah pembaruan, dikembalikan sebagai respons.

```
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
```

Isi dari `key.json`:

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

Isi dari `expression-attribute-names.json`:

```
{  
  "#Y": "Year", "#AT": "AlbumTitle"  
}
```

Isi dari `expression-attribute-values.json`:

```
{  
  ":y": {"N": "2015"},  
  ":t": {"S": "Louder Than Ever"}  
}
```

Output:

```
{
  "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
  ]
}
```

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk memperbarui item secara kondisional

Contoh berikut memperbarui item dalam `MusicCollection` tabel, tetapi hanya jika item yang ada belum memiliki `Year` atribut.

```
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)"
```

Isi dari `key.json`:

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

Isi dari `expression-attribute-names.json`:

```
{  
  "#Y": "Year",  
  "#AT": "AlbumTitle"  
}
```

Isi dari `expression-attribute-values.json`:

```
{  
  ":y": {"N": "2015"},  
  ":t": {"S": "Louder Than Ever"}  
}
```

Jika item sudah memiliki `Year` atribut, DynamoDB mengembalikan output berikut.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem  
operation: The conditional request failed
```

Untuk informasi selengkapnya, lihat [Menulis Item](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(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(context.TODO(),
            &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
}

// 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"])
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Memperbarui item dalam tabel menggunakan [DynamoDbClient](#).

```
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!");
}
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [UpdateCommand](#).

```
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;
};
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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 { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- Untuk detail API, lihat [UpdateItem](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
    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
            ]
        ],
    ]);
}
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Perbarui item dengan menggunakan ekspresi pembaruan.

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

    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"]
```

Perbarui item dengan menggunakan ekspresi pembaruan yang menyertakan operasi aritmatika.

```
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"]
```

Perbarui item hanya jika memenuhi persyaratan tertentu.

```
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"]

```

- Untuk detail API, lihat [UpdateItem](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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
```

- Untuk detail API, lihat [UpdateItem](#) di Referensi AWS SDK for Ruby API.

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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.
```

- Untuk detail API, lihat [UpdateItem](#) di AWS SDK untuk referensi SAP ABAP API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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]? {
    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: String = "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
}
```

- Untuk detail API, lihat referensi [UpdateItem AWSSDK](#) untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menulis sekumpulan item DynamoDB menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara untuk menulis batch item DynamoDB.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- [Memulai tabel, item, dan kueri](#)

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menulis batch item ke tabel film.

```
/// <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 = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
```

```
        PropertyNameCaseInsensitive = true
    });

    // Now return the first 250 entries.
    return allMovies.GetRange(0, 250);
}

/// <summary>
/// Writes 250 items to the movie table.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <param name="movieFileName">A string containing the full path to
/// the JSON file containing movie data.</param>
/// <returns>A long integer value representing the number of movies
/// imported from the JSON file.</returns>
public static async Task<long> BatchWriteItemsAsync(
    AmazonDynamoDBClient client,
    string movieFileName)
{
    var movies = ImportMovies(movieFileName);
    if (movies is null)
    {
        Console.WriteLine("Couldn't find the JSON file with movie
data.");
        return 0;
    }

    var context = new DynamoDBContext(client);

    var movieBatch = context.CreateBatchWrite<Movie>();
    movieBatch.AddPutItems(movies);

    Console.WriteLine("Adding imported movies to the table.");
    await movieBatch.ExecuteAsync();

    return movies.Count;
}
```

- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS SDK for .NET API.

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
#####
# 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 getopt 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 getopt "i:h" option; do
        case "${option}" in
            i) item="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
        esac
    done
}
```

```

    \?)
    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
}

```

Fungsi utilitas yang digunakan dalam contoh ini.

```

#####
# 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
}
```

- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS CLI Perintah.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
//! 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 true;
}

//! 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::JsonValue> &requestsJson,

    Aws::Vector<Aws::DynamoDB::Model::WriteRequest> &writeRequests) {
    for (size_t i = 0; i < requestsJson.GetLength(); ++i) {
        const Aws::Utils::Json::JsonValue &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::JsonValue> requestsMap =
requestsEntry.GetAllObjects();

        for (const auto &request: requestsMap) {
            const Aws::String &requestType = request.first;
            const Aws::Utils::Json::JsonValue &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::JsonValue> objectsMap =
jsonView.GetAllObjects();
for (const auto &entry: objectsMap) {
    const Aws::String &attributeKey = entry.first;
    const Aws::Utils::Json::JsonValue &attributeJsonValue = entry.second;

    if (!attributeJsonValue.IsObject()) {
        std::cerr << "Error: attribute not an object "
            << attributeJsonValue.WriteReadable() << std::endl;
        return false;
    }

    attributes.emplace(attributeKey,

Aws::DynamoDB::Model::AttributeValue(attributeJsonValue));
}

return true;
}
```

- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS SDK for C++ API.

CLI

AWS CLI

Untuk menambahkan beberapa item ke tabel

`batch-write-item` Contoh berikut menambahkan tiga item baru ke `MusicCollection` tabel menggunakan batch tiga `PutItem` permintaan. Ini juga meminta informasi tentang jumlah unit kapasitas tulis yang dikonsumsi oleh operasi dan koleksi item apa pun yang dimodifikasi oleh operasi.

```
aws dynamodb batch-write-item \
  --request-items file://request-items.json \
  --return-consumed-capacity INDEXES \
  --return-item-collection-metrics SIZE
```

Isi dari `request-items.json`:

```
{
```

```

"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"}
      }
    }
  }
]
}

```

Output:

```

{
  "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
            }
        }
    }
]
```

Untuk informasi selengkapnya, lihat [Operasi Batch](#) di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS CLI Perintah.

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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(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(context.TODO(),
&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
}

// 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"])
}
```

- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menyisipkan banyak item ke dalam tabel dengan menggunakan klien layanan.

```
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);
        }
    }
}
```

Menyisipkan banyak item ke dalam tabel menggunakan klien yang disempurnakan.

```
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);
    }
}
}
```

- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat [BatchWrite](#).

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

- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS SDK for JavaScript API.

SDK untuk JavaScript (v2)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat [Panduan Developer AWS SDK for JavaScript](#).
- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS SDK for JavaScript API.

PHP

SDK for PHP

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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;
        }
    }
}
```

- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

    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
```

- Untuk detail API, lihat [BatchWriteItem](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :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
end
```

- Untuk detail API, lihat [BatchWriteItem](#) di Referensi AWS SDK for Ruby API.

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
/// 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 {
    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)
    }
}
```

- Untuk detail API, lihat referensi [BatchWriteItem AWS SDK](#) untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Skenario untuk DynamoDB menggunakan AWS SDK

Contoh kode berikut menunjukkan cara menerapkan skenario umum di DynamoDB dengan SDK. AWS Skenario-skenario ini menunjukkan cara menyelesaikan tugas tertentu dengan memanggil beberapa fungsi di dalam DynamoDB. Setiap skenario menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode.

Contoh-contoh

- [Mempercepat pembacaan DynamoDB dengan DAX menggunakan SDK AWS](#)
- [Memulai tabel, item, dan kueri DynamoDB menggunakan SDK AWS](#)
- [Kueri tabel DynamoDB dengan menggunakan kumpulan pernyataan PartiQL dan SDK AWS](#)
- [Kueri tabel DynamoDB menggunakan PartiQL dan SDK AWS](#)
- [Menggunakan model dokumen untuk DynamoDB menggunakan SDK AWS](#)
- [Menggunakan model persistensi objek tingkat tinggi untuk DynamoDB menggunakan SDK AWS](#)

Mempercepat pembacaan DynamoDB dengan DAX menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat dan tulis data ke tabel dengan klien DAX dan SDK.
- Dapatkan, kueri, dan pindai tabel dengan kedua klien tersebut dan bandingkan performanya.

Untuk informasi selengkapnya, lihat [Melakukan pengembangan dengan Klien DynamoDB Accelerator](#).

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat tabel dengan klien DAX atau Boto3.

```
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"},
    ],
    "ProvisionedThroughput": {"ReadCapacityUnits": 10, "WriteCapacityUnits":
10},
    }
    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.")

```

Tulis data uji ke tabel tersebut.

```

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)

```

Dapatkan item untuk sejumlah iterasi untuk klien DAX maupun klien Boto3 dan laporkan waktu yang dihabiskan untuk setiap klien.

```

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}."
)

```

Kueri tabel untuk sejumlah iterasi untuk klien DAX maupun klien Boto3 dan laporkan waktu yang dihabiskan untuk setiap klien.

```

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}."
)
```

Pindai tabel untuk sejumlah iterasi untuk klien DAX maupun klien Boto3 dan laporkan waktu yang dihabiskan untuk setiap klien.

```
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}."
)
```

Hapus tabel tersebut.

```
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!")
```

- Untuk detail API, lihat topik berikut di Referensi API SDK AWS untuk Python (Boto3).
 - [CreateTable](#)
 - [DeleteTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Memulai tabel, item, dan kueri DynamoDB menggunakan SDK AWS

Contoh-contoh kode berikut menunjukkan cara:

- Buat tabel yang dapat menyimpan data film.
- Masukkan, dapatkan, dan perbarui satu film dalam tabel tersebut.
- Tulis data film ke tabel dari file JSON sampel.
- Kueri untuk film yang dirilis pada tahun tertentu.
- Pindai film yang dirilis dalam suatu rentang tahun.
- Hapus film dari tabel, lalu hapus tabel tersebut.

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// This example application performs the following basic Amazon DynamoDB
// functions:
//
// CreateTableAsync
// PutItemAsync
// UpdateItemAsync
// BatchWriteItemAsync
// GetItemAsync
// DeleteItemAsync
// Query
// Scan
// DeleteItemAsync
//
using Amazon.DynamoDBv2;
using DynamoDB_Actions;

public class DynamoDB_Basics
{
    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();

        var tableName = "movie_table";

        // Relative path to moviedata.json in the local repository.
        var movieFileName = @"..\..\..\..\..\resources\sample_files
\movies.json";

        DisplayInstructions();
    }
}
```

```
// Create a new table and wait for it to be active.
Console.WriteLine($"Creating the new table: {tableName}");

var success = await DynamoDbMethods.CreateMovieTableAsync(client,
tableName);

if (success)
{
    Console.WriteLine($"\\nTable: {tableName} successfully created.");
}
else
{
    Console.WriteLine($"\\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 DynamoDbMethods.PutItemAsync(client, 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 DynamoDbMethods.UpdateItemAsync(client, 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 DynamoDbMethods.BatchWriteItemsAsync(client,
movieFileName);
    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 DynamoDbMethods.GetItemAsync(client, lookupMovie,
tableName);
    if (item.Count > 0)
    {
        DynamoDbMethods.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 DynamoDbMethods.DeleteItemAsync(client, 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($"Movies released in {findYear}");
    var queryCount = await DynamoDbMethods.QueryMoviesAsync(client,
tableName, findYear);
    Console.WriteLine($"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 DynamoDbMethods.ScanTableAsync(client, tableName,
startYear, endYear);
    Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

    WaitForEnter();
```

```
// Delete the table.
success = await DynamoDbMethods.DeleteTableAsync(client, tableName);

if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
else
{
    Console.WriteLine($"Could not delete {tableName}");
}

Console.WriteLine("The DynamoDB Basics example application is done.");

WaitForEnter();
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
private static void DisplayInstructions()
{
    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()
    {
        Console.WriteLine("\nPress <Enter> to continue.");
        Console.WriteLine(SepBar);
        _ = Console.ReadLine();
    }
}
```

Membuat tabel yang akan berisi data film.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
client, string tableName)
    {
        var response = await client.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,
        },
    },
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = 5,
        WriteCapacityUnits = 5,
    },
});

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

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

TableStatus status;

int sleepDuration = 2000;

do
{
    System.Threading.Thread.Sleep(sleepDuration);

    var describeTableResponse = await
client.DescribeTableAsync(request);
```

```

        status = describeTableResponse.Table.TableStatus;

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

    return status == TableStatus.ACTIVE;
}

```

Menambahkan satu film ke tabel.

```

    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
Movie newMovie, string tableName)
    {
        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,
        };

        var response = await client.PutItemAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```


Memperbarui satu item dalam tabel.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <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 static async Task<bool> UpdateItemAsync(
        AmazonDynamoDBClient client,
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        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,
    };

    var response = await client.UpdateItemAsync(request);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

Mengambil satu item dari tabel film.

```

    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <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 static async Task<Dictionary<string, AttributeValue>>
    GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
    {
        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 client.GetItemAsync(request);
return response.Item;
}
```

Menulis batch item ke tabel film.

```
/// <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 = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

    // Now return the first 250 entries.
    return allMovies.GetRange(0, 250);
}

/// <summary>
/// Writes 250 items to the movie table.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <param name="movieFileName">A string containing the full path to
/// the JSON file containing movie data.</param>
```

```
/// <returns>A long integer value representing the number of movies
/// imported from the JSON file.</returns>
public static async Task<long> BatchWriteItemsAsync(
    AmazonDynamoDBClient client,
    string movieFileName)
{
    var movies = ImportMovies(movieFileName);
    if (movies is null)
    {
        Console.WriteLine("Couldn't find the JSON file with movie
data.");
        return 0;
    }

    var context = new DynamoDBContext(client);

    var movieBatch = context.CreateBatchWrite<Movie>();
    movieBatch.AddPutItems(movies);

    Console.WriteLine("Adding imported movies to the table.");
    await movieBatch.ExecuteAsync();

    return movies.Count;
}
```

Menghapus satu item dari tabel.

```
/// <summary>
/// Deletes a single item from a DynamoDB table.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <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 static async Task<bool> DeleteItemAsync(
    AmazonDynamoDBClient client,
    string tableName,
```

```
        Movie movieToDelete)
    {
        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,
        };

        var response = await client.DeleteItemAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

Melakukan kueri tabel untuk film yang dirilis pada tahun tertentu.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient
client, string tableName, int year)
{
    var movieTable = Table.LoadTable(client, tableName);
    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;

    Search 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;
}
```

Memindai tabel untuk film yang dirilis dalam suatu rentang tahun.

```
public static async Task<int> ScanTableAsync(
    AmazonDynamoDBClient client,
    string tableName,
    int startYear,
    int endYear)
{
    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 client.ScanAsync(request);
        foundCount += response.Items.Count;
        response.Items.ForEach(i => DisplayItem(i));
        request.ExclusiveStartKey = response.LastEvaluatedKey;
    }
    while (response.LastEvaluatedKey.Count > 0);
    return foundCount;
}

```

Menghapus tabel film.

```

    public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient
client, string tableName)
    {
        var request = new DeleteTableRequest
        {
            TableName = tableName,

```

```
};

var response = await client.DeleteTableAsync(request);
if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
{
    Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
    return true;
}
else
{
    Console.WriteLine("Could not delete table.");
    return false;
}
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for .NET .
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)
 - [UpdateItem](#)

Bash

AWS CLI dengan skrip Bash

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Skenario mulai DynamoDB.

```
#####
# 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

local provisioned_throughput="ReadCapacityUnits=5,WriteCapacityUnits=5"

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 "$key_schema_json_file" -p "$provisioned_throughput" 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
}

```

Fungsi DynamoDB yang digunakan dalam skenario ini.

```

#####
# 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.
#     -p provisioned_throughput -- Provisioned throughput settings for the
#     table.
#
# Returns:
#     0 - If successful.
#     1 - If it fails.
#####

```

```
function dynamodb_create_table() {
    local table_name attribute_definitions key_schema provisioned_throughput
    response
    local option OPTARG # Required to use getopt command in a function.

#####
# Function usage explanation
#####
function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table."
    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 " -p provisioned_throughput -- Provisioned throughput settings for the
table."
    echo ""
}

# Retrieve the calling parameters.
while getopt "n:a:k:p:h" option; do
    case "${option}" in
        n) table_name="${OPTARG}" ;;
        a) attribute_definitions="${OPTARG}" ;;
        k) key_schema="${OPTARG}" ;;
        p) provisioned_throughput="${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
fi
```



```
    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

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

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

response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"${attribute_definitions}" \
    --key-schema file://"${key_schema}" \
    --provisioned-throughput "$provisioned_throughput")

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 getopt 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 getopt "n:h" option; do
        case "${option}" in
            n) table_name="${OPTARG}" ;;
            h)
                usage
                return 0
                ;;
            \?)
                echo "Invalid parameter"
                usage
                return 1
        esac
    done
}

```

```
;;
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 getopt 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 getopt "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 getopt 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 getopt 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 getopt "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 getopt 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 getopt 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 getopt "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 getopt 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 getopt 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 getopt "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
}

```

```

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 getopt 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
}
```

Fungsi utilitas yang digunakan dalam skenario ini.

```
#####
# 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
}
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)
 - [UpdateItem](#)

C++

SDK for C++

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
{
    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::JsonValue> 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 clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    // 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;  
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for C++ .
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)
 - [UpdateItem](#)

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Jalankan skenario interaktif untuk membuat tabel dan melakukan tindakan pada tabel tersebut.

```
// 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(
    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()
    if err != nil {
        panic(err)
    }
    if !exists {
        log.Printf("Creating table %v...\n", tableName)
        _, err = tableBasics.CreateMovieTable()
        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.IAnswerValidator{demotools.NotEmpty{}})
customMovie.Year = questioner.AskInt("What year was it released?",
    []demotools.IAnswerValidator{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.IAnswerValidator{
        demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10}})
customMovie.Info["plot"] = questioner.Ask("What's the plot? ",
    []demotools.IAnswerValidator{demotools.NotEmpty{}})
err = tableBasics.AddMovie(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.IAnswerValidator{
        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.IAnswerValidator{demotools.NotEmpty{}})
attributes, err := tableBasics.UpdateMovie(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(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.IAnswerValidator{
        demotools.InIntRange{Lower: 1, Upper: show}},
    )
movie, err := tableBasics.GetMovie(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.IAnswerValidator{demotools.InIntRange{Lower: 1972, Upper: 2018}},
    )
releases, err := tableBasics.Query(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.IAnswerValidator{
    demotools.InIntRange{Lower: 1972, Upper: 2018}})
endYear := questioner.AskInt("Enter another year: ",
[]demotools.IAnswerValidator{
    demotools.InIntRange{Lower: 1972, Upper: 2018}})
releases, err = tableBasics.Scan(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()
    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(customMovie)
}
if err == nil {
    log.Printf("Deleted %v.\n", customMovie.Title)
}

if questioner.AskBool("Delete the table, too? (y/n)", "y") {
    err = tableBasics.DeleteTable()
} 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))
}
```

Tentukan struct `Movie` yang digunakan dalam contoh ini.

```
// 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"])
}
```

Buat struct dan metode yang memanggil tindakan DynamoDB.

```
// 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() (bool, error) {
    exists := true
    _, err := basics.DynamoDbClient.DescribeTable(
        context.TODO(), &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() (*types.TableDescription, error) {
    var tableDesc *types.TableDescription
    table, err := basics.DynamoDbClient.CreateTable(context.TODO(),
        &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),
            ProvisionedThroughput: &types.ProvisionedThroughput{
                ReadCapacityUnits:  aws.Int64(10),
                WriteCapacityUnits: aws.Int64(10),
            },
        })
    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(context.TODO(), &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
    }
    return tableDesc, err
}
```

```
}

// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables() ([]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(context.TODO())
        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(movie Movie) error {
    item, err := attributevalue.MarshalMap(movie)
    if err != nil {
        panic(err)
    }
    _, err = basics.DynamoDbClient.PutItem(context.TODO(), &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(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(context.TODO(),
        &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(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(context.TODO(),
&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(title string, year int) (Movie, error) {
```

```
movie := Movie{Title: title, Year: year}
response, err := basics.DynamoDbClient.GetItem(context.TODO(),
&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(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(context.TODO())
            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(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(context.TODO())
        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(movie Movie) error {
    _, err := basics.DynamoDbClient.DeleteItem(context.TODO(),
        &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() error {
    _, err := basics.DynamoDbClient.DeleteTable(context.TODO(),
        &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
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for Go .
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)
 - [UpdateItem](#)

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat tabel DynamoDB.

```
// 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)
    .provisionedThroughput(ProvisionedThroughput.builder()
        .readCapacityUnits(10L)
        .writeCapacityUnits(10L)
        .build())
    .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.waitForTableExists(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);
    }
}

```

Buat fungsi pembantu untuk mengunduh dan mengekstrak file JSON sampel.

```

// 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++;
    }
}
```

Dapatkan item dari tabel.

```
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);
    }
}
```

Contoh lengkap.

```
/**
 * 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
 *
 * This Java example performs these tasks:
 *
 * 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 {
        final String usage = ""

            Usage:
            <fileName>

            Where:
```

```
        fileName - The path to the moviedata.json file that you can
download from the Amazon DynamoDB Developer Guide.
```

```
        """;

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

    String tableName = "Movies";
    String fileName = args[0];
    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)
    .provisionedThroughput(ProvisionedThroughput.builder()
        .readCapacityUnits(10L)
        .writeCapacityUnits(10L)
        .build())
    .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);
    }
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for Java 2.x .
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)
 - [UpdateItem](#)

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
import { readFileSync } from "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.");
};
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for JavaScript .
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)
 - [UpdateItem](#)

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat tabel DynamoDB.

```
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 provisionedVal = ProvisionedThroughput {
        readCapacityUnits = 10
        writeCapacityUnits = 10
    }

    val request = CreateTableRequest {
        attributeDefinitions = listOf(attDef, attDef1)
        keySchema = listOf(keySchemaVal, keySchemaVal1)
        provisionedThroughput = provisionedVal
        tableName = tableNameVal
    }
}
```

```

    }

    DynamoDbClient { 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}")
    }
}

```

Buat fungsi pembantu untuk mengunduh dan mengekstrak file JSON sampel.

```

// 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 { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}
```

Dapatkan item dari tabel.

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

Contoh lengkap.

```
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <fileName>

        Where:
            fileName - The path to the moviedata.json you can download from the
Amazon DynamoDB Developer Guide.
        """

    if (args.size != 1) {
        println(usage)
        exitProcess(1)
    }

    // Get the moviedata.json from the Amazon DynamoDB Developer Guide.
    val tableName = "Movies"
    val fileName = args[0]
    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.")
    deleteIssuesTable(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 provisionedVal = ProvisionedThroughput {
    readCapacityUnits = 10
    writeCapacityUnits = 10
}

val request = CreateTableRequest {
    attributeDefinitions = listOf(attDef, attDef1)
    keySchema = listOf(keySchemaVal, keySchemaVal1)
    provisionedThroughput = provisionedVal
    tableName = tableNameVal
}

DynamoDbClient { 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 { 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 { 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 { 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 { 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 { 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]}")  
            }  
        }  
    }  
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK For Kotlin.
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)
 - [UpdateItem](#)

PHP

SDK for PHP

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
namespace DynamoDb\Basics;

use Aws\DynamoDb\Marshaller;
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, $key);
echo "\n\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}. \n\n";
echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n\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\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\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);
    }
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for PHP .
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)
 - [UpdateItem](#)

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat kelas yang merangkum tabel DynamoDB.

```
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."""

    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
                {"AttributeName": "title", "KeyType": "RANGE"}, # Sort key
            ],
            AttributeDefinitions=[
                {"AttributeName": "year", "AttributeType": "N"},
                {"AttributeName": "title", "AttributeType": "S"},
            ],
            ProvisionedThroughput={
                "ReadCapacityUnits": 10,
                "WriteCapacityUnits": 10,
            },
        )
        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
```

Buat fungsi pembantu untuk mengunduh dan mengekstrak file JSON sampel.

```
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]
```

Jalankan skenario interaktif untuk membuat tabel dan melakukan tindakan pada tabel tersebut.

```
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}")
```

Skenario ini menggunakan kelas pembantu berikut untuk mengajukan pertanyaan pada prompt perintah.

```
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
```

- Untuk detail API, lihat topik berikut di Referensi API SDK AWS untuk Python (Boto3).

- [BatchWriteItem](#)
- [CreateTable](#)
- [DeleteItem](#)
- [DeleteTable](#)
- [DescribeTable](#)
- [GetItem](#)
- [PutItem](#)
- [Query](#)
- [Scan](#)
- [UpdateItem](#)

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat kelas yang merangkum tabel DynamoDB.

```
# 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"}
    ],
    provisioned_throughput: {read_capacity_units: 10, write_capacity_units:
10})
    @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
end

```

Buat fungsi pembantu untuk mengunduh dan mengekstrak file JSON sampel.

```

# 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

```

```
end
```

Jalankan skenario interaktif untuk membuat tabel dan melakukan tindakan pada tabel tersebut.

```
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 if !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
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for Ruby .
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)

- [PutItem](#)
- [Query](#)
- [Scan](#)
- [UpdateItem](#)

SAP ABAP

SDK untuk SAP ABAP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
" 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 =
'Speed' ) ) )
                ( 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'.
    ENDRTRY.

" 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'.
    ENDRTRY.

" 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.
```


- Untuk mengetahui hal detail mengenai API, silakan lihat topik-topik berikut di referensi API SDK AWS untuk ABAP SAP.
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)
 - [UpdateItem](#)

Swift

SDK for Swift

Note

Ini adalah dokumentasi prarilis untuk SDK dalam rilis pratinjau. Dokumentasi ini dapat berubah.

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kelas Swift yang menangani panggilan DynamoDB ke SDK untuk Swift.

```
import Foundation
import AWSDynamoDB

/// 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? = nil
    let tableName: String

    /// Create an object representing a movie table in an Amazon DynamoDB
    /// database.
    ///
    /// - Parameters:
    ///   - region: The 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 = "us-east-2", tableName: String) async throws {
    ddbClient = try DynamoDBClient(region: region)
    self.tableName = tableName

    try await self.createTable()
}

///
/// Create a movie table in the Amazon DynamoDB data store.
///
private func createTable() async throws {
    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),
        ],
        keySchema: [
            DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
            DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
        ],
        provisionedThroughput: DynamoDBClientTypes.ProvisionedThroughput(
            readCapacityUnits: 10,
            writeCapacityUnits: 10
        ),
        tableName: self.tableName
    )
    let output = try await client.createTable(input: input)
    if output.tableDescription == nil {
        throw MoviesError.TableNotFound
    }
}

/// Check to see if the table exists online yet.
```

```
///
/// - Returns: `true` if the table exists, or `false` if not.
///
func tableExists() async throws -> Bool {
    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)
}

///
/// Waits for the table to exist and for its status to be active.
///
func awaitTableActive() async throws {
    while (try await tableExists() == false) {
        Thread.sleep(forTimeInterval: 0.25)
    }

    while (try await getTableStatus() != .active) {
        Thread.sleep(forTimeInterval: 0.25)
    }
}

///
/// Deletes the table from Amazon DynamoDB.
///
func deleteTable() async throws {
    guard let client = self.ddbClient else {
        throw MoviesError.UninitializedClient
    }

    let input = DeleteTableInput(
        tableName: self.tableName
    )
    _ = try await client.deleteTable(input: input)
```

```
}

/// Get the table's status.
///
/// - Returns: The table status, as defined by the
///   `DynamoDBClientTypes.TableStatus` enum.
///
func getTableStatus() async throws -> DynamoDBClientTypes.TableStatus {
    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
}

/// 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 {
    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)
    }
}

/// 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 {
    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)
}

/// 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 {
    let movie = Movie(title: title, year: year, rating: rating, plot: plot)
    try await self.add(movie: movie)
}

/// 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 {
    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
}

/// 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] {
    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
    )
    let output = try await client.query(input: input)

    guard let items = output.items else {
        throw MoviesError.ItemNotFound
    }
}
```

```
// Convert the found movies into `Movie` objects and return an array
// of them.

var movieList: [Movie] = []
for item in items {
    let movie = try Movie(withItem: item)
    movieList.append(movie)
}
return movieList
}

/// 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] {
    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 output = try await client.scan(input: input)

guard let items = output.items else {
    return movieList
}

// Build an array of `Movie` objects for the returned items.

for item in items {
    let movie = try Movie(withItem: item)
    movieList.append(movie)
}

// Call this function recursively to continue collecting matching
// movies, if necessary.

if output.lastEvaluatedKey != nil {
    let movies = try await self.getMovies(firstYear: firstYear, lastYear:
lastYear,
                                        startKey: output.lastEvaluatedKey)
    movieList += movies
}
return movieList
}

/// 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]? {
    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: String = "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
}
```

```

    }

    /// 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 {
        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)
    }
}

```

Struktur yang digunakan oleh MovieTable kelas untuk mewakili film.

```

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
}
}
```

Sebuah program yang menggunakan MovieTable kelas untuk mengakses database DynamoDB.

```
import Foundation
import ArgumentParser
import AWSDynamoDB
import ClientRuntime

@testable import MovieList

struct ExampleCommand: ParsableCommand {
    @Argument(help: "The path of the sample movie data JSON file.")
    var jsonPath: String = "../../../../../resources/sample_files/movies.json"

    @Option(help: "The AWS Region to run AWS API calls in.")
    var awsRegion = "us-east-2"

    @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!")
SDKLoggingSystem.initialize(logLevel: .error)

//=====
// 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))"
//let tableName = String.uniqueName(withPrefix: "ddb-movies-sample",
maxDigits: 8)

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)
        }
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API SDK AWS untuk Swift.
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Query](#)
 - [Scan](#)
 - [UpdateItem](#)

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Kueri tabel DynamoDB dengan menggunakan kumpulan pernyataan PartiQL dan SDK AWS

Contoh-contoh kode berikut menunjukkan cara:

- Dapatkan batch item dengan menjalankan beberapa pernyataan SELECT.
- Tambahkan batch item dengan menjalankan beberapa pernyataan INSERT.
- Perbarui batch item dengan menjalankan beberapa pernyataan UPDATE.
- Hapus batch item dengan menjalankan beberapa pernyataan DELETE.

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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 = "moviedata.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 =>
        {
            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;
}

```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for .NET API.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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.AddEntry(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.AddEntry(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.GetResponse();

        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 clientConfiguration: AWS client configuration.
 \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::Client::ClientConfiguration
                                       &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    // 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;
}
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for C++ API.

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Jalankan skenario yang membuat tabel dan menjalankan batch kueri PartiQL.

```
// 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(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()
if err != nil {
    panic(err)
}
if !exists {
    log.Printf("Creating table %v...\n", tableName)
    _, err = tableBasics.CreateMovieTable()
    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(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(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(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(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(customMovies)
if err == nil {
    log.Printf("Deleted %v movies.\n", len(customMovies))
}

err = tableBasics.DeleteTable()
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))
}
```

Tentukan struct `Movie` yang digunakan dalam contoh ini.

```
// 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"])
}
```

Buat struct dan metode yang menjalankan pernyataan PartiQL.

```
// 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(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(context.TODO(),
&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(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(context.TODO(),
    &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(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(context.TODO(),
&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(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(context.TODO(),
&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(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(context.TODO(),
&dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
})
if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
}
return err
}
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
public class ScenarioPartiQLBatch {
    public static void main(String[] args) throws IOException {
        String tableName = "MoviesPartiQLBatch";
        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);

        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 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)

.provisionedThroughput(ProvisionedThroughput.builder()
            .readCapacityUnits(new Long(10))
            .writeCapacityUnits(new Long(10))
            .build())
            .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(String.valueOf("2022"))  
        .build();  
  
    AttributeValue att2 = AttributeValue.builder()  
        .s("My Movie 1")  
        .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(String.valueOf("2022"))  
        .build();  
  
    AttributeValue attMovie2A = AttributeValue.builder()  
        .s("My Movie 2")  
        .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);
    }
}
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Jalankan pernyataan PartiQL batch.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async () => {
  /**
   * 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 addItemStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/q1-
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.");
};
```


- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
suspend fun main() {
    val ddb = DynamoDbClient { region = "us-east-1" }
    val tableName = "MoviesPartiQLBatch"
    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 provisionedVal = ProvisionedThroughput {
    readCapacityUnits = 10
    writeCapacityUnits = 10
}

val request = CreateTableRequest {
    attributeDefinitions = listOf(attDef, attDef1)
    keySchema = listOf(keySchemaVal, keySchemaVal1)
    provisionedThroughput = provisionedVal
    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")
    }
}
```

```
}
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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,
            ],
        ],
    ]);
}

```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat kelas yang dapat menjalankan batch pernyataan PartiQL.

```

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
(PPOs) 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
```

Jalankan skenario yang membuat tabel dan menjalankan kueri PartiQL dalam batch.

```
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}")
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Jalankan skenario yang membuat tabel dan menjalankan batch kueri PartiQL.

```
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.")
if scaffold.exists?(table_name)
  scaffold.delete_table
end
end
```

- Untuk detail API, lihat [BatchExecuteStatement](#) di Referensi AWS SDK for Ruby API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Kueri tabel DynamoDB menggunakan PartiQL dan SDK AWS

Contoh-contoh kode berikut menunjukkan cara:

- Dapatkan item dengan menjalankan pernyataan SELECT.
- Tambahkan item dengan menjalankan pernyataan INSERT.
- Perbarui item dengan menjalankan pernyataan UPDATE.
- Hapus item dengan menjalankan pernyataan DELETE.

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
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;
    }
}
```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for .NET API.

C++

SDK for C++

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
// 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 clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    // 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;
}
```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for C++ API.

Go

SDK for Go V2

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Jalankan skenario yang membuat tabel dan menjalankan kueri PartiQL.

```
// 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(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()
if err != nil {
    panic(err)
}
if !exists {
    log.Printf("Creating table %v...\n", tableName)
    _, err = tableBasics.CreateMovieTable()
    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(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(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(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(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(customMovie)
if err == nil {
    log.Printf("Deleted %v.\n", customMovie.Title)
}

err = tableBasics.DeleteTable()
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))
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
// 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"])
}
```

Buat struct dan metode yang menjalankan pernyataan PartiQL.

```
// 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(movie Movie) error {
    params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
    movie.Info})
    if err != nil {
        panic(err)
    }
    _, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
    &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(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(context.TODO(),
    &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(movie Movie, rating float64) error {
    params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
    movie.Year})
    if err != nil {
        panic(err)
    }
    _, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
    &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(movie Movie) error {
    params, err := attributevalue.MarshalList([]interface{}{movie.Title,
    movie.Year})
    if err != nil {
        panic(err)
    }
}
```

```

_, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
&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
}

```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

public class ScenarioPartiQ {
    public static void main(String[] args) throws IOException {
        final String usage = ""

            Usage:
                <fileName>

            Where:
                fileName - The path to the moviedata.json file that you can
download from the Amazon DynamoDB Developer Guide.
                """;

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

```

```
        System.exit(1);
    }

    String fileName = args[0];
    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)
    .provisionedThroughput(ProvisionedThroughput.builder()
        .readCapacityUnits(new Long(10))
        .writeCapacityUnits(new Long(10))
        .build())
    .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());
}
}
```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Jalankan pernyataan PartiQL tunggal.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async () => {
  /**
   * 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.");
};
```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for JavaScript API.

Kotlin

SDK for Kotlin

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
        <fileName>

        Where:
            fileName - The path to the moviedata.json you can download from the
Amazon DynamoDB Developer Guide.
        """

    if (args.size != 1) {
        println(usage)
        exitProcess(1)
    }

    val ddb = DynamoDbClient { region = "us-east-1" }
    val tableName = "MoviesPartiQ"

    // Get the moviedata.json from the Amazon DynamoDB Developer Guide.
    val fileName = args[0]
    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 provisionedVal = ProvisionedThroughput {
        readCapacityUnits = 10
        writeCapacityUnits = 10
    }

    val request = CreateTableRequest {
        attributeDefinitions = listOf(attDef, attDef1)
        keySchema = listOf(keySchemaVal, keySchemaVal1)
        provisionedThroughput = provisionedVal
        tableName = tableNameVal
    }

    val response = ddb.createTable(request)
```

```
    ddb.waitForTableExists { // 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)
}
```

- Untuk detail API, lihat [ExecuteStatement](#) di AWS SDK untuk referensi API Kotlin.

PHP

SDK for PHP

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaller;
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,  
    ]);  
}
```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for PHP API.

Python

SDK for Python (Boto3)

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat kelas yang dapat menjalankan pernyataan PartiQL.

```
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
```

Jalankan skenario yang membuat tabel dan menjalankan kueri PartiQL.

```
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}")
```

- Untuk detail API, lihat [ExecuteStatement](#) di AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK for Ruby

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Jalankan skenario yang membuat tabel dan menjalankan kueri PartiQL.

```
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}".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 "\n\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.")
if scaffold.exists?(table_name)
  scaffold.delete_table
end
end
end

```

- Untuk detail API, lihat [ExecuteStatement](#) di Referensi AWS SDK for Ruby API.

Rust

SDK for Rust

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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");

let pt = ProvisionedThroughput::builder()
.read_capacity_units(10)
.write_capacity_units(5)
.build()
.expect("creating ProvisionedThroughput");

match client
.create_table()
.table_name(table)
.key_schema(ks)
.attribute_definitions(ad)
.provisioned_throughput(pt)
.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 {{
    "{}": ?,
    "account_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 "{}" WHERE "{}" = ?"#))
    .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(())
}
```

- Untuk detail API, lihat [ExecuteStatement](#) referensi AWS SDK for Rust API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menggunakan model dokumen untuk DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara melakukan operasi Create, Read, Update, and Delete (CRUD) dan batch menggunakan model dokumen untuk DynamoDB dan SDK. AWS

Untuk informasi lebih lanjut, lihat [Model dokumen](#).

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Lakukan operasi CRUD menggunakan model dokumen.

```
/// <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.ExpressionAttributeValue[":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);
    }
}
}
```

Lakukan operasi tulis batch menggunakan model dokumen.

```
/// <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();
}
}
```

Pindai tabel menggunakan model dokumen.

```
/// <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}");
}
}
```

Kueri dan pindai tabel menggunakan model dokumen.

```
/// <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}");
    }
}
```

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menggunakan model persistensi objek tingkat tinggi untuk DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara melakukan operasi Create, Read, Update, and Delete (CRUD) dan batch menggunakan model persistensi objek untuk DynamoDB dan SDK. AWS

Untuk informasi lebih lanjut, lihat [Model persistensi objek](#).

.NET

AWS SDK for .NET

Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di [Repositori Contoh Kode AWS](#).

Lakukan operasi CRUD menggunakan model persistensi objek tingkat tinggi.

```
/// <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");
        }
    }
}
```

Lakukan operasi tulis batch menggunakan model persistensi objek tingkat tinggi.

```
/// <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);
    }
}

```

Memetakan data arbitrer ke tabel menggunakan model persistensi objek tingkat tinggi.

```

/// <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);
    }
}
```

Kueri dan pindai tabel menggunakan model persistensi objek tingkat tinggi.

```
/// <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}");
    }
}
```

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Contoh lintas layanan untuk DynamoDB menggunakan SDK AWS

Contoh aplikasi berikut menggunakan AWS SDK untuk menggabungkan DynamoDB dengan yang lain. Layanan AWS Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan aplikasi.

Contoh-contoh

- [Membangun aplikasi untuk mengirimkan data ke tabel DynamoDB](#)
- [Membuat API REST Gateway API untuk melacak data COVID-19](#)
- [Membuat aplikasi messenger dengan Step Functions](#)
- [Membuat aplikasi manajemen aset foto yang memungkinkan pengguna mengelola foto menggunakan label](#)
- [Membuat aplikasi web untuk melacak data DynamoDB](#)
- [Membuat aplikasi obrolan websocket dengan API Gateway](#)
- [Mendeteksi APD dalam gambar dengan Amazon AWS Rekognition menggunakan SDK](#)
- [Menginvokasi fungsi Lambda dari browser](#)
- [Simpan EXIF dan informasi gambar lainnya menggunakan SDK AWS](#)
- [Menggunakan API Gateway untuk menginvokasi fungsi Lambda](#)
- [Menggunakan Step Functions untuk menginvokasi fungsi Lambda](#)
- [Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda](#)

Membangun aplikasi untuk mengirimkan data ke tabel DynamoDB

Contoh kode berikut menunjukkan cara membangun aplikasi yang mengirimkan data ke tabel Amazon DynamoDB dan memberi tahu Anda saat pengguna memperbarui tabel.

Java

SDK untuk Java 2.x

Menunjukkan cara membuat aplikasi web dinamis yang mengirimkan data menggunakan API Java Amazon DynamoDB dan mengirim pesan teks menggunakan API Java Amazon Simple Notification Service.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SNS

JavaScript

SDK untuk JavaScript (v3)

Contoh ini menunjukkan cara membangun aplikasi yang memungkinkan pengguna mengirimkan data ke tabel Amazon DynamoDB, dan mengirim pesan teks ke administrator menggunakan Amazon Simple Notification Service (Amazon SNS).

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Contoh ini juga tersedia di [panduan developer AWS SDK for JavaScript v3](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SNS

Kotlin

SDK untuk Kotlin

Menunjukkan cara membuat aplikasi Android native yang mengirimkan data menggunakan API Kotlin Amazon DynamoDB dan mengirim pesan teks menggunakan API Kotlin Amazon SNS.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SNS

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Membuat API REST Gateway API untuk melacak data COVID-19

Contoh kode berikut menunjukkan cara membuat API REST yang menyimulasikan sistem untuk melacak kasus COVID-19 harian di Amerika Serikat, menggunakan data fiksi.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS Chalice dengan membuat REST API tanpa server yang menggunakan Amazon API Gateway, AWS Lambda dan Amazon DynamoDB. AWS SDK for Python (Boto3) API REST menyimulasikan sistem untuk melacak kasus COVID-19 harian di Amerika Serikat, menggunakan data fiksi. Pelajari cara:

- Gunakan AWS Chalice untuk menentukan rute dalam fungsi Lambda yang dipanggil untuk menangani permintaan REST yang datang melalui API Gateway.
- Menggunakan fungsi Lambda untuk mengambil dan menyimpan data dalam tabel DynamoDB untuk melayani permintaan REST.
- Tentukan struktur tabel dan sumber daya peran keamanan dalam AWS CloudFormation template.
- Gunakan AWS Chalice dan CloudFormation untuk mengemas dan menyebarkan semua sumber daya yang diperlukan.
- Gunakan CloudFormation untuk membersihkan semua sumber daya yang dibuat.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- AWS CloudFormation
- DynamoDB
- Lambda

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Membuat aplikasi messenger dengan Step Functions

Contoh kode berikut menunjukkan cara membuat aplikasi AWS Step Functions messenger yang mengambil catatan pesan dari tabel database.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS SDK for Python (Boto3) with AWS Step Functions untuk membuat aplikasi messenger yang mengambil catatan pesan dari tabel Amazon DynamoDB dan mengirimkannya dengan Amazon Simple Queue Service (Amazon SQS). Mesin state terintegrasi dengan AWS Lambda fungsi untuk memindai database untuk pesan yang tidak terkirim.

- Buat mesin status yang mengambil dan memperbarui catatan pesan dari tabel Amazon DynamoDB.
- Perbarui definisi mesin status untuk mengirim pesan ke Amazon Simple Queue Service (Amazon SQS).
- Mulai dan hentikan berjalannya mesin status.
- Terhubung ke Lambda, DynamoDB, dan Amazon SQS dari mesin status menggunakan integrasi layanan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Membuat aplikasi manajemen aset foto yang memungkinkan pengguna mengelola foto menggunakan label

Contoh kode berikut ini menunjukkan cara membuat aplikasi nirserver yang memungkinkan pengguna mengelola foto menggunakan label.

.NET

AWS SDK for .NET

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat posting di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK untuk C++

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat posting di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK untuk Java 2.x

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat posting di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat posting di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK untuk Kotlin

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat posting di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK for PHP

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat posting di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK untuk Rust

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat posting di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Membuat aplikasi web untuk melacak data DynamoDB

Contoh-contoh kode berikut menunjukkan cara membuat aplikasi web yang melacak item kerja dalam tabel Amazon DynamoDB dan menggunakan Amazon Simple Email Service (Amazon SES) untuk mengirim laporan.

.NET

AWS SDK for .NET

Menunjukkan cara menggunakan Amazon DynamoDB .NET API untuk membuat aplikasi web dinamis yang melacak data kerja DynamoDB.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SES

Java

SDK untuk Java 2.x

Menunjukkan cara menggunakan Amazon DynamoDB API untuk membuat aplikasi web dinamis yang melacak data kerja DynamoDB.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SES

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan Amazon DynamoDB API untuk membuat aplikasi web dinamis yang melacak data kerja DynamoDB.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SES

Kotlin

SDK untuk Kotlin

Menunjukkan cara menggunakan Amazon DynamoDB API untuk membuat aplikasi web dinamis yang melacak data kerja DynamoDB.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SES

Python

SDK for Python (Boto3)

Menunjukkan cara menggunakan AWS SDK for Python (Boto3) untuk membuat layanan REST yang melacak item kerja di Amazon DynamoDB dan laporan email dengan menggunakan Amazon Simple Email Service (Amazon SES). Contoh ini menggunakan rangka kerja web Flask untuk menangani perutean HTTP dan terintegrasi dengan halaman web React untuk menyajikan aplikasi web yang berfungsi penuh.

- Bangun layanan Flask REST yang terintegrasi dengan. Layanan AWS
- Baca, tulis, dan perbarui item kerja yang disimpan dalam tabel DynamoDB.
- Gunakan Amazon SES untuk mengirim laporan email tentang item pekerjaan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di [Repositori Contoh AWS Kode](#) di. GitHub

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SES

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Membuat aplikasi obrolan websocket dengan API Gateway

Contoh kode berikut menunjukkan cara membuat aplikasi obrolan yang dilayani oleh API websocket yang dibangun di Amazon API Gateway.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS SDK for Python (Boto3) dengan Amazon API Gateway V2 untuk membuat API websocket yang terintegrasi dengan AWS Lambda dan Amazon DynamoDB.

- Buat API websocket yang dilayani oleh API Gateway.

- Tentukan penanganan Lambda yang menyimpan koneksi di DynamoDB dan memposting pesan ke peserta obrolan lainnya.
- Hubungkan ke aplikasi obrolan websocket dan kirim pesan dengan paket WebSocket.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Mendeteksi APD dalam gambar dengan Amazon AWS Rekognition menggunakan SDK

Contoh kode berikut menunjukkan cara membuat aplikasi yang menggunakan Amazon Rekognition untuk mendeteksi Alat Pelindung Diri (APD) dalam gambar.

Java

SDK for Java 2.x

Menunjukkan cara membuat AWS Lambda fungsi yang mendeteksi gambar dengan Alat Pelindung Diri.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon Rekognition
- Amazon S3

- Amazon SES

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan Amazon Rekognition dengan AWS SDK for JavaScript membuat aplikasi untuk mendeteksi alat pelindung diri (APD) pada gambar yang terletak di bucket Amazon Simple Storage Service (Amazon S3). Aplikasi tersebut menyimpan hasilnya ke tabel Amazon DynamoDB, dan mengirimkan notifikasi email kepada admin beserta hasilnya menggunakan Amazon Simple Email Service (Amazon SES).

Pelajari cara:

- Membuat pengguna yang tidak diautentikasi menggunakan Amazon Cognito.
- Menganalisis gambar untuk APD menggunakan Amazon Rekognition.
- Memverifikasi alamat email untuk Amazon SES.
- Memperbarui tabel DynamoDB dengan hasil.
- Mengirim notifikasi email menggunakan Amazon SES.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menginvokasi fungsi Lambda dari browser

Contoh kode berikut menunjukkan cara memanggil AWS Lambda fungsi dari browser.

JavaScript

SDK untuk JavaScript (v2)

Anda dapat membuat aplikasi berbasis browser yang menggunakan AWS Lambda fungsi untuk memperbarui tabel Amazon DynamoDB dengan pilihan pengguna.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda

SDK untuk JavaScript (v3)

Anda dapat membuat aplikasi berbasis browser yang menggunakan AWS Lambda fungsi untuk memperbarui tabel Amazon DynamoDB dengan pilihan pengguna. Aplikasi ini menggunakan AWS SDK for JavaScript v3.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Simpan EXIF dan informasi gambar lainnya menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara:

- Mendapatkan informasi EXIF dari file JPG, JPEG, atau PNG.
- Mengunggah file gambar ke bucket Amazon S3.
- Menggunakan Amazon Rekognition untuk mengidentifikasi tiga atribut teratas (label) dalam file.

- Menambahkan informasi EXIF dan label ke tabel Amazon DynamoDB di Wilayah.

Rust

SDK untuk Rust

Mendapatkan informasi EXIF dari file JPG, JPEG, atau PNG, mengunggah file gambar ke bucket Amazon S3, menggunakan Amazon Rekognition untuk mengidentifikasi tiga atribut teratas (label di Amazon Rekognition) dalam file, dan menambahkan EXIF dan informasi label ke tabel Amazon DynamoDB di Wilayah.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon Rekognition
- Amazon S3

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menggunakan API Gateway untuk menginvokasi fungsi Lambda

Contoh kode berikut menunjukkan cara membuat AWS Lambda fungsi yang dipanggil oleh Amazon API Gateway.

Java

SDK for Java 2.x

Menunjukkan cara membuat AWS Lambda fungsi dengan menggunakan Lambda Java runtime API. Contoh ini memanggil AWS layanan yang berbeda untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat fungsi Lambda yang diinvokasi oleh Amazon API Gateway yang memindai peringatan hari jadi kerja di tabel Amazon DynamoDB dan menggunakan Amazon Simple Notification Service (Amazon SNS) untuk mengirim pesan teks berisi ucapan selamat kepada karyawan Anda pada tanggal hari jadi kerja satu tahun mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara membuat AWS Lambda fungsi dengan menggunakan API JavaScript runtime Lambda. Contoh ini memanggil AWS layanan yang berbeda untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat fungsi Lambda yang diinvokasi oleh Amazon API Gateway yang memindai peringatan hari jadi kerja di tabel Amazon DynamoDB dan menggunakan Amazon Simple Notification Service (Amazon SNS) untuk mengirim pesan teks berisi ucapan selamat kepada karyawan Anda pada tanggal hari jadi kerja satu tahun mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Contoh ini juga tersedia di [panduan developer AWS SDK for JavaScript v3](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menggunakan Step Functions untuk menginvokasi fungsi Lambda

Contoh kode berikut menunjukkan cara membuat mesin AWS Step Functions status yang memanggil AWS Lambda fungsi secara berurutan.

Java

SDK for Java 2.x

Menunjukkan cara membuat alur kerja AWS tanpa server dengan menggunakan AWS Step Functions dan. AWS SDK for Java 2.x Setiap langkah alur kerja diimplementasikan menggunakan AWS Lambda fungsi.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara membuat alur kerja AWS tanpa server dengan menggunakan AWS Step Functions dan. AWS SDK for JavaScript Setiap langkah alur kerja diimplementasikan menggunakan AWS Lambda fungsi.

Lambda adalah layanan komputasi yang memungkinkan Anda menjalankan kode tanpa perlu menyediakan atau mengelola server. Step Functions adalah layanan orkestrasi nirserver yang memungkinkan Anda menggabungkan fungsi Lambda dan layanan AWS lainnya untuk membangun aplikasi bisnis penting.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Contoh ini juga tersedia di [panduan developer AWS SDK for JavaScript v3](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda

Contoh kode berikut menunjukkan cara membuat AWS Lambda fungsi yang dipanggil oleh acara EventBridge terjadwal Amazon.

Java

SDK for Java 2.x

Menunjukkan cara membuat acara EventBridge terjadwal Amazon yang memanggil AWS Lambda fungsi. Konfigurasi EventBridge untuk menggunakan ekspresi cron untuk menjadwalkan saat fungsi Lambda dipanggil. Dalam contoh ini, Anda membuat fungsi Lambda menggunakan API runtime Java Lambda. Contoh ini memanggil AWS layanan yang berbeda untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat aplikasi yang mengirimkan pesan teks seluler kepada karyawan Anda berisi ucapan selamat pada hari jadi setahun kerja mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara membuat acara EventBridge terjadwal Amazon yang memanggil AWS Lambda fungsi. Konfigurasi EventBridge untuk menggunakan ekspresi cron untuk menjadwalkan saat fungsi Lambda dipanggil. Dalam contoh ini, Anda membuat fungsi Lambda dengan menggunakan API runtime JavaScript Lambda. Contoh ini memanggil AWS layanan yang berbeda untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat aplikasi yang mengirimkan pesan teks seluler kepada karyawan Anda berisi ucapan selamat pada hari jadi setahun kerja mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Contoh ini juga tersedia di [panduan developer AWS SDK for JavaScript v3](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihat [Menggunakan DynamoDB dengan SDK AWS](#). Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Keamanan dan kepatuhan di Amazon DynamoDB

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. [Model tanggung jawab bersama](#) menjelaskan hal ini sebagai keamanan dari cloud dan keamanan di cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Efektivitas keamanan kami diuji dan diverifikasi secara rutin oleh auditor pihak ketiga sebagai bagian dari [program kepatuhan AWS](#). Untuk mempelajari tentang program kepatuhan yang berlaku untuk DynamoDB, lihat [Layanan AWS dalam Cakupan Program Kepatuhan](#).
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor-faktor lain termasuk sensitivitas data Anda, persyaratan organisasi Anda, serta undang-undang dan peraturan yang berlaku.

Dokumentasi ini akan membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan DynamoDB. Topik berikut menunjukkan cara mengonfigurasi DynamoDB untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga akan belajar cara menggunakan AWS layanan lain yang dapat membantu Anda memantau dan mengamankan sumber daya DynamoDB Anda.

Topik

- [AWS kebijakan terkelola untuk Amazon DynamoDB](#)
- [Menggunakan kebijakan berbasis sumber daya untuk DynamoDB](#)
- [Perlindungan data di DynamoDB](#)
- [AWS Identity and Access Management \(IAM\)](#)
- [Validasi kepatuhan oleh industri untuk DynamoDB](#)
- [Ketahanan dan pemulihan yang Amazon DynamoDB](#)
- [Keamanan infrastruktur di Amazon DynamoDB](#)
- [AWS PrivateLink untuk DynamoDB](#)
- [Analisis konfigurasi dan kelemahan di Amazon DynamoDB](#)

- [Praktik terbaik keamanan untuk Amazon DynamoDB](#)

AWS kebijakan terkelola untuk Amazon DynamoDB

DynamoDB AWS menggunakan kebijakan terkelola untuk menentukan sekumpulan izin yang dibutuhkan layanan untuk melakukan tindakan tertentu. DynamoDB memelihara dan memperbarui AWS kebijakan yang dikelola. Anda tidak dapat mengubah izin dalam kebijakan AWS terkelola. Untuk informasi selengkapnya tentang kebijakan AWS [AWS terkelola](#), lihat [kebijakan terkelola](#) di Panduan Pengguna IAM.

DynamoDB terkadang dapat menambahkan izin tambahan ke kebijakan terkelola untuk AWS mendukung fitur baru. Jenis pembaruan ini akan memengaruhi semua identitas (pengguna, grup, dan peran) di mana kebijakan tersebut dilampirkan. Kebijakan AWS terkelola kemungkinan besar akan diperbarui saat fitur baru diluncurkan atau saat operasi baru tersedia. DynamoDB tidak akan menghapus izin dari AWS kebijakan terkelola, sehingga pembaruan kebijakan tidak akan merusak izin yang ada.

AWS kebijakan terkelola: DynamoDB ReplicationServiceRolePolicy

Anda tidak dapat melampirkan kebijakan `DynamoDBReplicationServiceRolePolicy` ke entitas IAM Anda. Kebijakan ini dilampirkan ke peran tertaut layanan yang memungkinkan DynamoDB melakukan tindakan atas nama Anda. Untuk informasi selengkapnya, lihat [Menggunakan IAM dengan tabel global](#).

Kebijakan ini memberikan izin yang memungkinkan peran tertaut layanan untuk melakukan replikasi data antara replika tabel global. Kebijakan ini juga memberikan izin administratif untuk mengelola replika tabel global atas nama Anda.

Detail izin

Kebijakan ini memberikan izin untuk melakukan hal berikut:

- `dynamodb` – Melakukan replikasi data dan mengelola replika tabel.
- `application-autoscaling`— Mengambil dan mengelola pengaturan tabel `AutoScaling`
- `account` – Mengambil status wilayah untuk mengevaluasi aksesibilitas replika.
- `iam`— Untuk membuat peran terkait layanan untuk aplikasi `AutoScaling` jika peran terkait layanan belum ada.

Definisi kebijakan yang dikelola ini dapat ditemukan [di sini](#).

AWS kebijakan terkelola: AmazonDynamo DB ReadOnlyAccess

Anda dapat melampirkan kebijakan AmazonDynamoDBReadOnlyAccess ke identitas IAM Anda.

Kebijakan ini memberikan akses hanya-baca ke Amazon DynamoDB.

Detail izin

Kebijakan ini mencakup izin berikut:

- Amazon DynamoDB— Menyediakan akses read-only ke Amazon DynamoDB.
- Amazon DynamoDB Accelerator (DAX)— Menyediakan akses read-only ke Amazon DynamoDB Accelerator (DAX).
- Application Auto Scaling— Memungkinkan prinsipal untuk melihat konfigurasi dari Application Auto Scaling. Ini diperlukan agar pengguna dapat melihat kebijakan penskalaan otomatis yang dilampirkan ke tabel.
- CloudWatch— Memungkinkan kepala sekolah untuk melihat data metrik dan alarm yang dikonfigurasi. CloudWatch Ini diperlukan agar pengguna dapat melihat ukuran tabel yang dapat ditagih dan CloudWatch alarm yang telah dikonfigurasi untuk tabel.
- AWS Data Pipeline— Memungkinkan prinsipal untuk melihat AWS Data Pipeline dan objek terkait.
- Amazon EC2— Memungkinkan kepala sekolah untuk melihat VPC Amazon EC2, subnet, dan grup keamanan.
- IAM— Memungkinkan kepala sekolah untuk melihat peran IAM.
- AWS KMS— Memungkinkan kepala sekolah untuk melihat kunci yang dikonfigurasi di AWS KMS Ini diperlukan agar pengguna dapat melihat AWS KMS keys bahwa mereka membuat dan mengelola di akun mereka.
- Amazon SNS— Memungkinkan kepala sekolah untuk mencantumkan topik dan langganan Amazon SNS berdasarkan topik.
- AWS Resource Groups— Memungkinkan kepala sekolah untuk melihat grup sumber daya dan kueri mereka.
- AWS Resource Groups Tagging— Memungkinkan prinsipal untuk mencantumkan semua sumber daya yang ditandai atau ditandai sebelumnya di suatu Wilayah.
- Kinesis— Memungkinkan kepala sekolah untuk melihat deskripsi aliran data Kinesis.

- Amazon CloudWatch Contributor Insights— Izinkan kepala sekolah untuk melihat data deret waktu yang dikumpulkan oleh aturan Contributor Insights.

Untuk meninjau kebijakan dalam JSON format, lihat [AmazonDynamoDB ReadOnlyAccess](#).

DynamoDB memperbarui kebijakan terkelola AWS

Tabel ini menunjukkan pembaruan kebijakan manajemen AWS akses untuk DynamoDB.

Perubahan	Deskripsi	Tanggal Perubahan
Pembaruan AmazonDynamoDBReadOnlyAccess pada kebijakan yang sudah ada	AmazonDynamoDBReadOnlyAccess menambahkan izin dynamodb:GetResourcePolicy . Izin ini menyediakan akses ke kebijakan berbasis sumber daya baca yang dilampirkan ke sumber daya DynamoDB.	Maret 20, 2024
Pembaruan DynamoDBReadOnlyAccess pada kebijakan yang sudah ada	DynamoDBReadOnlyAccess menambahkan izin dynamodb:GetResourcePolicy . Izin ini memungkinkan peran terkait layanan untuk membaca kebijakan berbasis sumber daya yang dilampirkan ke sumber daya DynamoDB.	15 Desember 2023
Pembaruan DynamoDBReadOnlyAccess pada kebijakan yang sudah ada	DynamoDBReadOnlyAccess menambahkan izin account:ListRegions . Izin ini memungkinkan peran tertaut layanan dapat mengevaluasi aksesibilitas replika	10 Mei 2023
DynamoDBReadOnlyAccess	Menambahkan informasi tentang kebijakan yang dikelola DynamoDB	10 Mei 2023

Perubahan	Deskripsi	Tanggal Perubahan
nServiceRolePolicy ditambahkan ke daftar kebijakan yang dikelola	ApplicationServiceRolePolicy, yang digunakan oleh peran tertaut layanan tabel global DynamoDB.	
Tabel global DynamoDB mulai melacak perubahan	Tabel global DynamoDB mulai melacak perubahan untuk AWS kebijakan yang dikelola.	10 Mei 2023

Menggunakan kebijakan berbasis sumber daya untuk DynamoDB

DynamoDB mendukung kebijakan berbasis sumber daya untuk tabel, indeks, dan aliran. Kebijakan berbasis sumber daya memungkinkan Anda menentukan izin akses dengan menentukan siapa yang memiliki akses ke setiap sumber daya, dan tindakan yang diizinkan untuk dilakukan pada setiap sumber daya.

Anda dapat melampirkan kebijakan berbasis sumber daya ke sumber daya DynamoDB, seperti tabel atau aliran. Dalam kebijakan ini, Anda menentukan izin untuk [prinsip Identity and Access Management \(IAM\) yang dapat melakukan tindakan tertentu pada sumber daya DynamoDB](#) ini. Misalnya, kebijakan yang dilampirkan pada tabel akan berisi izin untuk mengakses tabel dan indeksnya. Akibatnya, kebijakan berbasis sumber daya dapat membantu Anda menyederhanakan kontrol akses untuk tabel, indeks, dan aliran DynamoDB, dengan menentukan izin di tingkat sumber daya. Ukuran maksimum kebijakan yang dapat Anda lampirkan ke sumber daya DynamoDB adalah 20 KB.

Manfaat signifikan menggunakan kebijakan berbasis sumber daya adalah untuk menyederhanakan kontrol akses lintas akun untuk menyediakan akses lintas akun ke prinsip-prinsip IAM yang berbeda. Akun AWS Untuk informasi selengkapnya, lihat [Kebijakan berbasis sumber daya untuk akses lintas akun](#).

[Kebijakan berbasis sumber daya juga mendukung integrasi dengan IAM Access Analyzer external access analyzer dan kemampuan Block Public Access \(BPA\)](#). IAM Access Analyzer melaporkan akses lintas akun ke entitas eksternal yang ditentukan dalam kebijakan berbasis sumber daya. Ini juga memberikan visibilitas untuk membantu Anda memperbaiki izin dan mematuhi prinsip

hak istimewa paling sedikit. BPA membantu Anda mencegah akses publik ke tabel, indeks, dan aliran DynamoDB Anda, dan secara otomatis diaktifkan dalam alur kerja pembuatan dan modifikasi kebijakan berbasis sumber daya.

Topik

- [Membuat tabel dengan kebijakan berbasis sumber daya](#)
- [Lampirkan kebijakan ke tabel yang ada](#)
- [Lampirkan kebijakan berbasis sumber daya ke aliran](#)
- [Menghapus kebijakan berbasis sumber daya dari tabel](#)
- [Akses lintas akun dengan kebijakan berbasis sumber daya](#)
- [Memblokir akses publik dengan kebijakan berbasis sumber daya](#)
- [Tindakan IAM didukung oleh kebijakan berbasis sumber daya](#)
- [Otorisasi dengan kebijakan berbasis identitas IAM dan kebijakan berbasis sumber daya DynamoDB](#)
- [Contoh kebijakan berbasis sumber daya](#)
- [Pertimbangan kebijakan berbasis sumber daya](#)
- [Praktik terbaik kebijakan berbasis sumber daya](#)

Membuat tabel dengan kebijakan berbasis sumber daya

[Anda dapat menambahkan kebijakan berbasis sumber daya saat membuat tabel menggunakan konsol DynamoDB, CreateTableAPI, SDK, atau template. AWS CLI](#) [AWS CloudFormation](#)

AWS CLI

Contoh berikut membuat tabel bernama *MusicCollection* menggunakan `create-table` AWS CLI perintah. Perintah ini juga menyertakan `resource-policy` parameter yang menambahkan kebijakan berbasis sumber daya ke tabel. Kebijakan ini memungkinkan pengguna *John* untuk melakukan tindakan [RestoreTableToPointInTimeGetItem](#), dan [PutItemAPI](#) pada tabel.

Ingatlah untuk mengganti teks yang *dicetak miring dengan informasi* spesifik sumber daya Anda.

```
aws dynamodb create-table \  
  --table-name MusicCollection \  
  --attribute-definitions AttributeName=Artist,AttributeType=S  
  AttributeName=SongTitle,AttributeType=S \  
  --resource-policy Policy
```



```

--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. [Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/.](https://console.aws.amazon.com/dynamodb/)
2. Di dasbor, pilih Buat tabel.
3. Dalam Rincian tabel, masukkan nama tabel, kunci partisi, dan urutkan detail kunci.
4. Di Pengaturan tabel, pilih Sesuaikan pengaturan.
5. (Opsional) Tentukan opsi Anda untuk kelas Tabel, Kalkulator kapasitas, Pengaturan kapasitas baca/tulis, Indeks sekunder, Enkripsi saat istirahat, dan Perlindungan penghapusan.
6. Dalam kebijakan berbasis sumber daya, tambahkan kebijakan untuk menentukan izin akses untuk tabel dan indeksnya. Dalam kebijakan ini, Anda menentukan siapa yang memiliki akses ke sumber daya ini, dan tindakan yang diizinkan untuk dilakukan pada setiap sumber daya. Untuk menambahkan kebijakan, lakukan salah satu hal berikut:
 - Ketik atau tempel dokumen kebijakan JSON. Untuk detail tentang bahasa kebijakan IAM, lihat [Membuat kebijakan menggunakan editor JSON di Panduan Pengguna IAM](#).

i Tip

Untuk melihat contoh kebijakan berbasis sumber daya di Panduan Pengembang Amazon DynamoDB, pilih Contoh kebijakan.

- Pilih Tambahkan pernyataan baru untuk menambahkan pernyataan baru dan masukkan informasi di bidang yang disediakan. Ulangi langkah ini untuk sebanyak mungkin pernyataan yang ingin Anda tambahkan.

A Important

Pastikan Anda menyelesaikan peringatan, kesalahan, atau saran keamanan sebelum menyimpan kebijakan.

Contoh kebijakan IAM berikut memungkinkan pengguna *John* untuk melakukan tindakan [RestoreTableToPointInTime](#), [GetItem](#), dan [PutItem](#) API di atas meja *MusicCollection*.

Ingatlah untuk mengganti teks yang *dicetak miring dengan informasi* spesifik sumber daya Anda.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/John"
      },
      "Action": [
        "dynamodb:RestoreTableToPointInTime",
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

7. (Opsional) Pilih Pratinjau akses eksternal di sudut kanan bawah untuk melihat bagaimana kebijakan baru memengaruhi publik dan akses lintas akun ke sumber daya Anda. Sebelum Anda menyimpan kebijakan Anda, Anda dapat memeriksa apakah itu memperkenalkan temuan Penganalisa Akses IAM baru atau menyelesaikan temuan yang ada. Jika Anda tidak melihat penganalisis aktif, pilih Buka Penganalisis Akses untuk [membuat penganalisis akun](#) di Penganalisis Akses IAM. Untuk informasi selengkapnya, lihat [Pratinjau akses](#).
8. Pilih Buat tabel.

AWS CloudFormation Template

Using the AWS::DynamoDB::Table resource

CloudFormation Template berikut membuat tabel dengan aliran menggunakan sumber daya [AWS::DynamoDB::Table](#). Template ini juga menyertakan kebijakan berbasis sumber daya yang dilampirkan ke tabel dan aliran.

```
{
  "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

CloudFormation Template berikut membuat tabel dengan sumber daya [AWS: :DynamoDB::](#) dan melampirkan kebijakan berbasis GlobalTable sumber daya ke tabel dan alirannya.

```
{
  "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": "*"
      }]
    }
  ]
}

```

Lampirkan kebijakan ke tabel yang ada

[Anda dapat melampirkan kebijakan berbasis sumber daya ke tabel yang ada atau mengubah kebijakan yang ada menggunakan konsol DynamoDB, PutResourcePolicyAPI, SDK, atau templat. AWS CLI](#)
[AWS CloudFormation](#)

AWS CLI contoh untuk melampirkan kebijakan baru

Contoh kebijakan IAM berikut menggunakan `put-resource-policy` AWS CLI perintah untuk melampirkan kebijakan berbasis sumber daya ke tabel yang ada. Contoh ini memungkinkan pengguna *John* untuk melakukan tindakan [GetItem](#), [PutItem](#), [UpdateItem](#), dan [UpdateTableAPI](#) pada tabel yang ada bernama *MusicCollection*.

Ingatlah untuk mengganti teks yang *dicetak miring dengan informasi* spesifik sumber daya Anda.

```
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\"
        }
      ]
    }"
```

AWS CLI contoh untuk memperbarui kebijakan yang ada secara kondisional

Untuk memperbarui kebijakan tabel berbasis sumber daya yang ada secara kondisional, Anda dapat menggunakan parameter opsional. `expected-revision-id` Contoh berikut hanya akan memperbarui kebijakan jika ada di DynamoDB dan ID revisi saat ini cocok dengan parameter yang disediakan. `expected-revision-id`

```
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:PutItem\",
            \"dynamodb:UpdateItem\",
            \"dynamodb:UpdateTable\"
          ],
          \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
        }
      ]
    }"
```

```

        \ "Action\": [
            \ "dynamodb:GetItem\",
            \ "dynamodb:UpdateItem\",
            \ "dynamodb:UpdateTable\"
        ],
        \ "Resource\": \ "arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\\"
    }
]
}"

```

AWS Management Console

1. [Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/.](https://console.aws.amazon.com/dynamodb/)
2. Dari dasbor, pilih tabel yang ada.
3. Arahkan ke tab Izin, dan pilih Buat kebijakan tabel.
4. Di editor kebijakan berbasis sumber daya, tambahkan kebijakan yang ingin dilampirkan dan pilih Buat kebijakan.

Contoh kebijakan IAM berikut memungkinkan pengguna *John* untuk melakukan tindakan [GetItem](#)., [PutItemUpdateItem](#), dan [UpdateTable](#) API pada tabel yang ada bernama *MusicCollection*.

Ingatlah untuk mengganti teks yang *dicetak miring dengan informasi* spesifik sumber daya Anda.

```

{
  "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"
  }
]
}
```

AWS SDK for Java 2.x

Contoh kebijakan IAM berikut menggunakan `putResourcePolicy` metode untuk melampirkan kebijakan berbasis sumber daya ke tabel yang ada. Kebijakan ini memungkinkan pengguna untuk melakukan tindakan [GetItem](#) API pada tabel yang ada.

```
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.
                allowed AWS Principal - 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" +
        "\n" +
        "            \"Action\": [\n" +
```

```

        "                \"dynamodb:GetItem\"\\n\" +
        "                ],\\n\" +
        "                \"Resource\": \"\" + tableArn + \"\"\\n\" +
        "            }\\n\" +
        "        ]\\n\" +
        "    }";
    }
}

```

Lampirkan kebijakan berbasis sumber daya ke aliran

[Anda dapat melampirkan kebijakan berbasis sumber daya ke aliran tabel yang ada atau mengubah kebijakan yang ada menggunakan konsol DynamoDB, PutResourcePolicyAPI, SDK, atau templat. \[AWS CLI\]\(#\)\[AWS CloudFormation\]\(#\)](#)

Note

Anda tidak dapat melampirkan kebijakan ke aliran saat membuatnya menggunakan [CreateTable](#) atau [UpdateTable](#) API. Namun, Anda dapat mengubah atau menghapus kebijakan setelah tabel dihapus. Anda juga dapat mengubah atau menghapus kebijakan aliran yang dinonaktifkan.

AWS CLI

Contoh kebijakan IAM berikut menggunakan `put-resource-policy` AWS CLI perintah untuk melampirkan kebijakan berbasis sumber daya ke aliran tabel bernama *MusicCollection*. Contoh ini memungkinkan pengguna *John* untuk melakukan tindakan [GetRecordsGetShardIterator](#), dan [DescribeStream](#) API di streaming.

Ingatlah untuk mengganti teks yang *dicetak miring dengan informasi* spesifik sumber daya Anda.

```

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. [Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/.](https://console.aws.amazon.com/dynamodb/)
2. Pada dashboard konsol DynamoDB, pilih Tabel dan kemudian pilih tabel yang ada.

Pastikan tabel yang Anda pilih memiliki aliran yang diaktifkan. Untuk informasi tentang mengaktifkan aliran untuk tabel, lihat [Mengaktifkan aliran](#).

3. Pilih tab Izin.
4. Dalam kebijakan berbasis sumber daya untuk aliran aktif, pilih Buat kebijakan aliran.
5. Di editor kebijakan berbasis Sumber Daya, tambahkan kebijakan untuk menentukan izin akses untuk aliran. Dalam kebijakan ini, Anda menentukan siapa yang memiliki akses ke aliran dan tindakan yang diizinkan untuk dilakukan di streaming. Untuk menambahkan kebijakan, lakukan salah satu hal berikut:
 - Ketik atau tempel dokumen kebijakan JSON. Untuk detail tentang bahasa kebijakan IAM, lihat [Membuat kebijakan menggunakan editor JSON di Panduan Pengguna IAM](#).

Tip

Untuk melihat contoh kebijakan berbasis sumber daya di Panduan Pengembang Amazon DynamoDB, pilih Contoh kebijakan.

- Pilih Tambahkan pernyataan baru untuk menambahkan pernyataan baru dan masukkan informasi di bidang yang disediakan. Ulangi langkah ini untuk sebanyak mungkin pernyataan yang ingin Anda tambahkan.

⚠ Important

Pastikan Anda menyelesaikan peringatan, kesalahan, atau saran keamanan sebelum menyimpan kebijakan.

6. (Opsional) Pilih Pratinjau akses eksternal di sudut kanan bawah untuk melihat bagaimana kebijakan baru memengaruhi publik dan akses lintas akun ke sumber daya Anda. Sebelum Anda menyimpan kebijakan Anda, Anda dapat memeriksa apakah itu memperkenalkan temuan Penganalisa Akses IAM baru atau menyelesaikan temuan yang ada. Jika Anda tidak melihat penganalisis aktif, pilih Buka Penganalisis Akses untuk [membuat penganalisis akun](#) di Penganalisis Akses IAM. Untuk informasi selengkapnya, lihat [Pratinjau akses](#).
7. Pilih Buat kebijakan.

Contoh kebijakan IAM berikut melampirkan kebijakan berbasis sumber daya ke aliran tabel bernama *MusicCollection*. Contoh ini memungkinkan pengguna *John* untuk melakukan tindakan [GetRecordsGetShardIterator](#), dan [DescribeStream](#) API di streaming.

Ingatlah untuk mengganti teks yang *dicetak miring dengan informasi* spesifik sumber daya Anda.

```
{
  "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"  
  ]  
}  
]  
}
```

Menghapus kebijakan berbasis sumber daya dari tabel

Anda dapat menghapus kebijakan berbasis sumber daya dari tabel yang ada dengan menggunakan konsol DynamoDB, [DeleteResourcePolicy](#) API, SDK, atau templat. AWS CLI AWS AWS CloudFormation

AWS CLI

Contoh berikut menggunakan `delete-resource-policy` AWS CLI perintah untuk menghapus kebijakan berbasis sumber daya dari tabel bernama. *MusicCollection*

Ingatlah untuk mengganti teks yang *dicetak miring dengan informasi* spesifik sumber daya Anda.

```
aws dynamodb delete-resource-policy \  
  --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

AWS Management Console

1. [Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/.](https://console.aws.amazon.com/dynamodb/)
2. Pada dashboard konsol DynamoDB, pilih Tabel dan kemudian pilih tabel yang ada.
3. Pilih Izin.
4. Dari menu tarik-turun Kelola kebijakan, pilih Hapus kebijakan.
5. Di kotak dialog Hapus kebijakan berbasis sumber daya untuk tabel, ketik **confirm** untuk mengonfirmasi tindakan penghapusan.
6. Pilih Hapus.

Akses lintas akun dengan kebijakan berbasis sumber daya

Dengan menggunakan kebijakan berbasis sumber daya, Anda dapat memberikan akses lintas akun ke sumber daya yang tersedia di berbagai sumber daya. Akun AWS Semua akses lintas akun yang

diizinkan oleh kebijakan berbasis sumber daya akan dilaporkan melalui temuan akses eksternal IAM Access Analyzer jika Anda memiliki penganalisis yang sama dengan sumber daya. Wilayah AWS Penganalisis Akses IAM menjalankan pemeriksaan kebijakan untuk memvalidasi kebijakan Anda terhadap [tata bahasa kebijakan](#) IAM dan [praktik terbaik](#). Pemeriksaan ini menghasilkan temuan dan memberikan rekomendasi yang dapat ditindaklanjuti untuk membantu Anda menulis kebijakan yang fungsional, dan sesuai dengan praktik terbaik keamanan. [Anda dapat melihat temuan aktif dari IAM Access Analyzer di tab Permissions konsol DynamoDB.](#)

Untuk informasi tentang memvalidasi kebijakan menggunakan IAM Access Analyzer, lihat [validasi kebijakan IAM Access Analyzer](#) di Panduan Pengguna IAM. Untuk melihat daftar peringatan, kesalahan, dan saran yang ditampilkan oleh Penganalisis Akses IAM, lihat referensi pemeriksaan kebijakan [Penganalisis Akses IAM](#).

Untuk memberikan [GetItem](#) izin kepada pengguna A di akun A untuk mengakses tabel B di akun B, lakukan langkah-langkah berikut:

1. Lampirkan kebijakan berbasis sumber daya ke tabel B yang memberikan izin kepada pengguna A untuk melakukan tindakan. `GetItem`
2. Lampirkan kebijakan berbasis identitas ke pengguna A yang memberinya izin untuk melakukan tindakan pada `GetItem` tabel B.

Menggunakan opsi Pratinjau akses eksternal yang tersedia di konsol [DynamoDB](#), Anda dapat melihat pratinjau bagaimana kebijakan baru memengaruhi akses publik dan lintas akun ke sumber daya Anda. Sebelum Anda menyimpan kebijakan Anda, Anda dapat memeriksa apakah itu memperkenalkan temuan Penganalisa Akses IAM baru atau menyelesaikan temuan yang ada. Jika Anda tidak melihat penganalisis aktif, pilih Buka Penganalisis Akses untuk [membuat penganalisis akun](#) di Penganalisis Akses IAM. Untuk informasi selengkapnya, lihat [Pratinjau akses](#).

Parameter nama tabel di bidang data DynamoDB dan API bidang kontrol menerima Nama Sumber Daya Amazon (ARN) lengkap dari tabel untuk mendukung operasi lintas akun. Jika Anda hanya memberikan parameter nama tabel alih-alih ARN lengkap, operasi API akan dilakukan pada tabel di akun tempat pemohon berada. Untuk contoh kebijakan yang menggunakan akses lintas akun, lihat [Kebijakan berbasis sumber daya untuk akses lintas akun](#).

Akun pemilik sumber daya akan dikenakan biaya bahkan ketika kepala sekolah dari akun lain membaca dari atau menulis ke tabel DynamoDB di akun pemilik. Jika tabel telah menyediakan throughput, jumlah semua permintaan dari akun pemilik dan pemohon di akun lain akan menentukan

apakah permintaan akan dibatasi (jika penskalaan otomatis dinonaktifkan) atau diperkecil jika penskalaan otomatis diaktifkan.

Permintaan akan dicatat di CloudTrail log pemilik dan akun pemohon sehingga masing-masing dari dua akun dapat melacak akun mana yang mengakses data apa.

Note

Akses lintas akun dari [API pesawat kontrol](#) memiliki batas transaksi per detik (TPS) yang lebih rendah yaitu 500 permintaan.

Memblokir akses publik dengan kebijakan berbasis sumber daya

[Blokir Akses Publik \(BPA\)](#) adalah fitur yang mengidentifikasi dan mencegah melampirkan kebijakan berbasis sumber daya yang memberikan akses publik ke tabel, indeks, atau aliran DynamoDB Anda di seluruh akun Amazon Web Services () Anda.AWS Dengan BPA, Anda dapat mencegah akses publik ke sumber daya DynamoDB Anda. BPA melakukan pemeriksaan selama pembuatan atau modifikasi kebijakan berbasis sumber daya dan membantu meningkatkan postur keamanan Anda dengan DynamoDB.

BPA menggunakan [penalaran otomatis](#) untuk menganalisis akses yang diberikan oleh kebijakan berbasis sumber daya Anda dan memberi tahu Anda jika izin tersebut ditemukan pada saat mengelola kebijakan berbasis sumber daya. Analisis memverifikasi akses di semua pernyataan kebijakan berbasis sumber daya, tindakan, dan kumpulan kunci kondisi yang digunakan dalam kebijakan Anda.

Important

BPA membantu melindungi sumber daya Anda dengan mencegah akses publik diberikan melalui kebijakan berbasis sumber daya yang langsung dilampirkan ke sumber daya DynamoDB Anda, seperti tabel, indeks, dan aliran. Selain menggunakan BPA, hati-hati memeriksa kebijakan berikut untuk mengonfirmasi bahwa mereka tidak memberikan akses publik:

- Kebijakan berbasis identitas yang dilampirkan pada AWS prinsipal terkait (misalnya, peran IAM)

- Kebijakan berbasis sumber daya yang dilampirkan pada AWS sumber daya terkait (misalnya, kunci (KMS AWS Key Management Service))

Anda harus memastikan bahwa [prinsipal](#) tidak menyertakan * entri atau bahwa salah satu kunci kondisi yang ditentukan membatasi akses dari prinsipal ke sumber daya. Jika kebijakan berbasis sumber daya memberikan akses publik ke tabel, indeks, atau streaming Anda, Akun AWS DynamoDB akan memblokir Anda dari membuat atau memodifikasi kebijakan hingga spesifikasi dalam kebijakan diperbaiki dan dianggap non-publik.

Anda dapat membuat kebijakan non-publik dengan menentukan satu atau beberapa prinsip di dalam blok. `Principal` Contoh kebijakan berbasis sumber daya berikut memblokir akses publik dengan menentukan dua prinsip.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "*",
  "Resource": "*"
}
```

Kebijakan yang membatasi akses dengan menentukan kunci kondisi tertentu juga tidak dianggap publik. Seiring dengan evaluasi prinsipal yang ditentukan dalam kebijakan berbasis sumber daya, [kunci kondisi tepercaya](#) berikut digunakan untuk menyelesaikan evaluasi kebijakan berbasis sumber daya untuk akses non-publik:

- `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`

Selain itu, agar kebijakan berbasis sumber daya bersifat non-publik, nilai untuk Amazon Resource Name (ARN) dan kunci string tidak boleh berisi wildcard atau variabel. Jika kebijakan berbasis sumber daya menggunakan `aws:PrincipalIsAWSService` kunci, Anda harus memastikan bahwa Anda telah menetapkan nilai kunci ke `true`.

Kebijakan berikut membatasi akses ke pengguna John di akun yang ditentukan. Kondisi tersebut membuat `Principal` terkendala dan tidak dianggap publik.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "*",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/John"
    }
  }
}
```

Contoh berikut dari kebijakan berbasis sumber daya non-publik kendala menggunakan operator `sourceVPC StringEquals`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
"Effect": "Allow",
"Principal": {
  "AWS": "*"
},
"Action": "*",
"Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
"Condition": {
  "StringEquals": {
    "aws:SourceVpc": [
      "vpc-91237329"
    ]
  }
}
]
```

Tindakan IAM didukung oleh kebijakan berbasis sumber daya

Tindakan IAM dan dukungan lintas akun melalui kebijakan berbasis sumber daya dibatasi pada serangkaian API DynamoDB tertentu. Anda tidak dapat melampirkan kebijakan berbasis sumber daya ke jenis sumber daya, seperti pencadangan dan impor. Selain itu, API yang beroperasi pada jenis sumber daya ini dikecualikan dari tindakan IAM yang didukung dalam kebijakan berbasis sumber daya. Karena administrator tabel mengonfigurasi setelan tabel internal dalam akun yang sama, API, seperti [UpdateTimeToLive](#) dan [DisableKinesisStreamingDestination](#), tidak mendukung akses lintas akun melalui kebijakan berbasis sumber daya.

Bidang data DynamoDB dan API bidang kontrol yang mendukung akses lintas akun juga mendukung overloading nama tabel, yang memungkinkan Anda menentukan ARN tabel alih-alih nama tabel. Anda dapat menentukan tabel ARN dalam `TableName` parameter API ini. Namun, tidak semua API ini mendukung akses lintas akun.

Tabel berikut mencantumkan dukungan tingkat API untuk kebijakan berbasis sumber daya dan akses lintas akun.

Tindakan API	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
--------------	---	----------------------

Data Plane - Tables/indexes

Tindakan API	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DeleteItem	Yes	Yes
GetItem	Yes	Yes
PutItem	Yes	Yes
Query	Yes	Yes
Scan	Yes	Yes
UpdateItem	Yes	Yes
TransactGetItems	Yes	Yes
TransactWriteItems	Yes	Yes
BatchGetItem	Yes	Yes
BatchWriteItem	Yes	Yes
PartiQL		
BatchExecuteStatement	Yes	No
ExecuteStatement	Yes	No
ExecuteTransaction	Yes	No
Control Plane - Tables		
CreateTable	No	No
DeleteTable	Yes	Yes
DescribeTable	Yes	Yes
UpdateTable	Yes	Yes
Version 2019.11.21 (Current) global tables		

Tindakan API	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeTableReplicaAutoScaling	Yes	No
UpdateTableReplicaAutoScaling	Yes	No
Version 2017.11.29 (Legacy) global table		
CreateGlobalTable	No	No
DescribeGlobalTable	No	No
DescribeGlobalTableSettings	No	No
ListGlobalTables	No	No
UpdateGlobalTable	No	No
UpdateGlobalTableSettings	No	No
Tags		
ListTagsOfResource	Yes	Yes
TagResource	Yes	Yes
UntagResource	Yes	Yes
Backup/Restore		
CreateBackup	Yes	No
DescribeBackup	No	No
DeleteBackup	No	No
RestoreTableFromBackup	No	No
Continuous Backup/Restore (PITR)		

Tindakan API	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeContinuousBackups	Yes	No
RestoreTableToPointInTime	Yes	No
UpdateContinuousBackups	Yes	No
Contributor Insights		
DescribeContributorInsights	Yes	No
ListContributorInsights	No	No
UpdateContributorInsights	Yes	No
Export		
DescribeExport	No	No
ExportTableToPointInTime	Yes	No
ListExports	No	No
Import		
DescribeImport	No	No
ImportTable	No	No
ListImports	No	No
Kinesis		
DescribeKinesisStreamingDestination	Yes	No
DisableKinesisStreamingDestination	Yes	No

Tindakan API	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
EnableKinesisStreamingDestination	Yes	No
UpdateKinesisStreamingDestination	Yes	No
Resource policies		
GetResourcePolicy	Yes	No
PutResourcePolicy	Yes	No
DeleteResourcePolicy	Yes	No
Time-to-Live		
DescribeTimeToLive	Yes	No
UpdateTimeToLive	Yes	No
Others		
DescribeLimits	No	No
DescribeEndpoints	No	No
ListBackups	No	No
ListTables	No	No

Tabel berikut mencantumkan dukungan API API API DynamoDB Streams API untuk kebijakan berbasis sumber daya dan akses lintas akun.

Tindakan API	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeStream	Ya	Ya

Tindakan API	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
GetRecords	Ya	Ya
GetShardIterator	Ya	Ya
ListStreams	Tidak	Tidak

Otorisasi dengan kebijakan berbasis identitas IAM dan kebijakan berbasis sumber daya DynamoDB

Kebijakan berbasis identitas dilampirkan pada identitas, seperti pengguna IAM, grup pengguna, dan peran. Ini adalah dokumen kebijakan IAM yang mengontrol tindakan apa yang dapat dilakukan identitas, pada sumber daya mana, dan dalam kondisi apa. [Kebijakan berbasis identitas dapat dikelola atau kebijakan inline.](#)

Kebijakan berbasis sumber daya adalah dokumen kebijakan IAM yang Anda lampirkan ke sumber daya, seperti tabel DynamoDB. Kebijakan ini memberikan izin prinsipal yang ditentukan untuk melakukan tindakan tertentu pada sumber daya tersebut dan menetapkan di bawah ketentuan yang berlaku. Misalnya, kebijakan berbasis sumber daya untuk tabel DynamoDB juga menyertakan indeks yang terkait dengan tabel. Kebijakan berbasis sumber daya merupakan kebijakan inline. Tidak ada kebijakan berbasis sumber daya terkelola.

Untuk informasi selengkapnya tentang kebijakan ini, lihat Kebijakan [berbasis identitas dan kebijakan berbasis sumber daya](#) di Panduan Pengguna IAM.

Jika prinsipal IAM berasal dari akun yang sama dengan pemilik sumber daya, kebijakan berbasis sumber daya cukup untuk menentukan izin akses ke sumber daya. Anda masih dapat memilih untuk memiliki kebijakan berbasis identitas IAM bersama dengan kebijakan berbasis sumber daya. Untuk akses lintas akun, Anda harus secara eksplisit mengizinkan akses dalam kebijakan identitas dan sumber daya sebagaimana ditentukan dalam [Akses lintas akun dengan kebijakan berbasis sumber daya](#). Saat Anda menggunakan kedua jenis kebijakan tersebut, kebijakan dievaluasi seperti yang dijelaskan dalam [Menentukan apakah permintaan diizinkan atau ditolak dalam akun.](#)

Contoh kebijakan berbasis sumber daya

Saat Anda menentukan ARN di Resource bidang kebijakan berbasis sumber daya, kebijakan akan berlaku hanya jika ARN yang ditentukan cocok dengan ARN sumber daya DynamoDB yang dilampirkan.

Note

Ingatlah untuk mengganti teks yang *dicetak miring dengan informasi* spesifik sumber daya Anda.

Kebijakan berbasis sumber daya untuk tabel

Kebijakan berbasis sumber daya berikut yang dilampirkan ke tabel DynamoDB bernama *MusicCollection*, memberikan izin kepada pengguna IAM *John* dan *Jane* untuk melakukan dan tindakan pada sumber daya. [GetItemBatchGetItemMusicCollection](#)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:user/John",
          "arn:aws:iam::111122223333:user/Jane"
        ]
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
      ]
    }
  ]
}
```

Kebijakan berbasis sumber daya untuk aliran

Kebijakan berbasis sumber daya berikut yang dilampirkan ke aliran DynamoDB bernama `2024-02-12T18:57:26.492` memberi pengguna IAM *John* dan *Jane* izin untuk melakukan, dan tindakan API pada sumber daya. [GetRecordsGetShardIteratorDescribeStream](#)`2024-02-12T18:57:26.492`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:user/John",
          "arn:aws:iam::111122223333:user/Jane"
        ]
      },
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

Kebijakan berbasis sumber daya untuk akses untuk melakukan semua tindakan pada sumber daya tertentu

Untuk memungkinkan pengguna melakukan semua tindakan pada tabel dan semua indeks terkait dengan tabel, Anda dapat menggunakan wildcard (*) untuk mewakili tindakan dan sumber daya yang

terkait dengan tabel. Menggunakan karakter wild card untuk sumber daya, akan memungkinkan pengguna mengakses tabel DynamoDB dan semua indeks terkait, termasuk yang belum dibuat. Misalnya, kebijakan berikut akan memberikan izin kepada pengguna *John* untuk melakukan tindakan apa pun di atas *MusicCollection* meja dan semua indeksnya, termasuk indeks apa pun yang akan dibuat di masa mendatang.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": "arn:aws:iam::111122223333:user/John",
      "Action": "*",
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/index/*"
      ]
    }
  ]
}
```

Kebijakan berbasis sumber daya untuk akses lintas akun

Anda dapat menentukan izin untuk identitas IAM lintas akun untuk mengakses sumber daya DynamoDB. Misalnya, Anda mungkin memerlukan pengguna dari akun tepercaya untuk mendapatkan akses membaca konten tabel Anda, dengan syarat bahwa mereka hanya mengakses item tertentu dan atribut tertentu dalam item tersebut. Kebijakan berikut memungkinkan akses ke pengguna *John* dari Akun AWS ID tepercaya *111111111111* untuk mengakses data dari tabel di akun *123456789012* dengan menggunakan API. [GetItem](#) Kebijakan ini memastikan bahwa pengguna hanya dapat mengakses item dengan kunci utama *Jane* dan bahwa pengguna hanya dapat mengambil atribut *Artist* dan *SongTitle*, tetapi tidak ada atribut lainnya.

```
{
  "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"
    ]
  }
}
}
]
}

```

Selain kebijakan berbasis sumber daya sebelumnya, kebijakan berbasis identitas yang dilampirkan pada pengguna *John* juga perlu mengizinkan tindakan API agar akses lintas akun berfungsi. *GetItem* **berikut ini adalah contoh kebijakan berbasis identitas yang harus Anda lampirkan ke pengguna John.**

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountIdentityBasedPolicy",
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
      ],
      "Condition": {
        "ForAllValues:StringEquals": {
          "dynamodb:LeadingKeys": "Jane",
          "dynamodb:Attributes": [
            "Artist",
            "SongTitle"
          ]
        }
      }
    }
  ]
}

```

```

    ]
  }
}
]
}

```

Pengguna John dapat membuat `GetItem` permintaan dengan menentukan tabel ARN dalam parameter untuk mengakses tabel `table-name` di akun `MusicCollection123456789012`.

```

aws dynamodb get-item \
  --table-name arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
  --key '{"Artist": {"S": "Jane"}, "bash": {"SongTitle": "Monsoon"}}' \
  --return-consumed-capacity TOTAL

```

Kebijakan berbasis sumber daya dengan kondisi alamat IP

Anda dapat menerapkan kondisi untuk membatasi alamat IP sumber, virtual private cloud (VPC), dan titik akhir VPC (VPCE). Anda dapat menentukan izin berdasarkan alamat sumber permintaan asal. Misalnya, Anda mungkin ingin mengizinkan pengguna mengakses sumber daya DynamoDB hanya jika mereka diakses dari sumber IP tertentu, seperti titik akhir VPN perusahaan. Tentukan alamat IP ini dalam `Condition` pernyataan.

Contoh berikut memungkinkan pengguna *John* akses ke sumber daya DynamoDB ketika IP sumber dan `54.240.143.0/24 2001:DB8:1234:5678::/64`

```

{
  "Id": "PolicyId2",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowIPmix",
      "Effect": "Allow",
      "Principal": "arn:aws:iam::111111111111:user/John",
      "Action": "dynamodb:*",
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "54.240.143.0/24",
            "2001:DB8:1234:5678::/64"
          ]
        }
      }
    }
  ]
}

```

```

    ]
  }
}
]
}

```

Anda juga dapat menolak semua akses ke sumber daya DynamoDB kecuali jika sumbernya adalah titik akhir VPC tertentu, misalnya `vpce-1a2b3c4d`.

```

{
  "Id": "PolicyId",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AccessToSpecificVPCEOnly",
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:sourceVpce": "vpce-1a2b3c4d"
        }
      }
    }
  ]
}

```

Kebijakan berbasis sumber daya menggunakan peran IAM

Anda juga dapat menentukan peran layanan IAM dalam kebijakan berbasis sumber daya. Entitas IAM yang mengambil peran ini dibatasi oleh tindakan yang diizinkan yang ditentukan untuk peran dan kumpulan sumber daya tertentu dalam kebijakan berbasis sumber daya.

Contoh berikut memungkinkan entitas IAM untuk melakukan tindakan apa pun pada sumber daya *MusicCollection* dan *MusicCollection* DynamoDB.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {

```

```
"Sid": "1111",
"Effect": "Allow",
"Principal": { "AWS": "arn:aws:iam::111122223333:role/John" }
"Action": "*"
"Resource": [
  "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
  "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/*"
]
}
]
}
```

Pertimbangan kebijakan berbasis sumber daya

Saat Anda menentukan kebijakan berbasis sumber daya untuk sumber daya DynamoDB Anda, pertimbangan berikut berlaku:

Pertimbangan umum

- Ukuran maksimum yang didukung untuk dokumen kebijakan berbasis sumber daya adalah 20 KB. DynamoDB menghitung spasi putih saat menghitung ukuran kebijakan terhadap batas ini.
- Pembaruan kebijakan berikutnya untuk sumber daya tertentu diblokir selama 15 detik setelah pembaruan kebijakan berhasil untuk sumber daya yang sama.
- Saat ini, Anda hanya dapat melampirkan kebijakan berbasis sumber daya ke aliran yang ada. Anda tidak dapat melampirkan kebijakan ke aliran saat membuatnya.

Pertimbangan tabel global

- Kebijakan berbasis sumber daya tidak didukung untuk replika [versi tabel Global 2017.11.29](#) (Legacy).
- Dalam kebijakan berbasis sumber daya, jika tindakan untuk DynamoDB service-linked role (SLR) untuk mereplikasi data untuk tabel global ditolak, menambahkan atau menghapus replika akan gagal dengan kesalahan.
- GlobalTableSumber daya [AWS: :DynamoDB:](#) tidak mendukung pembuatan replika dan menambahkan kebijakan berbasis sumber daya ke replika itu dalam pembaruan tumpukan yang sama di Wilayah selain Wilayah tempat Anda menerapkan pembaruan tumpukan.

Pertimbangan lintas akun

- Akses lintas akun menggunakan kebijakan berbasis sumber daya tidak mendukung tabel terenkripsi dengan kunci AWS terkelola karena Anda tidak dapat memberikan akses lintas akun ke kebijakan KMS terkelola. AWS

AWS CloudFormation pertimbangan

- [Kebijakan berbasis sumber daya tidak mendukung deteksi drift](#). Jika Anda memperbarui kebijakan berbasis sumber daya di luar template AWS CloudFormation tumpukan, Anda harus memperbarui CloudFormation tumpukan dengan perubahan.
- Kebijakan berbasis sumber daya tidak mendukung perubahan di luar band. Jika Anda menambahkan, memperbarui, atau menghapus kebijakan di luar CloudFormation templat, perubahan tidak akan ditimpa jika tidak ada perubahan pada kebijakan di dalam templat.

Misalnya, katakan bahwa template Anda berisi kebijakan berbasis sumber daya yang kemudian Anda perbarui di luar templat. Jika Anda tidak membuat perubahan apa pun pada kebijakan di template, kebijakan yang diperbarui di DynamoDB tidak akan disinkronkan dengan kebijakan di templat.

Sebaliknya, katakan bahwa template Anda tidak berisi kebijakan berbasis sumber daya, tetapi Anda menambahkan kebijakan di luar templat. Kebijakan ini tidak akan dihapus dari DynamoDB selama Anda tidak menambahkannya ke template. Saat Anda menambahkan kebijakan ke template dan memperbarui tumpukan, kebijakan yang ada di DynamoDB akan diperbarui agar sesuai dengan yang ditentukan dalam templat.

Praktik terbaik kebijakan berbasis sumber daya

Topik ini menjelaskan praktik terbaik untuk menentukan izin akses untuk sumber daya DynamoDB Anda dan tindakan yang diizinkan pada sumber daya ini.

Sederhanakan kontrol akses ke sumber daya DynamoDB

Jika AWS Identity and Access Management prinsipal yang memerlukan akses ke sumber daya DynamoDB adalah bagian yang Akun AWS sama dengan pemilik sumber daya, kebijakan berbasis identitas IAM tidak diperlukan untuk setiap prinsipal. Kebijakan berbasis sumber daya yang melekat pada sumber daya yang diberikan sudah cukup. Jenis konfigurasi ini menyederhanakan kontrol akses.

Lindungi sumber daya DynamoDB Anda dengan kebijakan berbasis sumber daya

Untuk semua tabel dan aliran DynamoDB, buat kebijakan berbasis sumber daya untuk menerapkan kontrol akses untuk sumber daya ini. Kebijakan berbasis sumber daya memungkinkan Anda memusatkan izin di tingkat sumber daya, menyederhanakan kontrol akses ke tabel DynamoDB, indeks, dan aliran, serta mengurangi overhead administrasi. Jika tidak ada kebijakan berbasis sumber daya yang ditentukan untuk tabel atau aliran, akses ke tabel atau aliran akan ditolak secara implisit, kecuali kebijakan berbasis identitas yang terkait dengan prinsip IAM mengizinkan akses.

Terapkan izin hak istimewa paling sedikit

Saat Anda menetapkan izin dengan kebijakan berbasis sumber daya untuk sumber daya DynamoDB, berikan hanya izin yang diperlukan untuk melakukan tindakan. Anda melakukan ini dengan menentukan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, juga dikenal sebagai izin hak akses paling rendah. Anda dapat memulai dengan izin luas saat menjelajahi izin yang diperlukan untuk beban kerja atau kasus penggunaan Anda. Saat kasus penggunaan Anda matang, Anda dapat bekerja untuk mengurangi izin yang Anda berikan untuk bekerja menuju hak istimewa yang paling sedikit.

Menganalisis aktivitas akses lintas akun untuk menghasilkan kebijakan hak istimewa paling sedikit

IAM Access Analyzer melaporkan akses lintas akun ke entitas eksternal yang ditentukan dalam kebijakan berbasis sumber daya, dan memberikan visibilitas untuk membantu Anda menyempurnakan izin dan menyesuaikan diri dengan hak istimewa yang paling sedikit. Untuk informasi selengkapnya tentang pembuatan kebijakan, lihat pembuatan [kebijakan IAM Access Analyzer](#).

Gunakan IAM Access Analyzer untuk menghasilkan kebijakan hak istimewa paling sedikit

Untuk hanya memberikan izin yang diperlukan untuk melakukan tugas, Anda dapat membuat kebijakan berdasarkan aktivitas akses yang masuk AWS CloudTrail. IAM Access Analyzer menganalisis layanan dan tindakan yang digunakan kebijakan Anda.

Perlindungan data di DynamoDB

Amazon DynamoDB menyediakan infrastruktur penyimpanan yang sangat tahan lama yang dirancang untuk penyimpanan data penting dan primer. Objek disimpan secara redundan pada sejumlah perangkat di beberapa fasilitas di Wilayah Amazon DynamoDB.

DynamoDB melindungi data pengguna yang disimpan saat istirahat dan juga data dalam transit antara klien lokal dan DynamoDB, dan antara DynamoDB dan sumber daya lain dalam Wilayah yang sama. AWS AWS

Topik

- [Enkripsi DynamoDB saat diam](#)
- [Perlindungan data di DynamoDB Accelerator](#)
- [Privasi lalu lintas jaringan internet](#)

Enkripsi DynamoDB saat diam

Semua data pengguna yang disimpan di Amazon DynamoDB dienkripsi sepenuhnya saat diam. Enkripsi DynamoDB saat diam memberikan keamanan yang ditingkatkan dengan mengenkripsi semua data diam Anda menggunakan kunci enkripsi yang disimpan di [AWS Key Management Service \(AWS KMS\)](#). Fungsi ini membantu mengurangi beban operasional dan kompleksitas yang terlibat dalam melindungi data sensitif. Dengan enkripsi saat diam, Anda dapat membuat aplikasi yang sensitif terhadap keamanan yang memenuhi persyaratan kepatuhan dan peraturan enkripsi yang ketat.

Enkripsi DynamoDB saat diam menyediakan lapisan tambahan perlindungan data dengan selalu mengamankan data Anda dalam tabel terenkripsi, termasuk kunci primer, indeks sekunder lokal dan global, aliran, tabel global, cadangan, dan kluster DynamoDB Accelerator (DAX) setiap kali data disimpan dalam media yang tahan lama. Kebijakan organisasi, peraturan industri atau pemerintah, dan persyaratan kepatuhan kerap mewajibkan penggunaan enkripsi saat diam untuk meningkatkan keamanan data aplikasi Anda.

Enkripsi saat istirahat terintegrasi dengan AWS KMS untuk mengelola kunci enkripsi yang digunakan untuk mengenkripsi tabel Anda. Untuk informasi selengkapnya tentang jenis dan status kunci, lihat [konsep AWS Key Management Service](#) di Panduan Developer AWS Key Management Service .

Saat membuat tabel baru, Anda dapat memilih salah satu AWS KMS key jenis berikut untuk mengenkripsi tabel Anda. Anda dapat beralih di antara jenis-jenis kunci ini kapan saja.

- Kunci milik AWS — Jenis enkripsi default. Kuncinya dimiliki oleh DynamoDB (tanpa biaya tambahan).
- Kunci yang dikelola AWS — Kunci disimpan di akun Anda dan dikelola oleh AWS KMS (AWS KMS dikenakan biaya).
- Kunci yang dikelola pelanggan – Kunci disimpan di akun Anda serta dibuat, dimiliki, dan dikelola oleh Anda. Anda memiliki kontrol penuh atas kunci KMS (AWS KMS dikenakan biaya).

Untuk informasi selengkapnya tentang jenis kunci, lihat [Kunci dan AWS kunci pelanggan](#).

Note

- Saat membuat klaster DAX baru dengan enkripsi saat diam diaktifkan, Kunci yang dikelola AWS akan digunakan untuk mengenkripsi data diam di klaster.
- Jika tabel Anda memiliki kunci urutan, beberapa kunci urutan yang menandai batas kisaran disimpan dalam plaintext di metadata tabel.

Ketika Anda mengakses tabel terenkripsi, DynamoDB mendekripsi data tabel secara transparan. Anda tidak perlu mengubah kode atau aplikasi apa pun untuk menggunakan atau mengelola tabel terenkripsi. DynamoDB terus memberikan latensi milidetik satu digit yang sama yang Anda harapkan, dan semua kueri DynamoDB bekerja dengan lancar pada data terenkripsi Anda.

Anda dapat menentukan kunci enkripsi saat membuat tabel baru atau mengganti kunci enkripsi pada tabel yang ada dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau Amazon DynamoDB API. Untuk mempelajari caranya, lihat [Mengelola tabel yang dienkripsi di DynamoDB](#).

Enkripsi saat istirahat menggunakan Kunci milik AWS ditawarkan tanpa biaya tambahan. Namun, AWS KMS biaya berlaku untuk Kunci yang dikelola AWS dan untuk kunci yang dikelola pelanggan. Untuk informasi selengkapnya tentang harga, silakan lihat [harga AWS KMS](#).

Enkripsi DynamoDB saat istirahat tersedia di AWS semua Wilayah, termasuk AWS Wilayah China (Beijing) AWS dan China (Ningxia) dan AWS GovCloud Wilayah (AS). Untuk informasi selengkapnya, lihat [Enkripsi saat diam: Cara kerjanya](#) dan [Catatan penggunaan enkripsi DynamoDB saat diam](#).

Enkripsi saat diam: Cara kerjanya

Enkripsi Amazon DynamoDB saat diam mengenkripsi data Anda menggunakan Advanced Encryption Standard 256-bit (AES-256), yang membantu mengamankan data Anda dari akses tidak sah ke penyimpanan dasar.

Enkripsi saat istirahat terintegrasi dengan AWS Key Management Service (AWS KMS) untuk mengelola kunci enkripsi yang digunakan untuk mengenkripsi tabel Anda.

Note

Pada Mei 2022, AWS KMS mengubah jadwal rotasi Kunci yang dikelola AWS dari setiap tiga tahun (sekitar 1.095 hari) menjadi setiap tahun (sekitar 365 hari).

Baru Kunci yang dikelola AWS secara otomatis diputar satu tahun setelah dibuat, dan kira-kira setiap tahun setelahnya.

Yang Kunci yang dikelola AWS ada secara otomatis diputar satu tahun setelah rotasi terbaru mereka, dan setiap tahun setelahnya.

Kunci milik AWS

Kunci milik AWS tidak disimpan di AWS akun Anda. Mereka adalah bagian dari kumpulan kunci KMS yang AWS memiliki dan mengelola untuk digunakan di beberapa AWS akun. AWS Layanan dapat digunakan Kunci milik AWS untuk melindungi data Anda. Kunci milik AWS digunakan oleh DynamoDB diputar setiap tahun (sekitar 365 hari).

Anda tidak dapat melihat, mengelola, atau menggunakan Kunci milik AWS, atau mengaudit penggunaannya. Namun, Anda tidak perlu melakukan hal apa pun atau mengubah program apa pun untuk melindungi kunci yang mengenkripsi data Anda.

Anda tidak dikenakan biaya bulanan atau biaya penggunaan untuk penggunaan Kunci milik AWS, dan mereka tidak dihitung terhadap AWS KMS kuota untuk akun Anda.

Kunci yang dikelola AWS

Kunci yang dikelola AWS adalah kunci KMS di akun Anda yang dibuat, dikelola, dan digunakan atas nama Anda oleh AWS layanan yang terintegrasi dengannya AWS KMS. Anda dapat melihat Kunci yang dikelola AWS di akun Anda, melihat kebijakan kuncinya, dan mengaudit penggunaannya di log AWS CloudTrail . Namun, Anda tidak dapat mengelola kunci KMS ini atau mengubah izinnya.

Enkripsi saat istirahat secara otomatis terintegrasi dengan AWS KMS untuk mengelola untuk DynamoDB `aws/dynamodb ()` yang digunakan Kunci yang dikelola AWS untuk mengenkripsi tabel Anda. Jika Kunci yang dikelola AWS tidak ada saat Anda membuat tabel DynamoDB terenkripsi, AWS KMS secara otomatis membuat kunci baru untuk Anda. Kunci ini digunakan dengan tabel terenkripsi yang dibuat di masa depan. AWS KMS menggabungkan perangkat keras dan perangkat lunak yang aman dan sangat tersedia untuk menyediakan sistem manajemen kunci yang diskalakan untuk cloud.

Untuk informasi selengkapnya tentang mengelola izin Kunci yang dikelola AWS, lihat [Mengotorisasi penggunaan Kunci yang dikelola AWS dalam Panduan AWS Key Management Service](#) Pengembang.

Kunci yang dikelola pelanggan

Kunci yang dikelola pelanggan adalah kunci KMS di AWS akun Anda yang Anda buat, miliki, dan kelola. Anda memiliki kontrol penuh atas KMS ini, termasuk membangun dan memelihara kebijakan kuncinya, kebijakan IAM, dan pemberian izin; mengaktifkan dan menonaktifkannya; merotasi materi kriptografinya; menambahkan tag; membuat alias yang merujuk kepadanya; dan menjadwalkan penghapusan KMS tersebut. Untuk informasi selengkapnya tentang mengelola izin kunci yang dikelola pelanggan, lihat [Kebijakan kunci yang dikelola pelanggan](#).

Saat Anda menentukan kunci yang dikelola pelanggan sebagai kunci enkripsi tingkat tabel, tabel DynamoDB, indeks sekunder lokal dan global, dan aliran dienkrpsi dengan kunci yang sama dengan yang dikelola pelanggan. Cadangan sesuai permintaan dienkrpsi dengan kunci enkripsi tingkat tabel yang ditentukan pada saat cadangan dibuat. Memperbarui kunci enkripsi tingkat tabel tidak mengubah kunci enkripsi yang terkait dengan cadangan sesuai permintaan yang ada.

Mengatur status kunci yang dikelola pelanggan ke nonaktif atau menjadwalkannya untuk dihapus akan membuat semua pengguna dan layanan DynamoDB tidak dapat mengenkripsi atau mendekripsi data serta melakukan operasi baca dan tulis pada tabel. DynamoDB harus memiliki akses ke kunci enkripsi Anda untuk memastikan bahwa Anda dapat terus mengakses tabel Anda dan untuk mencegah kehilangan data.

Jika Anda menonaktifkan kunci yang dikelola pelanggan atau menjadwalkan penghapusannya, status tabel Anda menjadi Tidak dapat diakses. Untuk memastikan bahwa Anda dapat terus bekerja dengan tabel, Anda harus memberikan akses DynamoDB ke kunci enkripsi yang ditentukan dalam waktu tujuh hari. Segera setelah layanan mendeteksi bahwa kunci enkripsi Anda tidak dapat diakses, DynamoDB mengirimkan pemberitahuan email untuk mengingatkan Anda.

 Note

- Jika kunci yang dikelola pelanggan tetap tidak dapat diakses untuk layanan DynamoDB selama lebih dari tujuh hari, tabel diarsipkan dan tidak dapat diakses lagi. DynamoDB menciptakan cadangan sesuai permintaan atas tabel Anda, dan Anda menerima tagihan terkait cadangan tersebut. Anda dapat menggunakan cadangan sesuai permintaan ini untuk memulihkan data Anda ke tabel baru. Untuk memulai pemulihan, kunci yang dikelola pelanggan terakhir pada tabel harus diaktifkan, dan DynamoDB harus dapat mengaksesnya.
- Jika kunci yang dikelola pelanggan yang digunakan untuk mengenkripsi replika tabel global tidak dapat diakses, DynamoDB akan menghapus replika ini dari grup replikasi. Replika ini tidak akan dihapus dan replikasi dari dan ke wilayah ini akan dihentikan, 20 jam setelah mendeteksi bahwa kunci yang dikelola pelanggan tidak dapat diakses.

Untuk informasi selengkapnya, lihat [mengaktifkan kunci](#) dan [menghapus kunci](#).

Catatan tentang penggunaan Kunci yang dikelola AWS

Amazon DynamoDB tidak dapat membaca data tabel Anda kecuali memiliki akses ke kunci KMS yang disimpan di akun Anda. AWS KMS DynamoDB menggunakan enkripsi amplop dan hierarki kunci untuk mengenkripsi data. Kunci AWS KMS enkripsi Anda digunakan untuk mengenkripsi kunci root hierarki kunci ini. Untuk informasi lebih lanjut, lihat [Enkripsi amplop](#) di Panduan Developer AWS Key Management Service .

Anda dapat menggunakan AWS CloudTrail dan Amazon CloudWatch Logs untuk melacak permintaan yang DynamoDB kirimkan atas nama AWS KMS Anda. Untuk informasi selengkapnya, lihat [Memantau interaksi DynamoDB dengan AWS KMS](#) di Panduan Developer AWS Key Management Service .

DynamoDB tidak AWS KMS memanggil setiap operasi DynamoDB. Kuncinya disegarkan setiap 5 menit sekali per penelepon dengan lalu lintas aktif.

Pastikan bahwa Anda telah mengonfigurasi SDK untuk menggunakan kembali koneksi. Jika tidak, Anda akan mengalami latensi dari DynamoDB harus membangun kembali entri AWS KMS cache baru untuk setiap operasi DynamoDB. Selain itu, Anda mungkin harus menghadapi CloudTrail biaya AWS KMS dan biaya yang lebih tinggi. Misalnya, untuk melakukannya menggunakan SDK Node.js, Anda dapat membuat agen HTTPS baru dengan `keepAlive` diaktifkan. Untuk informasi

selengkapnya, lihat [Mengonfigurasi keepAlive di Node.js](#) di Panduan Developer AWS SDK for JavaScript .

Catatan penggunaan enkripsi DynamoDB saat diam

Pertimbangkan hal berikut saat Anda menggunakan enkripsi saat diam di Amazon DynamoDB.

Semua data tabel dienkripsi

Enkripsi di sisi server saat diam diaktifkan pada semua data tabel DynamoDB dan tidak dapat dinonaktifkan. Anda tidak dapat mengenkripsi subset item saja dalam tabel.

Enkripsi saat diam hanya mengenkripsi data saat statis (saat diam) pada media penyimpanan persisten. Jika keamanan data menjadi perhatian atas data bergerak atau data dalam penggunaan, Anda mungkin perlu melakukan tindakan tambahan:

- **Data bergerak:** Semua data Anda di DynamoDB dienkripsi saat bergerak. Secara default, komunikasi ke dan dari DynamoDB menggunakan protokol HTTPS, yang melindungi lalu lintas jaringan menggunakan enkripsi Secure Sockets Layer (SSL)/Keamanan Lapisan Pengangkutan (TLS).
- **Data dalam penggunaan:** Melindungi data Anda sebelum mengirimkannya ke DynamoDB menggunakan enkripsi di sisi klien. Untuk informasi selengkapnya, lihat [Enkripsi di sisi klien dan di sisi server](#) dalam Panduan Developer Amazon DynamoDB Encryption Client.

Anda dapat menggunakan aliran dengan tabel terenkripsi. Aliran DynamoDB selalu dienkripsi dengan kunci enkripsi tingkat tabel. Untuk informasi selengkapnya, lihat [Ubah tangkapan data DynamoDB Streams](#).

Cadangan DynamoDB dienkripsi, dan tabel yang dipulihkan dari cadangan juga diaktifkan enkripsinya. Anda dapat menggunakan Kunci milik AWS, Kunci yang dikelola AWS, atau kunci yang dikelola pelanggan untuk mengenkripsi data cadangan Anda. Untuk informasi selengkapnya, lihat [Menggunakan cadangan Sesuai Permintaan dan DynamoDB Permintaan](#).

Indeks sekunder lokal dan indeks sekunder global dienkripsi menggunakan kunci yang sama seperti tabel dasar.

Jenis enkripsi

Note

Kunci yang dikelola pelanggan tidak didukung di Tabel Global Versi 2017. Jika ingin menggunakan kunci yang dikelola pelanggan di Tabel Global DynamoDB, Anda perlu meningkatkan tabel ke Tabel Global Versi 2019, lalu mengaktifkannya.

Pada AWS Management Console, jenis enkripsi adalah KMS ketika Anda menggunakan Kunci yang dikelola AWS atau kunci yang dikelola pelanggan untuk mengenkripsi data Anda. Jenis enkripsi adalah DEFAULT ketika Anda menggunakan Kunci milik AWS. Di Amazon DynamoDB API, jenis enkripsi KMS adalah saat Anda menggunakan atau kunci Kunci yang dikelola AWS yang dikelola pelanggan. Apabila tidak ada jenis enkripsi, data Anda dienkripsi menggunakan Kunci milik AWS. Anda dapat beralih antara Kunci milik AWS, Kunci yang dikelola AWS, dan kunci yang dikelola pelanggan pada waktu tertentu. Anda dapat menggunakan konsol, AWS Command Line Interface (AWS CLI), atau Amazon DynamoDB API untuk mengganti kunci enkripsi.

Perhatikan batasan berikut saat menggunakan kunci yang dikelola pelanggan:

- Anda tidak dapat menggunakan kunci yang dikelola pelanggan dengan kluster DynamoDB Accelerator (DAX). Untuk informasi selengkapnya, lihat [Enkripsi DAX saat istirahat](#).
- Anda dapat menggunakan kunci yang dikelola pelanggan untuk mengenkripsi tabel yang menggunakan transaksi. Namun, guna memastikan ketahanan untuk propagasi transaksi, salinan permintaan transaksi disimpan sementara oleh layanan dan dienkripsi menggunakan Kunci milik AWS. Data yang ditempatkan di tabel dan indeks sekunder Anda selalu dienkripsi saat diam menggunakan kunci yang dikelola pelanggan.
- Anda dapat menggunakan kunci yang dikelola pelanggan untuk mengenkripsi tabel yang menggunakan Contributor Insights. Namun, data yang dikirimkan ke Amazon CloudWatch dienkripsi dengan file. Kunci milik AWS
- Saat Anda beralih ke kunci terkelola pelanggan baru, pastikan untuk tetap mengaktifkan kunci asli hingga proses selesai. AWS masih memerlukan kunci asli untuk mendekripsi data sebelum mengenkripsi dengan kunci baru. Proses akan selesai ketika Status SSEDescription tabel DIAKTIFKAN dan KMS MasterKeyArn dari kunci terkelola pelanggan baru ditampilkan. Pada titik ini, kunci asli dapat dinonaktifkan atau dijadwalkan untuk dihapus.
- Setelah kunci yang dikelola pelanggan baru ditampilkan, tabel dan cadangan sesuai permintaan baru dienkripsi dengan kunci baru.

- Setiap cadangan sesuai permintaan yang ada tetap dienkripsi dengan kunci yang dikelola pelanggan yang digunakan saat cadangan tersebut dibuat. Anda akan membutuhkan kunci yang sama untuk memulihkan cadangan tersebut. Anda dapat mengidentifikasi kunci untuk periode ketika setiap cadangan dibuat dengan menggunakan DescribeBackup API untuk melihat SSEDescription cadangan tersebut.
- Jika Anda menonaktifkan kunci yang dikelola pelanggan atau menjadwalkannya untuk dihapus, data apa pun di DynamoDB Streams masih dikenakan masa aktif 24 jam. Data aktivitas yang tidak diambil memenuhi syarat untuk dipangkas saat berusia lebih dari 24 jam.
- Jika Anda menonaktifkan kunci yang dikelola pelanggan atau menjadwalkannya untuk dihapus, penghapusan Waktu untuk Beroperasi (TTL) berlanjut selama 30 menit. Penghapusan TTL ini terus dipancarkan ke DynamoDB Streams dan tunduk pada interval pemangkas/retensi standar.

Untuk informasi selengkapnya, lihat [mengaktifkan kunci](#) dan [menghapus kunci](#).

Menggunakan kunci KMS dan kunci data

Fitur enkripsi DynamoDB saat istirahat menggunakan AWS KMS key dan hierarki kunci data untuk melindungi data tabel Anda. DynamoDB menggunakan hierarki kunci yang sama untuk melindungi aliran, tabel global, dan cadangan DynamoDB ketika ditulis ke media tahan lama.

Kami menyarankan Anda merencanakan strategi enkripsi Anda sebelum menerapkan tabel Anda di DynamoDB. Jika Anda menyimpan data sensitif atau rahasia di DynamoDB, pertimbangkan untuk menyertakan enkripsi di sisi klien dalam paket Anda. Dengan cara ini, Anda dapat mengenkripsi data sedekat mungkin dengan asalnya, dan memastikan perlindungannya sepanjang siklus hidupnya. Untuk informasi selengkapnya, lihat dokumentasi [klien enkripsi DynamoDB](#).

AWS KMS key

Enkripsi saat diam melindungi tabel DynamoDB Anda di bawah AWS KMS key. Secara default, DynamoDB menggunakan [Kunci milik AWS](#), kunci enkripsi multi-penghuni yang dibuat dan dikelola di akun layanan DynamoDB. Namun Anda dapat mengenkripsi tabel DynamoDB Anda di bawah [kunci yang dikelola pelanggan](#) untuk DynamoDB (aws/dynamodb) di Akun AWS Anda. Anda dapat memilih kunci KMS yang berbeda untuk setiap tabel. Kunci KMS yang Anda pilih untuk tabel juga digunakan untuk mengenkripsi indeks sekunder lokal dan global, aliran, dan cadangan.

Anda memilih kunci KMS untuk tabel saat membuat atau memperbarui tabel. Anda dapat mengubah kunci KMS untuk tabel kapan saja, baik di konsol DynamoDB atau dengan

menggunakan operasi. [UpdateTable](#) Proses pengalihan kunci berjalan tanpa hambatan dan tidak memerlukan waktu henti atau menurunkan layanan.

⚠ Important

DynamoDB hanya mendukung [kunci KMS simetris](#). Anda tidak dapat menggunakan [kunci KMS asimetris](#) untuk mengenkripsi tabel DynamoDB Anda.

Gunakan kunci yang dikelola pelanggan untuk mendapatkan fitur berikut:

- Anda membuat dan mengelola kunci KMS, termasuk menetapkan [kebijakan kunci](#), [kebijakan IAM](#), dan [pemberian izin](#) untuk mengontrol akses ke kunci KMS. Anda dapat [mengaktifkan dan menonaktifkan](#) kunci KMS, mengaktifkan dan menonaktifkan [rotasi kunci otomatis](#), dan [menghapus kunci KMS](#) ketika sudah tidak digunakan.
- Anda dapat menggunakan kunci yang dikelola pelanggan dengan [material kunci yang diimpor](#) atau kunci yang dikelola pelanggan di [penyimpanan kunci kustom](#) yang Anda miliki dan kelola.
- [Anda dapat mengaudit enkripsi dan dekripsi tabel DynamoDB Anda dengan memeriksa panggilan API DynamoDB ke dalam log. AWS KMSAWS CloudTrail](#)

Gunakan Kunci yang dikelola AWS jika Anda memerlukan salah satu fitur berikut:

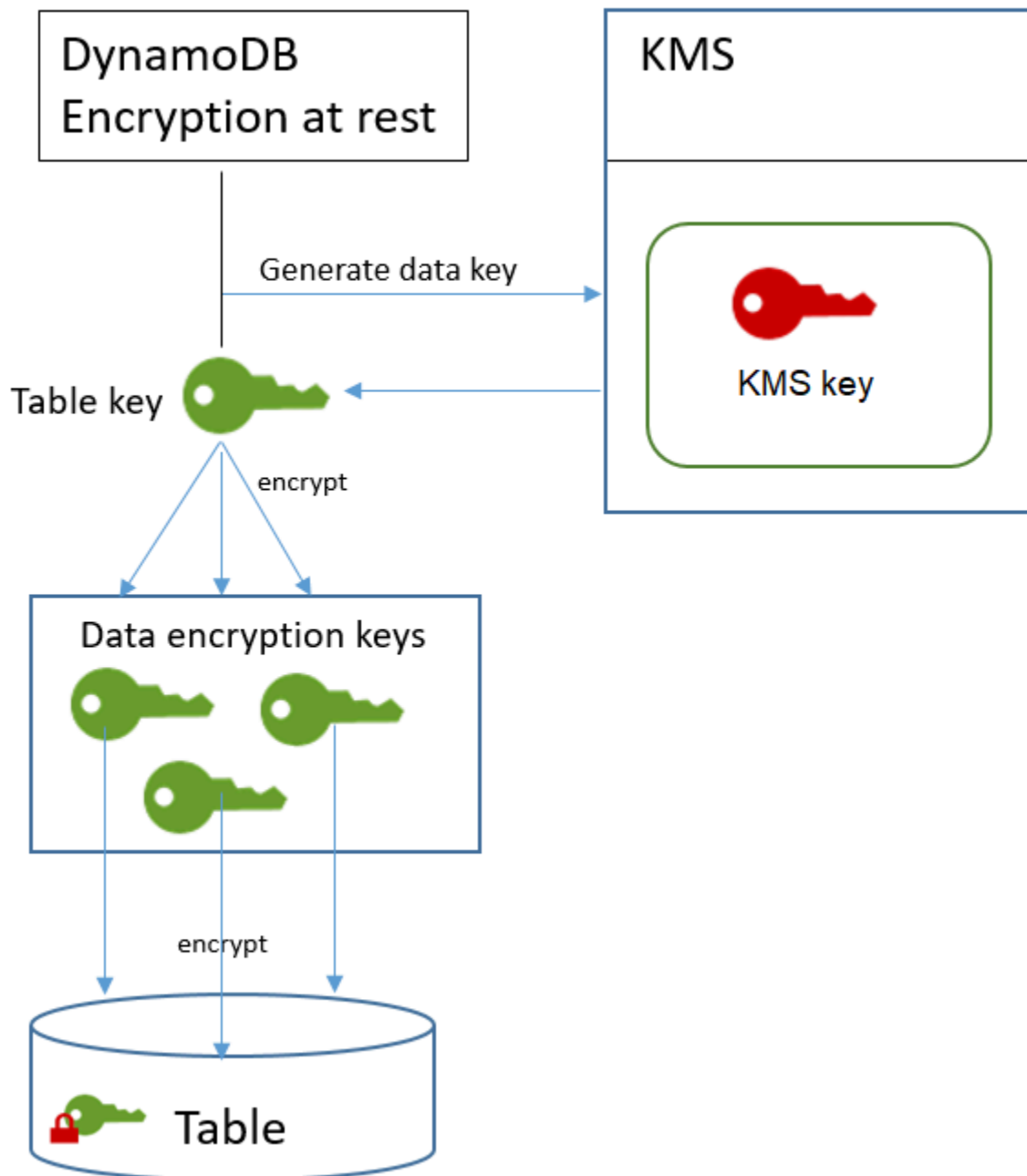
- Anda dapat [melihat kunci KMS](#) dan [melihat kebijakan kuncinya](#). (Anda tidak dapat mengubah kebijakan kunci.)
- [Anda dapat mengaudit enkripsi dan dekripsi tabel DynamoDB Anda dengan memeriksa panggilan API DynamoDB ke dalam log. AWS KMSAWS CloudTrail](#)

Namun, Kunci milik AWS ini gratis dan penggunaannya tidak dihitung terhadap [AWS KMS sumber daya atau kuota permintaan](#). Kunci yang dikelola pelanggan dan Kunci yang dikelola AWS [dikenakan biaya](#) untuk setiap panggilan API dan AWS KMS kuota berlaku untuk kunci KMS ini.

Kunci tabel

DynamoDB menggunakan kunci KMS untuk tabel guna [menghasilkan](#) dan mengenkripsi [kunci data](#) unik untuk tabel tersebut, yang dikenal sebagai kunci tabel. Kunci tabel bertahan sepanjang masa hidup tabel terenkripsi.

Kunci tabel digunakan sebagai kunci enkripsi kunci. DynamoDB menggunakan kunci tabel ini untuk melindungi kunci enkripsi data yang digunakan untuk mengenkripsi data tabel. DynamoDB menghasilkan kunci enkripsi data yang unik untuk setiap struktur dasar dalam tabel, tetapi beberapa item tabel mungkin dilindungi oleh kunci enkripsi data yang sama.



Saat pertama kali mengakses tabel terenkripsi, DynamoDB mengirimkan permintaan untuk menggunakan kunci KMS AWS KMS untuk mendekripsi kunci tabel. Kemudian, layanan ini menggunakan kunci tabel plaintext untuk mendekripsi kunci enkripsi data, dan menggunakan kunci enkripsi data plaintext untuk mendekripsi data tabel.

DynamoDB menyimpan dan menggunakan kunci tabel dan kunci enkripsi data di luar. AWS KMS Layanan ini melindungi semua kunci dengan enkripsi [Advanced Encryption Standard \(AES\)](#) dan

kunci enkripsi 256-bit. Kemudian, layanan ini menyimpan kunci terenkripsi dengan data terenkripsi agar kunci tersebut tersedia untuk mendekripsi data tabel sesuai permintaan.

Jika Anda mengubah kunci KMS untuk tabel Anda, DynamoDB menghasilkan kunci tabel baru. Kemudian, layanan ini menggunakan kunci tabel baru untuk melakukan enkripsi ulang kunci enkripsi data.

Melakukan cache pada kunci tabel

Untuk menghindari panggilan AWS KMS untuk setiap operasi DynamoDB, DynamoDB menyimpan tombol tabel plaintext untuk setiap pemanggil dalam memori. Jika DynamoDB mendapat permintaan untuk kunci tabel cache setelah lima menit tidak aktif, ia akan mengirimkan permintaan baru AWS KMS untuk mendekripsi kunci tabel. Panggilan ini akan menangkap setiap perubahan yang dibuat pada kebijakan akses kunci KMS di AWS KMS atau AWS Identity and Access Management (IAM) sejak permintaan terakhir untuk mendekripsi kunci tabel.

Mengotorisasi penggunaan kunci KMS Anda

Jika Anda menggunakan [kunci yang dikelola pelanggan](#) atau [Kunci yang dikelola AWS](#) di akun Anda untuk melindungi tabel DynamoDB Anda, kebijakan pada kunci KMS tersebut harus memberikan izin kepada DynamoDB untuk menggunakannya atas nama Anda. Konteks otorisasi pada Kunci yang dikelola AWS for DynamoDB menyertakan kebijakan utamanya dan hibah yang mendelegasikan izin untuk menggunakannya.

Anda memiliki kontrol penuh atas kebijakan dan pemberian izin pada kunci yang dikelola pelanggan. Karena Kunci yang dikelola AWS ada di akun Anda, Anda dapat melihat kebijakan dan pemberian izinnya. Tetapi, karena dikelola oleh AWS, Anda tidak dapat mengubah kebijakan.

DynamoDB tidak memerlukan otorisasi tambahan untuk menggunakan default untuk melindungi tabel DynamoDB di tabel [Kunci milik AWS](#) Anda. Akun AWS

Topik

- [Kebijakan utama untuk sebuah Kunci yang dikelola AWS](#)
- [Kebijakan kunci untuk kunci yang dikelola pelanggan](#)
- [Menggunakan pemberian izin untuk mengotorisasi DynamoDB](#)

Kebijakan utama untuk sebuah Kunci yang dikelola AWS

Ketika DynamoDB menggunakan [Kunci yang dikelola AWS](#) untuk DynamoDB (aws/dynamodb) dalam operasi kriptografi, layanan ini melakukannya atas nama pengguna yang mengakses [sumber daya DynamoDB](#). Kebijakan utama pada Kunci yang dikelola AWS memberikan semua pengguna di akun izin untuk menggunakan Kunci yang dikelola AWS untuk operasi tertentu. Namun izin hanya diberikan ketika DynamoDB membuat permintaan atas nama pengguna. [ViaService Kondisi](#) dalam kebijakan kunci tidak mengizinkan pengguna untuk menggunakan Kunci yang dikelola AWS kecuali permintaan berasal dari layanan DynamoDB.

Kebijakan utama ini, seperti kebijakan semua Kunci yang dikelola AWS, ditetapkan oleh AWS. Anda tidak dapat mengubahnya, tetapi Anda dapat melihatnya kapan saja. Untuk detailnya, lihat [Melihat kebijakan kunci](#).

Pernyataan kebijakan dalam kebijakan kunci memiliki efek sebagai berikut:

- Izinkan pengguna di akun untuk menggunakan Kunci yang dikelola AWS for DynamoDB dalam operasi kriptografi ketika permintaan berasal dari DynamoDB atas nama mereka. Kebijakan ini juga memungkinkan pengguna untuk [membuat pemberian izin](#) untuk kunci KMS.
- Memungkinkan identitas IAM yang diotorisasi di akun untuk melihat properti Kunci yang dikelola AWS untuk DynamoDB dan [mencabut pemberian izin](#) yang memungkinkan DynamoDB menggunakan kunci KMS. DynamoDB menggunakan [pemberian izin](#) untuk operasi pemeliharaan berkelanjutan.
- Memungkinkan DynamoDB melakukan operasi read-only untuk menemukan DynamoDB untuk akun Kunci yang dikelola AWS Anda.

```
{
  "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" : "*"
} ]
}

```

Kebijakan kunci untuk kunci yang dikelola pelanggan

Saat Anda memilih [kunci yang dikelola pelanggan](#) untuk melindungi tabel DynamoDB, DynamoDB mendapat izin untuk menggunakan kunci KMS atas nama pengguna utama yang membuat pilihan. Pengguna utama tersebut, pengguna atau peran, harus memiliki izin pada kunci KMS yang dibutuhkan DynamoDB. Anda dapat memberikan izin ini dalam [kebijakan kunci](#), [kebijakan IAM](#), atau [pemberian izin](#).

Minimal, DynamoDB memerlukan izin berikut pada kunci yang dikelola pelanggan:

- [kms:Encrypt](#)
- [kms:Decrypt](#)
- [kms: ReEncrypt](#) * (untuk kms: ReEncryptFrom dan kms:) ReEncryptTo
- kms: GenerateDataKey * (untuk [kms: GenerateDataKey](#) dan [kms:](#)) GenerateDataKeyWithoutPlaintext

- [km: DescribeKey](#)
- [km: CreateGrant](#)

Sebagai contoh, kebijakan kunci berikut hanya menyediakan izin yang diperlukan. Kebijakan ini memiliki efek sebagai berikut:

- Memungkinkan DynamoDB untuk menggunakan kunci KMS dalam operasi kriptografi dan membuat pemberian izin, tetapi hanya ketika bertindak atas nama pengguna utama di akun yang memiliki izin untuk menggunakan DynamoDB. Jika pengguna utama yang disebutkan dalam pernyataan kebijakan tidak memiliki izin untuk menggunakan DynamoDB, panggilan gagal, bahkan ketika panggilan berasal dari layanan DynamoDB.
- Kunci ViaService kondisi [kms:](#) mengizinkan izin hanya jika permintaan berasal dari DynamoDB atas nama prinsipal yang tercantum dalam pernyataan kebijakan. Pengguna utama ini tidak dapat memanggil operasi ini secara langsung. Perhatikan bahwa nilai `kms:ViaService, dynamodb.*.amazonaws.com`, memiliki tanda bintang (*) di posisi Wilayah. [DynamoDB memerlukan izin untuk independen dari apa pun sehingga dapat membuat panggilan lintas Wilayah AWS wilayah untuk mendukung tabel global DynamoDB.](#)
- Memberikan administrator kunci KMS (pengguna yang dapat mengasumsikan peran `db-team`) akses hanya-baca ke kunci KMS dan izin untuk mencabut pemberian izin, termasuk [pemberian izin yang diperlukan DynamoDB](#) untuk melindungi tabel.

Sebelum menggunakan kebijakan kunci contoh, ganti prinsip contoh dengan prinsip aktual dari Anda. Akun AWS

```
{
  "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": "*"
}
]
```

Menggunakan pemberian izin untuk mengotorisasi DynamoDB

Selain kebijakan kunci, DynamoDB menggunakan pemberian izin untuk mengatur izin pada kunci yang dikelola pelanggan atau Kunci yang dikelola AWS untuk DynamoDB (aws/dynamodb). Untuk melihat hibah pada kunci KMS di akun Anda, gunakan operasi. [ListGrants](#) DynamoDB tidak memerlukan pemberian izin, atau izin tambahan apa pun, untuk menggunakan [Kunci milik AWS](#) untuk melindungi tabel Anda.

DynamoDB menggunakan izin yang diberikan ketika melakukan pemeliharaan sistem latar belakang dan tugas perlindungan data berkelanjutan. Layanan ini juga menggunakan pemberian izin untuk menghasilkan [kunci tabel](#).

Setiap pemberian izin berlaku spesifik pada sebuah tabel. Jika akun mencakup beberapa tabel yang dienkripsi di bawah kunci KMS yang sama, ada pemberian izin dari setiap jenis untuk setiap tabel.

Hibah dibatasi oleh [konteks enkripsi DynamoDB](#), yang mencakup nama tabel dan Akun AWS ID, dan itu termasuk izin untuk [pensiun hibah jika tidak lagi diperlukan](#).

Untuk membuat pemberian izin, DynamoDB harus memiliki izin untuk memanggil `CreateGrant` atas nama pengguna yang membuat tabel terenkripsi. Untuk Kunci yang dikelola AWS, DynamoDB `kms:CreateGrant` mendapat izin dari kebijakan [kunci](#), yang memungkinkan pengguna akun untuk [CreateGrant](#) memanggil kunci KMS hanya ketika DynamoDB membuat permintaan atas nama pengguna yang berwenang.

Kebijakan kunci juga dapat memungkinkan akun untuk [mencabut pemberian izin](#) pada kunci KMS. Namun, jika Anda mencabut pemberian izin pada tabel dienkripsi yang aktif, DynamoDB tidak akan mampu melindungi dan menjaga tabel tersebut.

Konteks enkripsi DynamoDB

[Konteks enkripsi](#) adalah seperangkat pasangan kunci-nilai yang berisi data non-rahasia yang arbitrer. Ketika Anda menyertakan konteks enkripsi dalam permintaan untuk mengenkripsi data, secara AWS KMS kriptografis mengikat konteks enkripsi ke data terenkripsi. Untuk mendekripsi data, Anda harus meneruskan konteks enkripsi yang sama.

DynamoDB menggunakan konteks enkripsi yang sama di semua operasi kriptografi. AWS KMS Jika Anda menggunakan [kunci yang dikelola pelanggan](#) atau [Kunci yang dikelola AWS](#) untuk melindungi tabel DynamoDB Anda, Anda dapat menggunakan konteks enkripsi untuk mengidentifikasi penggunaan kunci KMS dalam catatan dan log audit. Itu juga muncul dalam plaintext di log, seperti dan [AWS CloudTrail Amazon CloudWatch](#) Logs.

Konteks enkripsi juga dapat digunakan sebagai syarat untuk otorisasi dalam kebijakan dan pemberian izin. DynamoDB menggunakan konteks enkripsi untuk membatasi hibah [yang](#) memungkinkan akses ke kunci yang dikelola pelanggan Kunci yang dikelola AWS atau di akun dan wilayah Anda.

Dalam permintaannya AWS KMS, DynamoDB menggunakan konteks enkripsi dengan dua pasangan kunci-nilai.

```
"encryptionContextSubset": {
  "aws:dynamodb:tableName": "Books"
  "aws:dynamodb:subscriberId": "111122223333"
}
```

- Tabel – Pasangan kunci–nilai pertama mengidentifikasi tabel yang dienkripsi DynamoDB. Kuncinya adalah `aws:dynamodb:tableName`. Nilainya adalah nama tabel.

```
"aws:dynamodb:tableName": "<table-name>"
```

Sebagai contoh:

```
"aws:dynamodb:tableName": "Books"
```

- Akun – Pasangan kunci–nilai kedua mengidentifikasi Akun AWS. Kuncinya adalah `aws:dynamodb:subscriberId`. Nilainya adalah ID akun.

```
"aws:dynamodb:subscriberId": "<account-id>"
```

Sebagai contoh:

```
"aws:dynamodb:subscriberId": "111122223333"
```

Memantau interaksi DynamoDB dengan AWS KMS

Jika Anda menggunakan [kunci yang dikelola pelanggan](#) atau [Kunci yang dikelola AWS](#) untuk melindungi tabel DynamoDB, Anda dapat AWS CloudTrail menggunakan log untuk melacak permintaan yang dikirimkan DynamoDB atas nama Anda. AWS KMS

Permintaan `GenerateDataKey`, `Decrypt`, dan `CreateGrant` dibahas dalam bagian ini. Selain itu, DynamoDB menggunakan [DescribeKey](#) operasi untuk menentukan apakah kunci KMS yang Anda pilih ada di akun dan wilayah. Ini juga menggunakan [RetireGrant](#) operasi untuk menghapus hibah saat Anda menghapus tabel.

GenerateDataKey

Ketika Anda mengaktifkan enkripsi saat diam pada tabel, DynamoDB menciptakan kunci tabel unik. Ini mengirimkan [GenerateDataKey](#) permintaan untuk AWS KMS yang menentukan kunci KMS untuk tabel.

Peristiwa yang mencatat operasi `GenerateDataKey` serupa dengan peristiwa contoh berikut. Pengguna adalah akun layanan DynamoDB. Parameternya mencakup Amazon Resource Name (ARN) dari kunci KMS, penentu kunci yang memerlukan kunci 256-bit, dan [konteks enkripsi](#) yang mengidentifikasi tabel dan Akun AWS.

```
{
  "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"
}
```

Dekripsi

Ketika Anda mengakses tabel DynamoDB yang dienkripsi, DynamoDB perlu mendekripsi kunci tabel agar dapat mendekripsi kunci di bawahnya dalam hierarki. Layanan ini kemudian mendekripsi data dalam tabel. Untuk mendekripsi kunci tabel, DynamoDB mengirimkan permintaan Dekripsi [yang](#) menentukan kunci AWS KMS untuk tabel.

Peristiwa yang mencatat operasi Decrypt serupa dengan peristiwa contoh berikut. Pengguna adalah kepala sekolah Anda Akun AWS yang mengakses tabel. Parameter termasuk kunci tabel terenkripsi (sebagai gumpalan ciphertext) dan [konteks enkripsi](#) yang mengidentifikasi tabel dan file. Akun AWS AWS KMS memperoleh ID kunci KMS dari ciphertext.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAIQDTESTANDEXAMPLE: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": "AROAIQDT3HGFQZX4RY6RU",
        "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

Saat Anda menggunakan [kunci yang dikelola pelanggan](#) atau [Kunci yang dikelola AWS](#) untuk melindungi tabel DynamoDB Anda, DynamoDB menggunakan [pemberian izin](#) untuk mengizinkan layanan guna melakukan perlindungan dan pemeliharaan data berkelanjutan serta tugas ketahanan. Pemberian izin ini tidak diperlukan pada [Kunci milik AWS](#).

Pemberian izin yang dibuat DynamoDB spesifik pada sebuah tabel. Prinsip dalam [CreateGrant](#) permintaan adalah pengguna yang membuat tabel.

Peristiwa yang mencatat operasi CreateGrant serupa dengan peristiwa contoh berikut. Parameternya mencakup Amazon Resource Name (ARN) dari kunci KMS untuk tabel tersebut, pengguna utama yang izin dan pengguna utama yang dipensiunkan (layanan DynamoDB), serta operasi yang dicakup oleh pemberian izin. Hal ini juga mencakup kendala yang mengharuskan semua operasi enkripsi menggunakan [konteks enkripsi](#) yang ditentukan.

```

{
  "eventVersion": "1.05",
  "userIdentity":
  {
    "type": "AssumedRole",
    "principalId": "AROAIQDTESTANDEXAMPLE: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": "AROAIQDTESTANDEXAMPLE",
        "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"
}
```

Mengelola tabel yang dienkripsi di DynamoDB

Anda dapat menggunakan AWS Management Console atau AWS Command Line Interface (AWS CLI) untuk menentukan kunci enkripsi pada tabel baru dan memperbarui kunci enkripsi pada tabel yang ada di Amazon DynamoDB.

Topik


- [Menentukan kunci enkripsi untuk tabel baru](#)
- [Memperbarui kunci enkripsi](#)

Menentukan kunci enkripsi untuk tabel baru

Ikuti langkah berikut untuk menentukan kunci enkripsi pada tabel baru menggunakan konsol Amazon DynamoDB atau AWS CLI.


Membuat tabel yang dienkripsi (konsol)

1. [Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/.](https://console.aws.amazon.com/dynamodb/)
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih Buat Tabel. Untuk Nama tabel, masukkan **Music**. Untuk kunci primer, masukkan **Artist**, dan untuk kunci urutan, masukkan **SongTitle**, keduanya sebagai string.
4. Di Pengaturan, pastikan bahwa Sesuaikan pengaturan dipilih.

 Note

Jika Gunakan pengaturan default dipilih, tabel dienkripsi saat istirahat dengan tanpa Kunci milik AWS biaya tambahan.

- Di bawah Enkripsi saat istirahat, pilih jenis enkripsi - Kunci milik AWS Kunci yang dikelola AWS, atau kunci yang dikelola pelanggan.
 - Dimiliki oleh Amazon DynamoDB. AWS kunci yang dimiliki, khusus dimiliki dan dikelola oleh DynamoDB. Anda tidak dikenakan biaya tambahan untuk menggunakan kunci ini.
 - AWS kunci yang dikelola. Alias kunci: aws/dynamodb. Kunci disimpan di akun Anda dan dikelola oleh AWS Key Management Service (AWS KMS). AWS KMS dikenakan biaya.
 - Disimpan di akun Anda, serta dimiliki dan dikelola oleh Anda. Kunci yang dikelola pelanggan. Kunci disimpan di akun Anda dan dikelola oleh AWS Key Management Service (AWS KMS). AWS KMS dikenakan biaya.

 Note

Jika Anda memilih untuk memiliki dan mengelola kunci Anda sendiri, pastikan Kebijakan Kunci KMS diatur dengan tepat. Untuk informasi selengkapnya termasuk contoh, lihat [Kebijakan kunci untuk kunci yang dikelola pelanggan](#).

- Pilih Buat tabel untuk membuat tabel terenkripsi. Untuk mengonfirmasi jenis enkripsi, pilih detail tabel pada tab Ikhtisar dan tinjau bagian Detail tambahan.

Membuat tabel yang dienkripsi (AWS CLI)

Gunakan AWS CLI untuk membuat tabel dengan default Kunci milik AWS, kunci yang dikelola pelanggan Kunci yang dikelola AWS, atau kunci yang dikelola pelanggan untuk Amazon DynamoDB.

Untuk membuat tabel terenkripsi dengan default Kunci milik AWS

- Buat tabel Music yang dienkripsi seperti berikut.

```
aws dynamodb create-table \  
  --table-name Music \  
  --attribute-definitions \  
    AttributeName=Artist,AttributeType=S \  
  --key-scheme ENCRYPTED
```



```
AttributeType=S \
--key-schema \
AttributeType=HASH \
AttributeType=RANGE \
--provisioned-throughput \
ReadCapacityUnits=10,WriteCapacityUnits=5
```

Note

Tabel ini sekarang dienkripsi menggunakan default Kunci milik AWS di akun layanan DynamoDB.

Untuk membuat tabel terenkripsi dengan untuk Kunci yang dikelola AWS DynamoDB

- Buat tabel Music yang dienkripsi seperti berikut.

```
aws dynamodb create-table \
--table-name Music \
--attribute-definitions \
AttributeType=S \
AttributeType=S \
--key-schema \
AttributeType=HASH \
AttributeType=RANGE \
--provisioned-throughput \
ReadCapacityUnits=10,WriteCapacityUnits=5 \
--sse-specification Enabled=true,SSEType=KMS
```

Status SSEDescription tabel diatur menjadi ENABLED dan SSEType adalah KMS.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Untuk membuat tabel yang dienkripsi dengan kunci yang dikelola pelanggan untuk DynamoDB

- Buat tabel Music yang dienkripsi seperti berikut.

```
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
```

Status SSEDescription tabel diatur menjadi ENABLED dan SSEType adalah KMS.

```
"SSEDescription": {  
  "SSEType": "KMS",  
  "Status": "ENABLED",  
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-  
a123-ab1234a1b234",  
}
```


Memperbarui kunci enkripsi

Anda juga dapat menggunakan konsol DynamoDB atau AWS CLI untuk memperbarui kunci enkripsi tabel yang ada antara Kunci yang dikelola AWS, dan kunci Kunci milik AWS yang dikelola pelanggan kapan saja.

Memperbarui kunci enkripsi (konsol)

1. [Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/.](https://console.aws.amazon.com/dynamodb/)
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tabel yang akan Anda perbarui.
4. Pilih dropdown Tindakan, lalu pilih opsi Perbarui pengaturan.

5. Buka tab Pengaturan tambahan.
6. Di bawah Enkripsi, pilih Kelola enkripsi.
7. Pilih jenis enkripsi:
 - Dimiliki oleh Amazon DynamoDB. AWS KMS Kunci dimiliki dan dikelola oleh DynamoDB. Anda tidak dikenakan biaya tambahan untuk menggunakan kunci ini.
 - AWS kunci terkelola Alias kunci:aws/dynamodb. Kunci disimpan di akun Anda dan dikelola oleh AWS Key Management Service. (AWS KMS). AWS KMS dikenakan biaya.
 - Disimpan di akun Anda, serta dimiliki dan dikelola oleh Anda. Kunci disimpan di akun Anda dan dikelola oleh AWS Key Management Service. (AWS KMS). AWS KMS dikenakan biaya.

 Note

Jika Anda memilih untuk memiliki dan mengelola kunci Anda sendiri, pastikan Kebijakan Kunci KMS diatur dengan tepat. Untuk informasi selengkapnya, lihat [Kebijakan kunci untuk kunci yang dikelola pelanggan](#).

Lalu pilih Simpan untuk memperbarui tabel yang dienkripsi. Untuk mengonfirmasi jenis enkripsi, periksa detail tabel di bawah tab Ikhtisar.

Memperbarui kunci enkripsi (AWS CLI)

Contoh-contoh berikut menunjukkan cara memperbarui tabel yang dienkripsi menggunakan AWS CLI.

Untuk memperbarui tabel terenkripsi dengan default Kunci milik AWS

- Perbarui tabel Music yang dienkripsi, seperti pada contoh berikut.

```
aws dynamodb update-table \  
  --table-name Music \  
  --sse-specification Enabled=false
```

Note

Tabel ini sekarang dienkripsi menggunakan default Kunci milik AWS di akun layanan DynamoDB.

Untuk memperbarui tabel terenkripsi dengan untuk Kunci yang dikelola AWS DynamoDB

- Perbarui tabel Music yang dienkripsi, seperti pada contoh berikut.

```
aws dynamodb update-table \  
  --table-name Music \  
  --sse-specification Enabled=true
```

Status SSEDescription dari deskripsi tabel diatur menjadi ENABLED dan SSEType adalah KMS.

```
"SSEDescription": {  
  "SSEType": "KMS",  
  "Status": "ENABLED",  
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-  
a123-ab1234a1b234",  
}
```

Untuk memperbarui tabel yang dienkripsi dengan kunci yang dikelola pelanggan untuk DynamoDB

- Perbarui tabel Music yang dienkripsi, seperti pada contoh berikut.

```
aws dynamodb update-table \  
  --table-name Music \  
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-  
a123-ab1234a1b234
```

Status SSEDescription dari deskripsi tabel diatur menjadi ENABLED dan SSEType adalah KMS.

```
"SSEDescription": {  
  "SSEType": "KMS",
```

```
"Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Perlindungan data di DynamoDB Accelerator

Enkripsi Amazon DynamoDB Accelerator (DAX) saat diam menyediakan lapisan perlindungan data tambahan dengan membantu mengamankan data Anda dari akses tidak sah ke penyimpanan yang mendasari. Kebijakan organisasi, peraturan industri atau pemerintah, dan persyaratan kepatuhan mungkin memerlukan penggunaan enkripsi saat diam untuk melindungi data Anda. Anda dapat menggunakan enkripsi untuk meningkatkan keamanan data aplikasi Anda yang di-deploy di cloud.

Untuk informasi selengkapnya tentang perlindungan data di DAX, lihat [Enkripsi DAX saat istirahat](#).

Privasi lalu lintas jaringan internet

Sambungan dilindungi baik antara Amazon DynamoDB dan aplikasi lokal dan antara DynamoDB dan sumber daya lain dalam Wilayah yang sama. AWS AWS

Kebijakan yang diperlukan untuk titik akhir

Amazon DynamoDB menyediakan API [DescribeEndpoints](#) yang memungkinkan Anda menghitung informasi titik akhir regional. Untuk permintaan dari titik akhir VPC, kebijakan titik akhir IAM dan Cloud Privat Virtual (VPC) harus mengotorisasi panggilan API DescribeEndpoints untuk pengguna utama Identity and Access Management (IAM) yang meminta menggunakan tindakan dynamodb:DescribeEndpoints IAM. Jika tidak, akses ke API DescribeEndpoints akan ditolak. Langkah-langkah otorisasi kebijakan titik akhir IAM dan VPC DescribeEndpoints untuk panggilan API tidak berlaku saat Anda mengakses titik akhir publik DynamoDB.

Berikut adalah sebuah contoh kebijakan titik akhir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "(Include IAM Principals)",
      "Action": "dynamodb:DescribeEndpoints",
```

```
    "Resource": "*"
  }
]
}
```

Lalu lintas antara layanan dan aplikasi serta klien on-premise

Anda memiliki dua opsi konektivitas antara jaringan pribadi Anda dan AWS:

- **AWS Site-to-Site VPN Koneksi.** Untuk informasi selengkapnya, lihat [Apa itu AWS Site-to-Site VPN?](#) dalam Panduan Pengguna AWS Site-to-Site VPN .
- **AWS Direct Connect Koneksi.** Untuk informasi selengkapnya, lihat [Apa itu AWS Direct Connect?](#) dalam Panduan Pengguna AWS Direct Connect .

Akses ke DynamoDB melalui jaringan adalah AWS melalui API yang diterbitkan. Klien harus mendukung Keamanan Lapisan Pengangkutan (TLS) 1.2. Kami merekomendasikan TLS 1.3. Klien juga harus mendukung suite cipher dengan Perfect Forward Secrecy (PFS) seperti Ephemeral Diffie-Hellman (DHE) atau Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Sebagian besar sistem modern seperti Java 7 dan versi yang lebih baru support mode ini. Selain itu, Anda harus menandatangani permintaan menggunakan kunci akses ID dan kunci akses rahasia yang terkait dengan IAM pengguna utama, atau Anda dapat menggunakan [AWS Security Token Service \(STS\)](#) untuk membuat kredensial keamanan sementara guna menandatangani permintaan.

Lalu lintas antara sumber daya AWS di Wilayah yang sama

Titik akhir Amazon Virtual Private Cloud (Amazon VPC) untuk DynamoDB adalah entitas logis dalam VPC yang memungkinkan konektivitas hanya ke DynamoDB. Amazon VPC merutekan permintaan ke DynamoDB dan merutekan respons kembali ke VPC. Untuk informasi selengkapnya, lihat [Titik akhir VPC](#) di Panduan Pengguna Amazon VPC. Misalnya, kebijakan yang dapat Anda gunakan untuk mengontrol akses dari titik akhir VPC, lihat [Menggunakan Kebijakan IAM untuk mengontrol akses ke DynamoDB](#).

Note

Titik akhir Amazon VPC tidak dapat diakses melalui atau. AWS Site-to-Site VPN AWS Direct Connect

AWS Identity and Access Management (IAM)

AWS Identity and Access Management adalah AWS layanan yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator mengontrol siapa yang dapat diautentikasi (masuk) dan diotorisasi (memiliki izin) untuk menggunakan sumber daya Amazon DynamoDB dan DynamoDB Accelerator. Anda dapat menggunakan IAM untuk mengelola izin akses dan menerapkan kebijakan keamanan untuk Amazon DynamoDB dan DynamoDB Accelerator. IAM adalah AWS layanan yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- [Manajemen Identitas dan Akses untuk Amazon DynamoDB](#)
- [Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci](#)
- [Manajemen identitas dan akses di DynamoDB Accelerator](#)

Manajemen Identitas dan Akses untuk Amazon DynamoDB

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang dapat diautentikasi (masuk) dan diotorisasi (memiliki izin) untuk menggunakan sumber daya DynamoDB. IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- [Audiens](#)
- [Mengautentikasi dengan identitas](#)
- [Mengelola akses menggunakan kebijakan](#)
- [Cara kerja Amazon DynamoDB dengan IAM](#)
- [Contoh kebijakan berbasis identitas untuk Amazon DynamoDB](#)
- [Pemecahan masalah identitas dan akses Amazon DynamoDB](#)
- [Kebijakan IAM untuk mencegah pembelian kapasitas terpesan DynamoDB](#)

Audiens

Bagaimana Anda menggunakan AWS Identity and Access Management (IAM) berbeda, tergantung pada pekerjaan yang Anda lakukan di DynamoDB.

Pengguna layanan – Jika Anda menggunakan layanan DynamoDB untuk melakukan pekerjaan Anda, administrator Anda akan memberi Anda kredensial dan izin yang diperlukan. Saat Anda menggunakan lebih banyak fitur DynamoDB untuk melakukan pekerjaan Anda, izin tambahan mungkin diperlukan. Memahami cara akses dikelola dapat membantu Anda meminta izin yang tepat dari administrator Anda. Jika Anda tidak dapat mengakses fitur di DynamoDB, lihat [Pemecahan masalah identitas dan akses Amazon DynamoDB](#).

Administrator layanan – Jika Anda bertanggung jawab atas sumber daya DynamoDB di perusahaan Anda, Anda mungkin memiliki akses penuh ke DynamoDB. Tugas Anda adalah menentukan fitur dan sumber daya DynamoDB mana yang harus diakses oleh pengguna layanan Anda. Kemudian, Anda harus mengirimkan permintaan kepada administrator IAM Anda untuk mengubah izin pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep Basic IAM. Untuk mempelajari selengkapnya tentang bagaimana perusahaan Anda dapat menggunakan IAM dengan DynamoDB, lihat [Cara kerja Amazon DynamoDB dengan IAM](#).

Administrator IAM – Jika Anda adalah administrator IAM, Anda mungkin ingin mempelajari detail tentang cara menulis kebijakan untuk mengelola akses ke DynamoDB. Untuk melihat contoh kebijakan berbasis identitas DynamoDB yang dapat Anda gunakan di IAM, lihat [Contoh kebijakan berbasis identitas untuk Amazon DynamoDB](#).

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensial identitas Anda. Anda harus diautentikasi (masuk ke AWS) sebagai Pengguna root akun AWS, sebagai pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk AWS sebagai identitas federasi dengan menggunakan kredensial yang disediakan melalui sumber identitas. AWS IAM Identity Center Pengguna (IAM Identity Center), autentikasi masuk tunggal perusahaan Anda, dan kredensial Google atau Facebook Anda adalah contoh identitas federasi. Saat Anda masuk sebagai identitas gabungan, administrator Anda sebelumnya menyiapkan federasi identitas menggunakan peran IAM. Ketika Anda mengakses AWS dengan menggunakan federasi, Anda secara tidak langsung mengambil peran.

Bergantung pada jenis pengguna Anda, Anda dapat masuk ke AWS Management Console atau portal AWS akses. Untuk informasi selengkapnya tentang masuk AWS, lihat [Cara masuk ke Panduan AWS Sign-In Pengguna Anda Akun AWS](#).

Jika Anda mengakses AWS secara terprogram, AWS sediakan kit pengembangan perangkat lunak (SDK) dan antarmuka baris perintah (CLI) untuk menandatangani permintaan Anda secara kriptografis dengan menggunakan kredensial Anda. Jika Anda tidak menggunakan AWS alat, Anda harus menandatangani permintaan sendiri. Untuk informasi selengkapnya tentang penggunaan metode yang disarankan untuk menandatangani permintaan sendiri, lihat [Menandatangani permintaan AWS API](#) di Panduan Pengguna IAM.

Apa pun metode autentikasi yang digunakan, Anda mungkin diminta untuk menyediakan informasi keamanan tambahan. Misalnya, AWS merekomendasikan agar Anda menggunakan otentikasi multi-faktor (MFA) untuk meningkatkan keamanan akun Anda. Untuk mempelajari selengkapnya, lihat [Autentikasi multi-faktor](#) dalam Panduan Pengguna AWS IAM Identity Center dan [Menggunakan autentikasi multi-faktor \(MFA\) di AWS](#) dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya di akun. Identitas ini disebut pengguna Akun AWS root dan diakses dengan masuk dengan alamat email dan kata sandi yang Anda gunakan untuk membuat akun. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari Anda. Lindungi kredensial pengguna root Anda dan gunakan kredensial tersebut untuk melakukan tugas yang hanya dapat dilakukan pengguna root. Untuk daftar lengkap tugas yang mengharuskan Anda masuk sebagai pengguna root, lihat [Tugas yang memerlukan kredensial pengguna root](#) dalam Panduan Pengguna IAM.

Identitas terfederasi

Sebagai praktik terbaik, mewajibkan pengguna manusia, termasuk pengguna yang memerlukan akses administrator, untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS dengan menggunakan kredensi sementara.

Identitas federasi adalah pengguna dari direktori pengguna perusahaan Anda, penyedia identitas web, direktori Pusat Identitas AWS Directory Service, atau pengguna mana pun yang mengakses Layanan AWS dengan menggunakan kredensi yang disediakan melalui sumber identitas. Ketika identitas federasi mengakses Akun AWS, mereka mengambil peran, dan peran memberikan kredensyal sementara.

Untuk manajemen akses terpusat, kami sarankan Anda menggunakan AWS IAM Identity Center. Anda dapat membuat pengguna dan grup di Pusat Identitas IAM, atau Anda dapat menghubungkan dan menyinkronkan ke sekumpulan pengguna dan grup di sumber identitas Anda sendiri untuk digunakan di semua aplikasi Akun AWS dan aplikasi Anda. Untuk informasi tentang Pusat Identitas IAM, lihat [Apa itu Pusat Identitas IAM?](#) dalam Panduan Pengguna AWS IAM Identity Center .

Pengguna dan grup IAM

[Pengguna IAM](#) adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus untuk satu orang atau aplikasi. Jika memungkinkan, sebaiknya andalkan kredensial sementara daripada membuat pengguna IAM yang memiliki kredensial jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan khusus yang memerlukan kredensial jangka panjang dengan pengguna IAM, kami sarankan Anda merotasi kunci akses. Untuk informasi selengkapnya, lihat [Rotasikan kunci akses secara rutin untuk kasus penggunaan yang memerlukan kredensial jangka panjang](#) dalam Panduan Pengguna IAM.

[Grup IAM](#) adalah identitas yang menentukan kumpulan pengguna IAM. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin untuk beberapa pengguna sekaligus. Grup membuat izin lebih mudah dikelola untuk sekelompok besar pengguna. Misalnya, Anda dapat memiliki grup yang bernama IAMAdmins dan memberikan izin kepada grup tersebut untuk mengelola sumber daya IAM.

Pengguna berbeda dari peran. Pengguna secara unik terkait dengan satu orang atau aplikasi, tetapi peran dimaksudkan untuk dapat digunakan oleh siapa pun yang membutuhkannya. Pengguna memiliki kredensial jangka panjang permanen, tetapi peran memberikan kredensial sementara. Untuk mempelajari selengkapnya, silakan lihat [Kapan harus membuat pengguna IAM \(bukan peran\)](#) dalam Panduan Pengguna IAM.

Peran IAM

[Peran IAM](#) adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus. Peran ini mirip dengan pengguna IAM, tetapi tidak terkait dengan orang tertentu. Anda dapat mengambil peran IAM untuk sementara AWS Management Console dengan [beralih peran](#). Anda dapat mengambil peran dengan memanggil operasi AWS CLI atau AWS API atau dengan menggunakan URL kustom. Untuk informasi selengkapnya tentang cara menggunakan peran, lihat [Menggunakan peran IAM](#) dalam Panduan Pengguna IAM.

Peran IAM dengan kredensial sementara berguna dalam situasi berikut:

- Akses pengguna gabungan – Untuk menetapkan izin ke sebuah identitas gabungan, Anda dapat membuat peran dan menentukan izin untuk peran tersebut. Saat identitas terfederasi mengautentikasi, identitas tersebut akan dikaitkan dengan peran dan diberi izin yang ditentukan oleh peran tersebut. Untuk informasi tentang peran-peran untuk federasi, lihat [Membuat peran untuk Penyedia Identitas pihak ketiga](#) dalam Panduan Pengguna IAM. Jika Anda menggunakan Pusat Identitas IAM, Anda perlu mengonfigurasi set izin. Untuk mengontrol apa yang dapat diakses identitas Anda setelah identitas tersebut diautentikasi, Pusat Identitas IAM mengorelasikan izin yang diatur ke peran dalam IAM. Untuk informasi tentang rangkaian izin, lihat [Rangkaian izin](#) dalam Panduan Pengguna AWS IAM Identity Center .
- Izin pengguna IAM sementara – Pengguna atau peran IAM dapat mengambil peran IAM guna mendapatkan berbagai izin secara sementara untuk tugas tertentu.
- Akses lintas akun – Anda dapat menggunakan peran IAM untuk mengizinkan seseorang (pengguna utama tepercaya) dengan akun berbeda untuk mengakses sumber daya yang ada di akun Anda. Peran adalah cara utama untuk memberikan akses lintas akun. Namun, dengan beberapa Layanan AWS, Anda dapat melampirkan kebijakan secara langsung ke sumber daya (alih-alih menggunakan peran sebagai proxy). Untuk mempelajari perbedaan antara kebijakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat [Bagaimana peran IAM berbeda dari kebijakan berbasis sumber daya](#) dalam Panduan Pengguna IAM.
- Akses lintas layanan — Beberapa Layanan AWS menggunakan fitur lain Layanan AWS. Sebagai contoh, ketika Anda melakukan panggilan dalam suatu layanan, biasanya layanan tersebut menjalankan aplikasi di Amazon EC2 atau menyimpan objek di Amazon S3. Sebuah layanan mungkin melakukannya menggunakan izin prinsipal yang memanggil, menggunakan peran layanan, atau peran terkait layanan.
 - Sesi akses teruskan (FAS) — Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Saat Anda menggunakan beberapa layanan, Anda mungkin melakukan tindakan yang kemudian memulai tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan saat membuat permintaan FAS, lihat [Teruskan sesi akses](#).
- Peran layanan – Peran layanan adalah [peran IAM](#) yang diambil oleh layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, memodifikasi, dan menghapus

peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat [Membuat peran untuk mendelegasikan izin ke Layanan AWS](#) dalam Panduan pengguna IAM.

- Peran terkait layanan — Peran terkait layanan adalah jenis peran layanan yang ditautkan ke peran layanan. Layanan AWS Layanan dapat menggunakan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.
- Aplikasi yang berjalan di Amazon EC2 — Anda dapat menggunakan peran IAM untuk mengelola kredensi sementara untuk aplikasi yang berjalan pada instans EC2 dan membuat atau permintaan API. AWS CLI AWS Cara ini lebih dianjurkan daripada menyimpan kunci akses dalam instans EC2. Untuk menetapkan AWS peran ke instans EC2 dan membuatnya tersedia untuk semua aplikasinya, Anda membuat profil instance yang dilampirkan ke instance. Profil instans berisi peran dan memungkinkan program yang berjalan di instans EC2 mendapatkan kredensial sementara. Untuk informasi selengkapnya, lihat [Menggunakan peran IAM untuk memberikan izin ke aplikasi yang berjalan di instans Amazon EC2](#) dalam Panduan Pengguna IAM.

Untuk mempelajari apakah kita harus menggunakan peran IAM atau pengguna IAM, lihat [Kapan harus membuat peran IAM \(bukan pengguna\)](#) dalam Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan adalah objek AWS yang, ketika dikaitkan dengan identitas atau sumber daya, menentukan izinnya. AWS mengevaluasi kebijakan ini ketika prinsipal (pengguna, pengguna root, atau sesi peran) membuat permintaan. Izin dalam kebijakan dapat menentukan permintaan yang diizinkan atau ditolak. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang struktur dan konten dokumen kebijakan JSON, lihat [Gambaran umum kebijakan JSON](#) dalam Panduan Pengguna IAM.

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, pengguna utama manakah yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Untuk memberikan izin kepada pengguna untuk melakukan tindakan pada sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat menjalankan peran.

Kebijakan IAM mendefinisikan izin untuk suatu tindakan terlepas dari metode yang Anda gunakan untuk operasi. Sebagai contoh, anggap saja Anda memiliki kebijakan yang mengizinkan tindakan `iam:GetRole`. Pengguna dengan kebijakan tersebut bisa mendapatkan informasi peran dari AWS Management Console, API AWS CLI, atau AWS API.

Kebijakan berbasis identitas

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan pengguna dan peran, di sumber daya mana, dan dengan ketentuan apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat [Membuat kebijakan IAM](#) dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat dikategorikan lebih lanjut sebagai kebijakan inline atau kebijakan terkelola. Kebijakan inline disematkan langsung ke satu pengguna, grup, atau peran. Kebijakan terkelola adalah kebijakan mandiri yang dapat Anda lampirkan ke beberapa pengguna, grup, dan peran dalam Akun AWS. Kebijakan AWS terkelola mencakup kebijakan terkelola dan kebijakan yang dikelola pelanggan. Untuk mempelajari cara memilih antara kebijakan terkelola atau kebijakan inline, lihat [Memilih antara kebijakan terkelola dan kebijakan inline](#) dalam Panduan Pengguna IAM.

Kebijakan berbasis sumber daya

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya yang dilampiri kebijakan tersebut, kebijakan ini menentukan jenis tindakan yang dapat dilakukan oleh pengguna utama tertentu di sumber daya tersebut dan apa ketentuannya. Anda harus [menentukan pengguna utama](#) dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau Layanan AWS.

Kebijakan berbasis sumber daya merupakan kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola dari IAM dalam kebijakan berbasis sumber daya.

Daftar kontrol akses (ACL)

Daftar kontrol akses (ACL) mengendalikan pengguna utama mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL sama dengan kebijakan berbasis sumber daya, meskipun tidak menggunakan format dokumen kebijakan JSON.

Amazon S3, AWS WAF, dan Amazon VPC adalah contoh layanan yang mendukung ACL. Untuk mempelajari ACL selengkapnya, silakan lihat [Gambaran umum daftar kontrol akses \(ACL\)](#) di Panduan Developer Layanan Penyimpanan Ringkas Amazon.

Tipe kebijakan lain

AWS mendukung jenis kebijakan tambahan yang kurang umum. Tipe-tipe kebijakan ini dapat mengatur izin maksimum yang diberikan kepada Anda berdasarkan tipe kebijakan yang lebih umum.

- **Batasan izin** – Batasan izin adalah fitur lanjutan di mana Anda menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM (pengguna atau peran IAM). Anda dapat menetapkan batasan izin untuk suatu entitas. Izin yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas milik entitas dan batasan izinnya. Kebijakan berbasis sumber daya yang menentukan pengguna atau peran dalam bidang `Principal` tidak dibatasi oleh batasan izin. Penolakan secara eksplisit terhadap salah satu kebijakan ini akan mengesampingkan izin tersebut. Untuk informasi selengkapnya tentang batasan izin, lihat [Batasan izin untuk entitas IAM](#) dalam Panduan Pengguna IAM.
- **Kebijakan kontrol layanan (SCP)** — SCP adalah kebijakan JSON yang menentukan izin maksimum untuk organisasi atau unit organisasi (OU) di AWS Organizations. AWS Organizations adalah layanan untuk mengelompokkan dan mengelola secara terpusat beberapa Akun AWS yang dimiliki bisnis Anda. Jika Anda mengaktifkan semua fitur di sebuah organisasi, maka Anda dapat menerapkan kebijakan kontrol layanan (SCP) ke salah satu atau semua akun Anda. SCP membatasi izin untuk entitas di akun anggota, termasuk masing-masing. Pengguna root akun AWS Untuk informasi selengkapnya tentang Organisasi dan SCP, lihat [Cara kerja SCP](#) dalam Panduan Pengguna AWS Organizations .
- **Kebijakan sesi** – Kebijakan sesi adalah kebijakan lanjutan yang Anda teruskan sebagai parameter saat Anda membuat sesi sementara secara terprogram untuk peran atau pengguna gabungan. Izin sesi yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas pengguna atau peran dan kebijakan sesi. Izin juga bisa datang dari kebijakan berbasis sumber daya. Penolakan eksplisit dalam salah satu kebijakan ini membatalkan izin. Untuk informasi selengkapnya, lihat [Kebijakan sesi](#) dalam Panduan Pengguna IAM.

Beberapa jenis kebijakan

Ketika beberapa jenis kebijakan berlaku untuk sebuah permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat [Logika evaluasi kebijakan](#) di Panduan Pengguna IAM.

Cara kerja Amazon DynamoDB dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke DynamoDB, pelajari fitur IAM apakah yang terdapat untuk digunakan pada DynamoDB.

Fitur-fitur IAM yang dapat Anda gunakan dengan Amazon DynamoDB

Fitur IAM	Dukungan DynamoDB
Kebijakan berbasis identitas	Ya
Kebijakan berbasis sumber daya	Ya
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
Kunci persyaratan kebijakan	Ya
ACL	Tidak
ABAC (tanda dalam kebijakan)	Parsial
Kredensial sementara	Ya
Izin pengguna utama	Ya
Peran layanan	Ya
Peran terkait layanan	Tidak

Untuk mendapatkan tampilan tingkat tinggi tentang cara kerja DynamoDB dan layanan AWS lainnya dengan sebagian besar fitur IAM, [AWS lihat layanan yang bekerja dengan IAM](#) di Panduan Pengguna IAM.

Kebijakan berbasis identitas untuk DynamoDB

Mendukung kebijakan berbasis identitas	Ya
--	----

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan pengguna dan peran, di sumber daya mana, dan dengan ketentuan apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat [Membuat kebijakan IAM](#) dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan tindakan dan sumber daya yang diizinkan atau ditolak, serta ketentuan terkait jenis tindakan yang diizinkan atau ditolak. Anda tidak dapat menentukan pengguna utama dalam kebijakan berbasis identitas karena kebijakan ini berlaku untuk pengguna atau peran yang dilampiri kebijakan. Untuk mempelajari semua elemen yang dapat Anda gunakan dalam kebijakan JSON, lihat [Referensi elemen kebijakan JSON IAM](#) dalam Panduan Pengguna IAM.

Contoh kebijakan berbasis identitas untuk DynamoDB

Untuk melihat contoh kebijakan berbasis identitas DynamoDB, silakan lihat [Contoh kebijakan berbasis identitas untuk Amazon DynamoDB](#).

Kebijakan berbasis sumber daya dalam DynamoDB

Mendukung kebijakan berbasis sumber daya Ya

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya yang dilampiri kebijakan tersebut, kebijakan ini menentukan jenis tindakan yang dapat dilakukan oleh pengguna utama tertentu di sumber daya tersebut dan apa ketentuannya. Anda harus [menentukan pengguna utama](#) dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan seluruh akun atau entitas IAM di akun lain sebagai pengguna utama dalam kebijakan berbasis sumber daya. Menambahkan pengguna utama lintas akun ke kebijakan berbasis sumber daya bagian dari membangun hubungan kepercayaan. Ketika prinsipal dan sumber daya berbeda Akun AWS, administrator IAM di akun tepercaya juga harus memberikan izin entitas utama (pengguna atau peran) untuk mengakses sumber daya. Izin diberikan dengan melampirkan kebijakan berbasis identitas ke entitas tersebut.

Namun, jika kebijakan berbasis sumber daya memberikan akses kepada pengguna utama dalam akun yang sama, kebijakan berbasis identitas lainnya tidak diperlukan. Untuk informasi selengkapnya, lihat [Perbedaan peran IAM dari kebijakan berbasis sumber daya](#) dalam Panduan Pengguna IAM.

Tindakan kebijakan untuk DynamoDB

Mendukung tindakan kebijakan

Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Elemen `Action` dari kebijakan JSON menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam sebuah kebijakan. Tindakan kebijakan biasanya memiliki nama yang sama dengan operasi AWS API terkait. Ada beberapa pengecualian, misalnya tindakan hanya izin yang tidak memiliki operasi API yang cocok. Ada juga beberapa operasi yang memerlukan beberapa tindakan dalam suatu kebijakan. Tindakan tambahan ini disebut tindakan dependen.

Menyertakan tindakan dalam kebijakan untuk memberikan izin untuk melakukan operasi terkait.

Untuk melihat daftar tindakan DynamoDB, lihat [Tindakan yang ditentukan oleh Amazon DynamoDB](#) di Referensi Otorisasi Layanan.

Tindakan kebijakan di DynamoDB menggunakan awalan berikut sebelum tindakan:

```
aws
```

Untuk menetapkan secara spesifik beberapa tindakan dalam satu pernyataan, pisahkan tindakan-tindakan tersebut dengan koma.

```
"Action": [  
  "aws:action1",  
  "aws:action2"  
]
```

Untuk melihat contoh kebijakan berbasis identitas DynamoDB, silakan lihat [Contoh kebijakan berbasis identitas untuk Amazon DynamoDB](#).

Sumber daya kebijakan untuk DynamoDB

Mendukung sumber daya kebijakan

Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Elemen kebijakan JSON `Resource` menentukan objek atau beberapa objek yang menjadi target penerapan tindakan. Pernyataan harus menyertakan elemen `Resource` atau `NotResource`. Praktik terbaiknya, tentukan sumber daya menggunakan [Amazon Resource Name \(ARN\)](#). Anda dapat melakukan ini untuk tindakan yang mendukung jenis sumber daya tertentu, yang dikenal sebagai izin tingkat sumber daya.

Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, misalnya operasi pencantuman, gunakan wildcard (*) untuk menunjukkan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*" 
```

Untuk melihat daftar jenis sumber daya DynamoDB dan ARN-nya, lihat [Sumber daya yang ditentukan oleh Amazon DynamoDB](#) dalam Referensi Otorisasi Layanan. Untuk mempelajari tindakan mana yang dapat Anda tentukan ARN setiap sumber daya, lihat [Tindakan yang ditentukan oleh Amazon DynamoDB](#).

Untuk melihat contoh kebijakan berbasis identitas DynamoDB, silakan lihat [Contoh kebijakan berbasis identitas untuk Amazon DynamoDB](#).

Kunci persyaratan kebijakan untuk DynamoDB

Mendukung kunci kondisi kebijakan spesifik layanan

Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Elemen `Condition` (atau blok `Condition`) memungkinkan Anda menentukan kondisi di mana suatu pernyataan akan diterapkan. Elemen `Condition` bersifat opsional. Anda dapat membuat ekspresi kondisional yang menggunakan [operator kondisi](#), misalnya sama dengan atau kurang dari, untuk mencocokkan kondisi dalam kebijakan dengan nilai-nilai yang diminta.

Jika Anda menentukan beberapa elemen `Condition` dalam satu pernyataan, atau beberapa kunci dalam satu elemen `Condition`, AWS akan mengevaluasinya dengan menggunakan operasi AND logis. Jika Anda menentukan beberapa nilai untuk satu kunci kondisi, AWS mengevaluasi kondisi menggunakan OR operasi logis. Semua kondisi harus dipenuhi sebelum izin pernyataan diberikan.

Anda juga dapat menggunakan variabel placeholder saat menentukan kondisi. Misalnya, Anda dapat memberikan izin kepada pengguna IAM untuk mengakses sumber daya hanya jika izin tersebut mempunyai tag yang sesuai dengan nama pengguna IAM mereka. Untuk informasi selengkapnya, lihat [Elemen kebijakan IAM: variabel dan tag](#) di Panduan Pengguna IAM.

AWS mendukung kunci kondisi global dan kunci kondisi khusus layanan. Untuk melihat semua kunci kondisi AWS global, lihat [kunci konteks kondisi AWS global](#) di Panduan Pengguna IAM.

Untuk melihat daftar kunci persyaratan DynamoDB, lihat [Kunci persyaratan untuk Amazon DynamoDB](#) dalam Referensi Otorisasi Layanan. Untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan dengan kunci kondisi, lihat [Tindakan yang ditentukan oleh Amazon DynamoDB](#).

Untuk melihat contoh kebijakan berbasis identitas DynamoDB, silakan lihat [Contoh kebijakan berbasis identitas untuk Amazon DynamoDB](#).

Daftar kontrol akses (ACL) di DynamoDB

Mendukung ACL

Tidak

Daftar kontrol akses (ACL) mengontrol pengguna utama (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL serupa dengan kebijakan berbasis sumber daya, meskipun tidak menggunakan format dokumen kebijakan JSON.

Kontrol akses berbasis atribut (ABAC) dengan DynamoDB

Mendukung ABAC (tanda dalam kebijakan) Parsial

Kontrol akses berbasis atribut (ABAC) adalah strategi otorisasi yang menentukan izin berdasarkan atribut. Dalam AWS, atribut ini disebut tag. Anda dapat melampirkan tag ke entitas IAM (pengguna atau peran) dan ke banyak AWS sumber daya. Pemberian tanda ke entitas dan sumber daya adalah langkah pertama dari ABAC. Kemudian, rancanglah kebijakan ABAC untuk mengizinkan operasi saat tag milik pengguna utama cocok dengan tag yang ada di sumber daya yang ingin diakses.

ABAC sangat berguna di lingkungan yang berkembang dengan cepat dan berguna di situasi di mana pengelolaan kebijakan menjadi rumit.

Untuk mengendalikan akses berdasarkan tag, berikan informasi tentang tag di [elemen kondisi](#) dari kebijakan dengan menggunakan kunci kondisi `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, atau `aws:TagKeys`.

Jika sebuah layanan mendukung ketiga kunci kondisi untuk setiap jenis sumber daya, nilainya adalah Ya untuk layanan tersebut. Jika suatu layanan mendukung ketiga kunci kondisi hanya untuk beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnya tentang ABAC, lihat [Apa itu ABAC?](#) di Panduan Pengguna IAM. Untuk melihat tutorial yang menguraikan langkah-langkah pengaturan ABAC, lihat [Menggunakan kontrol akses berbasis atribut \(ABAC\)](#) dalam Panduan Pengguna IAM.

Menggunakan Kredensial sementara dengan DynamoDB

Mendukung kredensial sementara Ya

Beberapa Layanan AWS tidak berfungsi saat Anda masuk menggunakan kredensial sementara. Untuk informasi tambahan, termasuk yang Layanan AWS bekerja dengan kredensial sementara, lihat [Layanan AWS yang bekerja dengan IAM di Panduan Pengguna IAM](#).

Anda menggunakan kredensial sementara jika Anda masuk AWS Management Console menggunakan metode apa pun kecuali nama pengguna dan kata sandi. Misalnya, ketika Anda mengakses AWS menggunakan tautan masuk tunggal (SSO) perusahaan Anda, proses tersebut secara otomatis membuat kredensial sementara. Anda juga akan membuat kredensial sementara secara otomatis saat

masuk ke konsol sebagai pengguna dan kemudian beralih peran. Untuk informasi selengkapnya tentang cara beralih peran, lihat [Beralih peran \(konsol\)](#) di Panduan Pengguna IAM.

Anda dapat membuat kredensial sementara secara manual menggunakan API AWS CLI atau AWS . Anda kemudian dapat menggunakan kredensial sementara tersebut untuk mengakses AWS AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensi sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat [Kredensial keamanan sementara di IAM](#).

Izin pengguna utama lintas layanan untuk DynamoDB


Mendukung sesi akses maju (FAS)	Ya
---------------------------------	----

Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Saat Anda menggunakan beberapa layanan, Anda mungkin melakukan tindakan yang kemudian memulai tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat [Sesi akses maju](#).

Peran layanan untuk DynamoDB

Mendukung peran layanan	Ya
-------------------------	----

Peran layanan adalah [peran IAM](#) yang diambil oleh layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat [Membuat peran untuk mendelegasikan izin ke Layanan AWS](#) dalam Panduan Pengguna IAM.

 Warning

Mengubah izin untuk sebuah peran layanan dapat merusak fungsionalitas DynamoDB. Edit peran layanan hanya jika DynamoDB memberikan panduan untuk melakukannya.

Peran terkait layanan untuk DynamoDB

Mendukung peran terkait layanan

Tidak

Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan dapat menggunakan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

Untuk detail tentang pembuatan atau pengelolaan peran terkait layanan, lihat [Layanan AWS yang berfungsi dengan IAM](#). Temukan sebuah layanan dalam tabel yang memiliki Yes di kolom Peran terkait layanan. Pilih tautan Ya untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Contoh kebijakan berbasis identitas untuk Amazon DynamoDB

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau melakukan modifikasi atas sumber daya DynamoDB. Mereka juga tidak dapat melakukan tugas dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau AWS API. Untuk memberikan izin kepada pengguna untuk melakukan tindakan pada sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat menjalankan peran.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM menggunakan contoh dokumen kebijakan JSON ini, lihat [Membuat kebijakan IAM](#) dalam Panduan Pengguna IAM.

Untuk mengetahui hal detail tentang tindakan dan jenis sumber daya yang ditentukan oleh DynamoDB, termasuk format ARN untuk setiap jenis sumber daya, silakan lihat [Tindakan, sumber daya, dan kunci persyaratan untuk Amazon DynamoDB](#) dalam Referensi Otorisasi Layanan.

Topik

- [Praktik terbaik kebijakan](#)
- [Menggunakan konsol DynamoDB](#)
- [Izinkan pengguna melihat izin mereka sendiri](#)
- [Menggunakan kebijakan berbasis identitas dengan Amazon DynamoDB](#)

Praktik terbaik kebijakan

Kebijakan berbasis identitas menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya DynamoDB yang ada di akun Anda. Tindakan ini mengenakan biaya kepada Akun AWS Anda. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Mulailah dengan kebijakan AWS terkelola dan beralih ke izin hak istimewa paling sedikit — Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan AWS terkelola yang memberikan izin untuk banyak kasus penggunaan umum. Mereka tersedia di Akun AWS. Kami menyarankan Anda mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola AWS pelanggan yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat [kebijakan yang dikelola AWS](#) atau [kebijakan yang dikelola AWS untuk fungsi pekerjaan](#) di Panduan Pengguna IAM.
- Menerapkan izin dengan hak akses paling rendah – Ketika Anda menetapkan izin dengan kebijakan IAM, hanya berikan izin yang diperlukan untuk melakukan tugas. Anda melakukan ini dengan menentukan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, juga dikenal sebagai izin hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk menerapkan izin, lihat [Kebijakan dan izin di IAM](#) di Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Sebagai contoh, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakan SSL. Anda juga dapat menggunakan ketentuan untuk memberikan akses ke tindakan layanan jika digunakan melalui yang spesifik Layanan AWS, seperti AWS CloudFormation. Untuk informasi selengkapnya, lihat [Elemen kebijakan JSON IAM: Syarat](#) di Panduan Pengguna IAM.
- Menggunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda guna memastikan izin yang aman dan berfungsi – IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan tersebut mematuhi bahasa kebijakan IAM (JSON) dan praktik terbaik IAM. IAM Access Analyzer menyediakan lebih dari 100 pemeriksaan kebijakan dan rekomendasi yang dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang aman dan fungsional. Untuk informasi selengkapnya, lihat [validasi kebijakan Analizer Akses IAM](#) di Panduan Pengguna IAM.
- Memerlukan otentikasi multi-faktor (MFA) - Jika Anda memiliki skenario yang mengharuskan pengguna IAM atau pengguna root di Anda, Akun AWS aktifkan MFA untuk keamanan tambahan. Untuk mewajibkan MFA saat operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda.

Untuk informasi selengkapnya, lihat [Mengonfigurasi akses API yang dilindungi MFA](#) di Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat [Praktik terbaik keamanan di IAM](#) dalam Panduan Pengguna IAM.

Menggunakan konsol DynamoDB

Untuk mengakses konsol Amazon DynamoDB, Anda harus memiliki serangkaian izin minimum. Izin ini harus memungkinkan Anda untuk daftar dan melihat rincian tentang sumber daya DynamoDB di Anda. Akun AWS Jika Anda membuat kebijakan berbasis identitas yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk entitas (pengguna atau peran) dengan kebijakan tersebut.

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau AWS API. Sebaliknya, izinkan akses hanya ke tindakan yang cocok dengan operasi API yang coba dilakukan.

Untuk memastikan bahwa pengguna dan peran masih dapat menggunakan konsol DynamoDB, lampirkan juga DynamoDB atau kebijakan terkelola ke ConsoleAccess entitas ReadOnly AWS . Untuk informasi selengkapnya, lihat [Menambahkan izin ke pengguna](#) dalam Panduan Pengguna IAM.

Izinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan para pengguna IAM melihat kebijakan inline dan terkelola yang dilampirkan ke identitas pengguna mereka. Kebijakan ini mencakup izin untuk menyelesaikan tindakan ini di konsol atau menggunakan API atau secara terprogram. AWS CLI AWS

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "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": "*"
  }
]
```

Menggunakan kebijakan berbasis identitas dengan Amazon DynamoDB

Topik ini mencakup penggunaan kebijakan berbasis identitas AWS Identity and Access Management (IAM) dengan Amazon DynamoDB dan memberikan contoh. Contoh ini menunjukkan bagaimana administrator akun dapat melampirkan kebijakan izin ke identitas IAM (pengguna, grup, dan peran) dan dengan demikian memberikan izin untuk melakukan operasi pada sumber daya Amazon DynamoDB.

Bagian dalam topik ini mencakup hal berikut:

- [Izin IAM diperlukan untuk menggunakan konsol Amazon DynamoDB](#)
- [AWS kebijakan IAM terkelola \(telah ditentukan\) untuk Amazon DynamoDB](#)
- [Contoh kebijakan yang dikelola pelanggan](#)

Berikut adalah contoh kebijakan izin.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
    "Sid": "DescribeQueryScanBooksTable",
    "Effect": "Allow",
    "Action": [
        "dynamodb:DescribeTable",
        "dynamodb:Query",
        "dynamodb:Scan"
    ],
    "Resource": "arn:aws:dynamodb:us-west-2:account-id:table/Books"
}
]
```

Kebijakan sebelumnya memiliki satu pernyataan yang memberikan izin untuk tiga tindakan DynamoDB (`dynamodb:DescribeTable`, `dynamodb:Scan`) pada tabel di `us-west-2` AWS Wilayah `dynamodb:Query`, yang dimiliki oleh akun yang ditentukan oleh AWS *account-id* Amazon Resource Name (ARN) dalam nilai `Resource` menentukan tabel tempat izin berlaku.

Izin IAM diperlukan untuk menggunakan konsol Amazon DynamoDB

Untuk bekerja dengan konsol DynamoDB, pengguna harus memiliki set izin minimum yang memungkinkan pengguna untuk bekerja dengan sumber daya DynamoDB AWS akun mereka. Selain izin DynamoDB ini, konsol memerlukan izin:

- CloudWatch Izin Amazon untuk menampilkan metrik dan grafik.
- AWS Data Pipeline izin untuk mengekspor dan mengimpor data DynamoDB.
- AWS Identity and Access Management izin untuk mengakses peran yang diperlukan untuk ekspor dan impor.
- Izin Layanan Pemberitahuan Sederhana Amazon untuk memberi tahu Anda setiap kali CloudWatch alarm dipicu.
- AWS Lambda izin untuk memproses catatan DynamoDB Streams.

Jika Anda membuat kebijakan IAM yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya bagi pengguna dengan kebijakan IAM tersebut. Untuk memastikan bahwa pengguna tersebut masih dapat menggunakan konsol DynamoDB, lampirkan juga kebijakan terkelola `AmazonDynamoDBReadOnlyAccess` AWS ke pengguna, seperti yang dijelaskan dalam [AWS kebijakan IAM terkelola \(telah ditentukan\) untuk Amazon DynamoDB](#)

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau Amazon DynamoDB API.

 Note

Jika Anda merujuk ke titik akhir VPC, Anda juga perlu mengotorisasi panggilan DescribeEndpoints API untuk prinsipal IAM yang meminta dengan tindakan IAM (dynamodb:). DescribeEndpoints Untuk mengetahui informasi selengkapnya, lihat [Kebijakan yang diperlukan untuk titik akhir](#).


AWS kebijakan IAM terkelola (telah ditentukan) untuk Amazon DynamoDB

AWS mengatasi beberapa kasus penggunaan umum dengan menyediakan kebijakan IAM mandiri yang dibuat dan dikelola oleh AWS. Kebijakan AWS terkelola ini memberikan izin yang diperlukan untuk kasus penggunaan umum sehingga Anda dapat menghindari keharusan menyelidiki izin mana yang diperlukan. Untuk informasi selengkapnya, lihat [Kebijakan Terkelola AWS](#) dalam Panduan Pengguna IAM.

Kebijakan AWS terkelola berikut, yang dapat Anda lampirkan ke pengguna di akun Anda, khusus untuk DynamoDB dan dikelompokkan berdasarkan skenario kasus penggunaan:

- AmazonDynamoDB ReadOnlyAccess - Memberikan akses hanya-baca ke sumber daya DynamoDB melalui file. AWS Management Console
- AmazonDynamoDB FullAccess - Memberikan akses penuh ke sumber daya DynamoDB melalui file. AWS Management Console

Anda dapat meninjau kebijakan izin AWS terkelola ini dengan masuk ke konsol IAM dan mencari kebijakan tertentu di sana.

 Important

Praktik terbaiknya adalah membuat kebijakan IAM khusus yang memberikan [hak akses paling rendah](#) kepada pengguna, peran, atau grup yang memerlukannya.

Contoh kebijakan yang dikelola pelanggan

Di bagian ini, Anda dapat menemukan contoh kebijakan yang memberikan izin untuk berbagai tindakan DynamoDB. Kebijakan ini berfungsi saat Anda menggunakan AWS SDK atau AWS CLI. Saat menggunakan konsol, Anda perlu memberikan izin tambahan yang khusus untuk konsol

tersebut. Untuk informasi selengkapnya, lihat [Izin IAM diperlukan untuk menggunakan konsol Amazon DynamoDB](#).

 Note

Semua contoh kebijakan berikut menggunakan salah satu AWS Wilayah dan berisi ID akun fiktif dan nama tabel.

Contoh:

- [Kebijakan IAM untuk memberikan izin untuk semua tindakan DynamoDB pada tabel](#)
- [Kebijakan IAM untuk memberikan izin hanya-baca pada item di tabel DynamoDB](#)
- [Kebijakan IAM untuk memberikan akses ke tabel DynamoDB tertentu dan indeksnya](#)
- [Kebijakan IAM untuk membaca, menulis, memperbarui, dan menghapus akses pada tabel DynamoDB](#)
- [Kebijakan IAM untuk memisahkan lingkungan DynamoDB di akun yang sama AWS](#)
- [Kebijakan IAM untuk mencegah pembelian kapasitas terpesan DynamoDB](#)
- [Kebijakan IAM untuk memberikan akses baca hanya untuk DynamoDB stream \(bukan untuk tabel\)](#)
- [Kebijakan IAM untuk mengizinkan AWS Lambda fungsi mengakses catatan aliran DynamoDB](#)
- [Kebijakan IAM untuk akses baca dan tulis ke kluster DynamoDB Accelerator \(DAX\)](#)

Panduan Pengguna IAM, mencakup [tiga contoh DynamoDB tambahan](#):

- [Amazon DynamoDB: Mengizinkan Akses ke Tabel Tertentu](#)
- [Amazon DynamoDB: Mengizinkan Akses ke Kolom Tertentu](#)
- [Amazon DynamoDB: Mengizinkan Akses Tingkat Baris ke DynamoDB Berdasarkan ID Amazon Cognito](#)

Kebijakan IAM untuk memberikan izin untuk semua tindakan DynamoDB pada tabel

Kebijakan berikut memberikan izin untuk semua tindakan DynamoDB pada tabel yang disebut Books. Sumber daya ARN yang ditentukan dalam Resource mengidentifikasi tabel di Wilayah tertentu. AWS Jika Anda mengganti nama tabel Books dalam ARN Resource dengan karakter

wildcard (*), semua tindakan DynamoDB diizinkan pada semua tabel di akun. Pertimbangkan dengan cermat kemungkinan implikasi keamanan sebelum menggunakan karakter wildcard pada kebijakan ini atau kebijakan IAM apa pun.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllAPIActionsOnBooks",
      "Effect": "Allow",
      "Action": "dynamodb:*",
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
    }
  ]
}
```

Note

Ini adalah contoh penggunaan karakter wildcard (*) untuk mengizinkan semua tindakan, termasuk administrasi, operasi data, pemantauan, dan pembelian kapasitas terpesan DynamoDB. Sebaliknya, praktik terbaiknya adalah menentukan secara eksplisit setiap tindakan yang akan diberikan dan hanya tindakan yang dibutuhkan pengguna, peran, atau grup tersebut.

Kebijakan IAM untuk memberikan izin hanya-baca pada item di tabel DynamoDB

Kebijakan izin berikut hanya memberikan izin untuk tindakan `GetItem`, `BatchGetItem`, `Scan`, `Query`, dan `ConditionCheckItem` DynamoDB, dan sebagai hasilnya, menetapkan akses hanya-baca pada tabel `Books`.

```
{
  "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"
}
]
}

```

Kebijakan IAM untuk memberikan akses ke tabel DynamoDB tertentu dan indeksnya

Kebijakan berikut ini memberikan izin untuk tindakan modifikasi data pada tabel DynamoDB yang disebut Books dan semua indeks tabel tersebut. Untuk informasi selengkapnya tentang cara kerja indeks, lihat [Meningkatkan akses data dengan indeks sekunder](#).

```

{
  "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/*"
      ]
    }
  ]
}

```

Kebijakan IAM untuk membaca, menulis, memperbarui, dan menghapus akses pada tabel DynamoDB

Gunakan kebijakan ini jika Anda perlu mengizinkan aplikasi Anda membuat, membaca, memperbarui, dan menghapus data di tabel, indeks, dan aliran Amazon DynamoDB. Gantikan nama AWS Region, ID akun Anda, dan nama tabel atau karakter wildcard (*) jika sesuai.

```
{
  "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/*"
  ]
}
```

Untuk memperluas kebijakan ini untuk mencakup semua tabel DynamoDB di AWS semua Wilayah untuk akun ini, gunakan wildcard (*) untuk Region dan nama tabel. Sebagai contoh:

```
"Resource": [
  "arn:aws:dynamodb:*:123456789012:table/*",
  "arn:aws:dynamodb:*:123456789012:table/*/index/*"
]
```

Kebijakan IAM untuk memisahkan lingkungan DynamoDB di akun yang sama AWS

Misalkan Anda memiliki lingkungan terpisah di mana setiap lingkungan memelihara versi tabelnya sendiri yang bernama ProductCatalog. Jika Anda membuat dua ProductCatalog tabel di AWS akun yang sama, bekerja di satu lingkungan dapat memengaruhi lingkungan lain karena cara pengaturan izin. Misalnya, kuota pada jumlah operasi bidang kontrol bersamaan (seperti CreateTable) ditetapkan pada tingkat AWS akun.

Sebagai hasilnya, setiap tindakan di satu lingkungan mengurangi jumlah operasi yang tersedia di lingkungan lain. Ada juga risiko bahwa kode di satu lingkungan mungkin secara tidak sengaja mengakses tabel di lingkungan lain.

Note

Jika Anda ingin memisahkan beban kerja produksi dan pengujian untuk membantu mengontrol potensi "radius ledakan" suatu peristiwa, praktik terbaiknya adalah membuat akun AWS terpisah untuk beban kerja pengujian dan produksi. Untuk informasi selengkapnya, lihat [Manajemen dan Pemisahan Akun AWS](#).

Misalkan Anda memiliki dua pengembang, Amit dan Alice, yang sedang menguji tabel `ProductCatalog`. Alih-alih setiap pengembang memerlukan AWS akun terpisah, pengembang Anda dapat berbagi AWS akun pengujian yang sama. Di akun pengujian ini, Anda dapat membuat salinan tabel yang sama untuk dikerjakan oleh setiap pengembang, seperti `Alice_ProductCatalog` dan `Amit_ProductCatalog`. Dalam hal ini, Anda dapat membuat pengguna Alice dan Amit di AWS akun yang Anda buat untuk lingkungan pengujian. Anda kemudian dapat memberikan izin kepada pengguna ini untuk melakukan tindakan DynamoDB pada tabel yang mereka miliki.

Untuk memberikan izin pengguna IAM ini, Anda dapat melakukan salah satu hal berikut:

- Buat kebijakan terpisah untuk setiap pengguna, lalu lampirkan setiap kebijakan ke penggunanya secara terpisah. Misalnya, Anda dapat melampirkan kebijakan berikut kepada pengguna Alice untuk mengizinkannya mengakses tindakan DynamoDB pada tabel `Alice_ProductCatalog`:

```
{
  "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/*"
    }
]
}

```

Kemudian, Anda dapat membuat kebijakan serupa dengan sumber daya berbeda (tabel `Amit_ProductCatalog`) untuk pengguna Amit.

- Daripada melampirkan kebijakan ke pengguna individual, Anda dapat menggunakan variabel kebijakan IAM untuk menulis kebijakan tunggal dan melampirkannya ke grup. Anda perlu membuat grup dan, untuk contoh ini, menambahkan pengguna Alice dan Amit ke grup. Contoh berikut memberikan izin untuk melakukan semua tindakan DynamoDB pada tabel `${aws:username}_ProductCatalog`. Variabel kebijakan `${aws:username}` diganti dengan nama pengguna pemohon ketika kebijakan dievaluasi. Misalnya, jika Alice mengirimkan permintaan untuk menambahkan item, tindakan tersebut hanya diperbolehkan jika Alice menambahkan item ke tabel `Alice_ProductCatalog`.

```

{
  "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

Saat menggunakan variabel kebijakan IAM, Anda harus secara eksplisit menentukan bahasa kebijakan IAM versi 2012-10-17 dalam kebijakan. Versi default bahasa kebijakan IAM (2008-10-17) tidak mendukung variabel kebijakan.

Alih-alih mengidentifikasi tabel tertentu sebagai sumber daya seperti yang biasa Anda lakukan, Anda bisa menggunakan karakter wildcard (*) untuk memberikan izin pada semua tabel yang nama tabelnya diawali dengan pengguna yang membuat permintaan, seperti yang ditunjukkan dalam contoh berikut.

```
"Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/${aws:username}_*"
```

Kebijakan IAM untuk mencegah pembelian kapasitas terpesan DynamoDB

Dengan kapasitas terpesan Amazon DynamoDB, Anda membayar biaya satu kali di muka dan berkomitmen untuk membayar tingkat penggunaan minimum dengan penghematan yang signifikan selama jangka waktu tertentu. Anda dapat menggunakan AWS Management Console untuk melihat dan membeli kapasitas cadangan. Namun, Anda mungkin tidak ingin semua pengguna di organisasi Anda dapat membeli kapasitas terpesan. Untuk informasi selengkapnya tentang kapasitas terpesan, lihat [harga Amazon DynamoDB](#).

DynamoDB menyediakan operasi API berikut untuk mengendalikan akses ke manajemen kapasitas terpesan:

- `dynamodb:DescribeReservedCapacity` – Mengembalikan pembelian kapasitas terpesan yang saat ini berlaku.
- `dynamodb:DescribeReservedCapacityOfferings` – Mengembalikan detail tentang paket kapasitas terpesan yang saat ini ditawarkan oleh AWS.
- `dynamodb:PurchaseReservedCapacityOfferings` – Melakukan pembelian aktual kapasitas terpesan.

AWS Management Console Menggunakan tindakan API ini untuk menampilkan informasi kapasitas cadangan dan melakukan pembelian. Anda tidak dapat memanggil operasi ini dari program aplikasi karena hanya dapat diakses dari konsol. Namun, Anda dapat mengizinkan atau menolak akses ke operasi ini dalam kebijakan izin IAM.

Kebijakan berikut memungkinkan pengguna untuk melihat pembelian dan penawaran kapasitas cadangan dengan menggunakan AWS Management Console — tetapi pembelian baru ditolak.

```
{
  "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:*"
}
```

Perhatikan bahwa kebijakan ini menggunakan karakter wildcard (*) untuk mengizinkan penjelasan izin untuk semua, dan untuk menolak pembelian kapasitas tersimpan DynamoDB untuk semua.

Kebijakan IAM untuk memberikan akses baca hanya untuk DynamoDB stream (bukan untuk tabel)

Saat Anda mengaktifkan DynamoDB Streams pada tabel, informasi diambil tentang setiap modifikasi pada item dalam tabel. Untuk informasi selengkapnya, lihat [Ubah tangkapan data DynamoDB Streams](#).

Dalam beberapa kasus, Anda mungkin ingin mencegah aplikasi membaca data dari tabel DynamoDB, tetapi tetap mengizinkan akses ke aliran tabel tersebut. Misalnya, Anda dapat mengonfigurasi AWS Lambda untuk melakukan polling aliran dan menjalankan fungsi Lambda saat pembaruan item terdeteksi, lalu melakukan pemrosesan tambahan.

Tindakan berikut tersedia untuk mengontrol akses ke DynamoDB stream:

- dynamodb:DescribeStream
- dynamodb:GetRecords
- dynamodb:GetShardIterator
- dynamodb:ListStreams

Contoh kebijakan berikut memberikan izin pengguna untuk mengakses aliran tabel bernama GameScores. Karakter wildcard (*) di ARN cocok dengan aliran yang terkait dengan tabel tersebut.

```
{
  "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/*"
  }
]
```

Perhatikan bahwa kebijakan ini memberikan akses ke aliran tabel GameScores, tetapi tidak ke tabel itu sendiri.

Kebijakan IAM untuk mengizinkan AWS Lambda fungsi mengakses catatan aliran DynamoDB

Jika Anda ingin tindakan tertentu dilakukan berdasarkan peristiwa dalam aliran DynamoDB, Anda dapat menulis fungsi AWS Lambda yang dipicu oleh peristiwa ini. Suatu fungsi Lambda seperti ini memerlukan izin untuk membaca data dari aliran DynamoDB. Untuk informasi selengkapnya tentang menggunakan Lambda dengan DynamoDB Streams, lihat [DynamoDB Streams dan pemicu AWS Lambda](#).

Untuk memberikan izin kepada Lambda, gunakan kebijakan izin yang terkait dengan peran IAM fungsi Lambda (juga dikenal sebagai peran eksekusi). Tentukan kebijakan ini saat Anda membuat fungsi Lambda.

Misalnya, Anda dapat mengaitkan kebijakan izin berikut dengan peran eksekusi untuk memberikan izin Lambda untuk melakukan tindakan DynamoDB Streams terdaftar.

```
{
  "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/*"
  }
]
}

```

Untuk informasi selengkapnya, lihat [izin AWS Lambda](#) dalam Panduan Pengembang AWS Lambda .

Kebijakan IAM untuk akses baca dan tulis ke kluster DynamoDB Accelerator (DAX)

Kebijakan berikut mengizinkan akses baca, tulis, pembaruan, dan hapus ke kluster DynamoDB Accelerator (DAX), tetapi tidak untuk tabel DynamoDB terkait. Untuk menggunakan kebijakan ini, ganti nama AWS Wilayah, ID akun, dan nama kluster DAX Anda.

Note

Kebijakan ini memberikan akses ke kluster DAX, tetapi tidak ke tabel DynamoDB yang terkait. Pastikan bahwa kluster DAX Anda memiliki kebijakan yang benar untuk melakukan operasi yang sama ini pada tabel DynamoDB atas nama Anda.

```

{
  "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"
    }
  ]
}

```

```
}
```

Untuk memperluas kebijakan ini guna mencakup akses DAX untuk semua AWS Wilayah untuk akun, gunakan karakter wildcard (*) untuk nama Wilayah.

```
"Resource": "arn:aws:dax:*:123456789012:cache/MyDAXCluster"
```

Pemecahan masalah identitas dan akses Amazon DynamoDB

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda hadapi ketika bekerja dengan DynamoDB dan IAM.

Topik

- [Saya tidak berwenang melakukan tindakan di DynamoDB](#)
- [Saya tidak berwenang untuk melakukan iam: PassRole](#)
- [Saya ingin mengizinkan orang di luar saya mengakses sumber daya Akun AWS DynamoDB saya](#)

Saya tidak berwenang melakukan tindakan di DynamoDB

Jika AWS Management Console memberitahu Anda bahwa Anda tidak berwenang untuk melakukan suatu tindakan, maka Anda harus menghubungi administrator Anda untuk bantuan. Administrator adalah orang yang memberikan nama pengguna dan kata sandi kepada Anda.

Contoh kesalahan berikut terjadi ketika pengguna mateojackson mencoba menggunakan konsol untuk melihat detail tentang suatu sumber daya *my-example-widget* fiktif, tetapi tidak memiliki izin `aws:GetWidget` fiktif.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:  
aws:GetWidget on resource: my-example-widget
```

Dalam hal ini, Mateo meminta administratornya untuk memperbarui kebijakan miliknya agar dia dapat mengakses sumber daya *my-example-widget* dengan menggunakan tindakan `aws:GetWidget`.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan yang menyatakan bahwa Anda tidak berwenang untuk melakukan tindakan `iam:PassRole`, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke DynamoDB.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh galat berikut terjadi ketika pengguna IAM bernama `marymajor` mencoba menggunakan konsol tersebut untuk melakukan tindakan di DynamoDB. Namun, tindakan tersebut memerlukan layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan `iam:PassRole` tersebut.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya ingin mengizinkan orang di luar saya mengakses sumber daya Akun AWS DynamoDB saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau pengguna di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACL), Anda dapat menggunakan kebijakan tersebut untuk memberi pengguna akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa hal berikut:

- Untuk mempelajari apakah DynamoDB mendukung fitur-fitur ini, lihat [Cara kerja Amazon DynamoDB dengan IAM](#).
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat [Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki](#) di Panduan Pengguna IAM.

- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat [Menyediakan akses yang Akun AWS dimiliki oleh pihak ketiga](#) dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat [Memberikan akses kepada pengguna eksternal yang sah \(federasi identitas\)](#) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara penggunaan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat [Perbedaan antara peran IAM dan kebijakan berbasis sumber daya](#) di Panduan Pengguna IAM.

Kebijakan IAM untuk mencegah pembelian kapasitas terpesan DynamoDB

Dengan kapasitas terpesan Amazon DynamoDB, Anda membayar biaya satu kali di muka dan berkomitmen untuk membayar tingkat penggunaan minimum dengan penghematan yang signifikan selama jangka waktu tertentu. Anda dapat menggunakan AWS Management Console untuk melihat dan membeli kapasitas cadangan. Namun, Anda mungkin tidak ingin semua pengguna di organisasi Anda dapat membeli kapasitas terpesan. Untuk informasi selengkapnya tentang kapasitas terpesan, lihat [harga Amazon DynamoDB](#).

DynamoDB menyediakan operasi API berikut untuk mengendalikan akses ke manajemen kapasitas terpesan:

- `dynamodb:DescribeReservedCapacity` – Mengembalikan pembelian kapasitas terpesan yang saat ini berlaku.
- `dynamodb:DescribeReservedCapacityOfferings` – Mengembalikan detail tentang paket kapasitas terpesan yang saat ini ditawarkan oleh AWS.
- `dynamodb:PurchaseReservedCapacityOfferings` – Melakukan pembelian aktual kapasitas terpesan.

AWS Management Console Menggunakan tindakan API ini untuk menampilkan informasi kapasitas cadangan dan melakukan pembelian. Anda tidak dapat memanggil operasi ini dari program aplikasi karena hanya dapat diakses dari konsol. Namun, Anda dapat mengizinkan atau menolak akses ke operasi ini dalam kebijakan izin IAM.

Kebijakan berikut memungkinkan pengguna untuk melihat pembelian dan penawaran kapasitas cadangan dengan menggunakan AWS Management Console — tetapi pembelian baru ditolak.

```
{
```

```
"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:*"
  }
]
```

Perhatikan bahwa kebijakan ini menggunakan karakter wildcard (*) untuk mengizinkan penjelasan izin untuk semua, dan untuk menolak pembelian kapasitas terpesan DynamoDB untuk semua.

Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci

Ketika memberikan izin di DynamoDB, Anda dapat menetapkan syarat yang menentukan bagaimana kebijakan izin berlaku.

Gambaran Umum

Di DynamoDB, Anda memiliki pilihan untuk menentukan syarat saat memberikan izin menggunakan kebijakan IAM (lihat [Manajemen Identitas dan Akses untuk Amazon DynamoDB](#)). Sebagai contoh, Anda dapat:

- Memberi izin kepada pengguna akan akses hanya-baca pada item dan atribut tertentu dalam tabel atau indeks sekunder.
- Memberi izin kepada pengguna akan akses hanya-tulis pada atribut tertentu dalam tabel, berdasarkan identitas pengguna tersebut.

Di DynamoDB, Anda dapat menentukan ketentuan dalam kebijakan IAM menggunakan kunci ketentuan, seperti yang diilustrasikan dalam kasus penggunaan di bagian berikut.

Note

Beberapa AWS layanan juga mendukung kondisi berbasis tag; Namun, DynamoDB tidak.

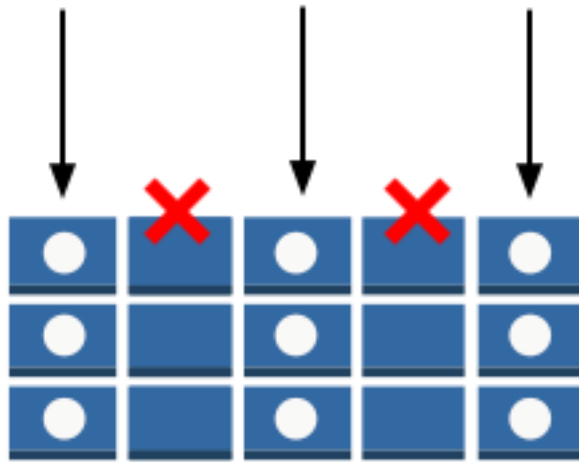
Kasus penggunaan izin

Selain mengontrol akses ke tindakan DynamoDB API, Anda juga dapat mengontrol akses ke item data dan atribut individual. Misalnya, Anda dapat melakukan hal berikut:

- Memberikan izin pada tabel, namun batasi akses ke item tertentu dalam tabel tersebut berdasarkan nilai kunci primer tertentu. Misalnya aplikasi jejaring sosial untuk permainan, yang mana semua data permainan yang disimpan pengguna disimpan dalam satu tabel, tetapi tidak ada pengguna yang dapat mengakses item data yang bukan miliknya, seperti yang diperlihatkan dalam ilustrasi berikut:



- Menyembunyikan informasi sehingga hanya subset dari atribut yang terlihat oleh pengguna. Misalnya aplikasi yang menampilkan data penerbangan untuk bandara terdekat, berdasarkan lokasi pengguna. Nama maskapai penerbangan, waktu kedatangan dan keberangkatan, serta nomor penerbangan semuanya ditampilkan. Namun atribut seperti nama pilot atau jumlah penumpang disembunyikan, seperti terlihat pada ilustrasi berikut:



Untuk menerapkan kontrol akses terperinci semacam ini, Anda menulis kebijakan izin IAM yang menentukan kondisi untuk mengakses kredensial keamanan dan izin terkait. Anda kemudian menerapkan kebijakan tersebut kepada pengguna, grup, atau peran yang Anda buat menggunakan konsol IAM. Kebijakan IAM Anda dapat membatasi akses ke masing-masing item dalam tabel, akses ke atribut dalam item tersebut, atau keduanya secara bersamaan.

Anda juga dapat menggunakan federasi identitas web untuk mengontrol akses oleh pengguna yang diautentikasi dengan Login dengan Amazon, Facebook, atau Google. Untuk informasi selengkapnya, lihat [Menggunakan federasi identitas web](#).

Anda menggunakan elemen `Condition` IAM untuk menerapkan kebijakan kontrol akses yang terperinci. Dengan menambahkan elemen `Condition` ke kebijakan izin, Anda dapat mengizinkan atau menolak akses ke item dan atribut dalam tabel dan indeks DynamoDB, berdasarkan kebutuhan bisnis khusus Anda.

Sebagai contoh, pertimbangkan aplikasi game seluler yang memungkinkan pemain memilih dan memainkan berbagai permainan berbeda. Aplikasi ini menggunakan tabel DynamoDB bernama `GameScores` untuk melacak skor tertinggi dan data pengguna lainnya. Setiap item dalam tabel diidentifikasi secara unik berdasarkan ID pengguna dan nama permainan yang dimainkan pengguna. Tabel `GameScores` memiliki kunci primer yang terdiri dari kunci partisi (`UserId`) dan kunci urutan (`GameTitle`). Pengguna hanya memiliki akses ke data game yang terkait dengan ID pengguna mereka. Pengguna yang ingin memainkan game harus memiliki peran IAM bernama `GameRole`, yang memiliki kebijakan keamanan yang melekat padanya.

Untuk mengelola izin pengguna di aplikasi ini, Anda dapat menulis kebijakan izin seperti berikut:

```
{
  "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"
        }
      }
    }
  ]
}
```

Selain memberikan izin untuk tindakan DynamoDB tertentu (elemen Action) pada tabel GameScores (elemen Resource), elemen Condition menggunakan kunci kondisi khusus untuk DynamoDB berikut yang membatasi izin sebagai berikut:

- `dynamodb:LeadingKeys` – Kunci ketentuan ini memungkinkan pengguna untuk mengakses hanya item yang nilai kunci partisinya cocok dengan ID pengguna mereka. ID ini, `#{www.amazon.com:user_id}`, adalah variabel substitusi. Untuk informasi selengkapnya tentang variabel substitusi, lihat [Menggunakan federasi identitas web](#).
- `dynamodb:Attributes` – Kunci ketentuan ini membatasi akses ke atribut tertentu sehingga hanya tindakan yang tercantum dalam kebijakan izin yang dapat mengembalikan nilai untuk atribut ini. Selain itu, klausul `StringEqualsIfExists` memastikan bahwa aplikasi harus selalu menyediakan daftar atribut tertentu untuk ditindaklanjuti dan aplikasi tidak dapat meminta semua atribut.

Ketika kebijakan IAM dievaluasi, hasilnya selalu `true` (akses diizinkan) atau `false` (akses ditolak). Jika ada bagian dari elemen `Condition` yang `false`, seluruh kebijakan mengevaluasi ke `false` dan akses ditolak.

Important

Jika Anda menggunakan `dynamodb:Attributes`, Anda harus menentukan nama semua atribut kunci primer dan kunci indeks untuk tabel dan indeks sekunder apa pun yang tercantum dalam kebijakan. Jika tidak, DynamoDB tidak dapat menggunakan atribut kunci ini untuk melakukan tindakan yang diminta.

Dokumen kebijakan IAM hanya dapat berisi karakter Unicode berikut: tab horizontal (U+0009), umpan baris (U+000A), pengembalian pengangkutan (U+000D), dan karakter dalam rentang U+0020 hingga U+00FF.

Menentukan ketentuan: Menggunakan kunci syarat

AWS menyediakan satu set kunci kondisi yang telah ditentukan (AWS-wide condition keys) untuk semua AWS layanan yang mendukung IAM untuk kontrol akses. Misalnya, Anda dapat menggunakan kunci kondisi `aws:SourceIp` untuk memeriksa alamat IP pemohon sebelum mengizinkan tindakan yang akan dilakukan. Untuk informasi selengkapnya dan daftar tombol AWS-wide, lihat Kunci yang [Tersedia untuk Ketentuan](#) di Panduan Pengguna IAM.

Tabel berikut menunjukkan kunci kondisi layanan khusus DynamoDB yang berlaku untuk DynamoDB.

Kunci Kondisi DynamoDB	Deskripsi
<code>dynamodb:LeadingKeys</code>	<p>Mewakili atribut kunci pertama dari tabel—dengan kata lain, kunci partisi. Nama kunci <code>LeadingKeys</code> bersifat jamak, meskipun kuncinya digunakan dengan tindakan satu item. Selain itu, Anda harus menggunakan pengubah <code>ForAllValues</code> ketika menggunakan <code>LeadingKeys</code> dalam suatu kondisi.</p>
<code>dynamodb:Select</code>	<p>Mewakili parameter <code>Select</code> dari permintaan <code>Query</code> atau <code>Scan</code>. <code>Select</code> dapat berupa nilai berikut:</p> <ul style="list-style-type: none"> • <code>ALL_ATTRIBUTES</code> • <code>ALL_PROJECTED_ATTRIBUTES</code> • <code>SPECIFIC_ATTRIBUTES</code> • <code>COUNT</code>
<code>dynamodb:Attributes</code>	<p>Mewakili daftar nama atribut dalam suatu permintaan, atau atribut yang dikembalikan dari permintaan. Nilai <code>Attributes</code> diberi nama dengan cara yang sama dan memiliki arti yang sama sebagai parameter untuk tindakan DynamoDB API tertentu, seperti yang ditunjukkan berikut:</p> <ul style="list-style-type: none"> • <code>AttributesToGet</code> <p>Digunakan oleh: <code>BatchGetItem</code>, <code>GetItem</code>, <code>Query</code>, <code>Scan</code></p> • <code>AttributeUpdates</code> <p>Digunakan oleh: <code>UpdateItem</code></p> • <code>Expected</code> <p>Digunakan oleh: <code>DeleteItem</code>, <code>PutItem</code>, <code>UpdateItem</code></p> • <code>Item</code> <p>Digunakan oleh: <code>PutItem</code></p> • <code>ScanFilter</code> <p>Digunakan oleh: <code>Scan</code></p>

Kunci Kondisi DynamoDB	Deskripsi
dynamodb: ReturnValues	Mewakili parameter <code>ReturnValues</code> dari suatu permintaan. <code>ReturnValues</code> dapat berupa nilai berikut: <ul style="list-style-type: none">• ALL_OLD• UPDATED_OLD• ALL_NEW• UPDATED_NEW• NONE
dynamodb: ReturnConsumedCapacity	Mewakili parameter <code>ReturnConsumedCapacity</code> dari suatu permintaan. <code>ReturnConsumedCapacity</code> dapat berupa salah satu nilai berikut: <ul style="list-style-type: none">• TOTAL• NONE

Membatasi akses pengguna

Banyak kebijakan izin IAM memungkinkan pengguna mengakses hanya item dalam tabel yang nilai kunci partisinya cocok dengan pengidentifikasi pengguna. Misalnya, aplikasi game sebelumnya membatasi akses dengan cara ini sehingga pengguna hanya dapat mengakses data game yang dikaitkan dengan ID pengguna mereka. Variabel substitusi IAM `{www.amazon.com:user_id}`, `{graph.facebook.com:id}`, dan `{accounts.google.com:sub}` berisi pengenalan pengguna untuk Login with Amazon, Facebook, dan Google. Untuk mempelajari cara aplikasi masuk ke salah satu penyedia identitas ini dan memperoleh pengidentifikasi ini, lihat [Menggunakan federasi identitas web](#).

Note

Masing-masing contoh pada bagian berikut menetapkan klausul `Effect` untuk `Allow` dan menentukan hanya tindakan, sumber daya, dan parameter yang diizinkan. Akses hanya diizinkan untuk apa yang tercantum secara eksplisit dalam kebijakan IAM.

Dalam beberapa kasus, dimungkinkan untuk menulis ulang kebijakan ini sehingga kebijakan berbasis penolakan (yaitu, pengaturan klausul `Effect` untuk `Deny` dan membalikkan semua

logika dalam kebijakan). Namun, sebaiknya Anda menghindari penggunaan kebijakan berbasis penolakan dengan DynamoDB karena kebijakan sulit untuk ditulis dengan benar, dibandingkan dengan kebijakan berbasis izin. Selain itu, perubahan di masa mendatang pada DynamoDB API (atau perubahan input API yang ada) dapat membuat kebijakan berbasis penolakan tidak efektif.

Kebijakan Contoh: Menggunakan ketentuan untuk kontrol parameter terperinci

Bagian ini menunjukkan beberapa kebijakan untuk mengimplementasikan kontrol akses terperinci pada tabel dan indeks DynamoDB.

Note

Semua contoh menggunakan wilayah us-west-2 dan berisi ID akun fiktif.

1: Memberikan izin yang membatasi akses ke item dengan nilai kunci partisi tertentu

Kebijakan izin berikut memberikan izin yang memungkinkan seperangkat tindakan DynamoDB pada tabel `GameScore`. Kebijakan tersebut menggunakan kunci ketentuan `dynamodb:LeadingKeys` untuk membatasi tindakan pengguna hanya pada item yang nilai kunci partisi `UserID`-nya cocok dengan ID pengguna unik `Login with Amazon` untuk aplikasi ini.

Important

Daftar tindakan tidak termasuk izin untuk `Scan` karena `Scan` mengembalikan semua item apa pun kunci awalnya.

```
{
  "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

Saat menggunakan variabel kebijakan, Anda harus secara jelas menyebutkan versi 2012-10-17 dalam kebijakan tersebut. Versi default bahasa kebijakan akses, 2008-10-17, tidak mendukung variabel kebijakan.

Untuk menerapkan akses hanya baca, Anda dapat menghapus tindakan apa pun yang dapat mengubah data. Dalam kebijakan berikut, hanya tindakan yang menyediakan akses baca saja yang disertakan dalam ketentuan.

```

{
  "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

Jika Anda menggunakan `dynamodb:Attributes`, Anda harus menentukan nama semua atribut kunci primer dan kunci indeks, untuk tabel dan indeks sekunder apa pun yang tercantum dalam kebijakan. Jika tidak, DynamoDB tidak dapat menggunakan atribut kunci ini untuk melakukan tindakan yang diminta.

2: Memberikan izin yang membatasi akses ke atribut tertentu dalam tabel

Kebijakan izin berikut mengizinkan akses ke hanya dua atribut tertentu dalam tabel dengan menambahkan kunci ketentuan `dynamodb:Attributes`. Atribut ini dapat dibaca, ditulis, atau dievaluasi dalam penulisan bersyarat atau filter pemindaian.

```

{
  "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

Kebijakan ini mengambil pendekatan daftar izin, yang mengizinkan akses ke set atribut bernama. Anda dapat menulis kebijakan serupa yang menolak akses ke atribut lain. Kami tidak merekomendasikan pendekatan daftar penolakan ini. Pengguna dapat menentukan nama atribut yang ditolak ini dengan mengikuti prinsip hak akses paling rendah, seperti yang dijelaskan di Wikipedia di http://en.wikipedia.org/wiki/Principle_of_least_privilege, dan gunakan pendekatan daftar izin untuk menghitung semua nilai yang diizinkan, daripada menentukan atribut yang ditolak.

Kebijakan ini tidak mengizinkan PutItem, DeleteItem, atau BatchWriteItem. Tindakan ini selalu menggantikan seluruh item sebelumnya, yang memungkinkan pengguna menghapus nilai sebelumnya untuk atribut yang tidak boleh mereka akses.

Klausula `StringEqualsIfExists` dalam kebijakan izin memastikan hal berikut:

- Jika pengguna menentukan parameter `Select`, nilainya harus `SPECIFIC_ATTRIBUTES`. Persyaratan ini mencegah tindakan API mengembalikan atribut apa pun yang tidak diizinkan, seperti dari proyeksi indeks.

- Jika pengguna menentukan parameter `ReturnValues`, nilainya harus `NONE`, `UPDATED_OLD`, atau `UPDATED_NEW`. Hal ini diperlukan karena tindakan `UpdateItem` tersebut juga melakukan operasi baca implisit untuk memeriksa apakah suatu item ada sebelum menggantinya, dan agar nilai atribut sebelumnya dapat dikembalikan jika diminta. Membatasi `ReturnValues` dengan cara ini memastikan bahwa pengguna hanya dapat membaca atau menulis atribut yang diizinkan.
- Klausula `StringEqualsIfExists` memastikan bahwa hanya satu dari parameter ini — `Select` atau `ReturnValues` — yang dapat digunakan per permintaan, dalam konteks tindakan yang diizinkan.

Berikut ini adalah beberapa variasi pada kebijakan ini:

- Untuk hanya mengizinkan tindakan baca, Anda dapat menghapus `UpdateItem` dari daftar tindakan yang diizinkan. Karena tidak ada tindakan yang tersisa menerima `ReturnValues`, Anda dapat menghapus `ReturnValues` dari kondisi. Anda juga dapat mengubah `StringEqualsIfExists` menjadi `StringEquals` karena parameter `Select` selalu memiliki nilai (`ALL_ATTRIBUTES`, kecuali ditentukan lain).
- Untuk hanya mengizinkan tindakan tulis, Anda dapat menghapus semuanya kecuali `UpdateItem` dari daftar tindakan yang diizinkan. Karena `UpdateItem` tidak menggunakan parameter `Select`, Anda dapat menghapus `Select` dari kondisi. Anda juga harus mengubah `StringEqualsIfExists` menjadi `StringEquals` karena parameter `ReturnValues` selalu memiliki nilai (`NONE`, kecuali ditentukan lain).
- Untuk mengizinkan semua atribut yang namanya cocok dengan pola, gunakan `StringLike`, bukan `StringEquals`, dan gunakan pola multi-karakter yang cocok dengan karakter wildcard (*).

3: Memberikan izin untuk mencegah pembaruan pada atribut tertentu

Kebijakan izin berikut membatasi akses pengguna untuk memperbarui hanya atribut tertentu yang diidentifikasi oleh kunci kondisi `dynamodb:Attributes`. Kondisi `StringNotLike` mencegah aplikasi memperbarui atribut yang ditentukan menggunakan kunci kondisi `dynamodb:Attributes`.

```
{
  "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"
      ]
    }
  }
}
```

Perhatikan hal-hal berikut:

- Tindakan `UpdateItem`, seperti tindakan tulis lainnya, memerlukan akses baca ke item sehingga dapat mengembalikan nilai sebelum dan sesudah pembaruan. Dalam kebijakan tersebut, Anda membatasi tindakan untuk mengakses hanya atribut yang diizinkan yang akan diperbarui dengan menentukan kunci ketentuan `dynamodb:ReturnValues`. Kunci kondisi membatasi `ReturnValues` dalam permintaan untuk menentukan hanya `NONE`, `UPDATED_OLD`, atau `UPDATED_NEW` dan tidak mencakup `ALL_OLD` atau `ALL_NEW`.
- Tindakan `PutItem` dan `DeleteItem` mengganti seluruh item, dengan demikian memungkinkan aplikasi memodifikasi atribut apa pun. Jadi, ketika membatasi aplikasi untuk memperbarui hanya atribut tertentu, Anda tidak harus memberikan izin untuk API ini.

4: Berikan izin untuk mengkueri hanya atribut yang diproyeksikan dalam indeks

Kebijakan izin berikut memungkinkan kueri pada indeks sekunder (`TopScoreDateTimeIndex`) dengan menggunakan kunci ketentuan `dynamodb:Attributes`. Kebijakan ini juga membatasi kueri untuk meminta atribut tertentu saja yang telah diproyeksikan ke dalam indeks.

Untuk mengharuskan aplikasi menentukan daftar atribut dalam kueri, kebijakan juga menentukan kunci ketentuan dynamodb:Select untuk mengharuskan parameter Select dari tindakan Query DynamoDB adalah SPECIFIC_ATTRIBUTES. Daftar atribut terbatas pada daftar tertentu yang disediakan menggunakan kunci ketentuan dynamodb:Attributes.

```
{
  "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"
        }
      }
    }
  ]
}
```

Kebijakan izin berikut ini serupa, tetapi kueri harus meminta semua atribut yang telah diproyeksikan ke dalam indeks.

```
{
  "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: Memberikan izin untuk membatasi akses ke atribut tertentu dan nilai-nilai kunci partisi

Kebijakan izin berikut memungkinkan tindakan DynamoDB tertentu (ditentukan dalam elemen `Action`) pada tabel dan indeks tabel (ditentukan dalam elemen `Resource`). Kebijakan menggunakan kunci ketentuan `dynamodb:LeadingKeys` untuk membatasi izin hanya untuk item yang nilai kunci partisinya cocok dengan ID Facebook pengguna.

```

{
  "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"
        ]
      }
    }
  }
]
}

```

Perhatikan hal-hal berikut:

- Tulis tindakan yang diizinkan oleh kebijakan (`UpdateItem`) hanya dapat memodifikasi `attribute-A` atau `attribute-B`.
- Karena kebijakan mengizinkan `UpdateItem`, aplikasi dapat menyisipkan item baru, dan atribut yang tersembunyi akan null dalam item baru. Jika atribut ini diproyeksikan ke dalam `TopScoreDateTimeIndex`, kebijakan memiliki manfaat tambahan untuk mencegah kueri yang menyebabkan pengambilan dari tabel.
- Aplikasi tidak dapat membaca atribut apa pun selain yang tercantum dalam `dynamodb:Attributes`. Dengan adanya kebijakan ini, aplikasi harus menetapkan parameter `Select` ke `SPECIFIC_ATTRIBUTES` dalam permintaan baca, dan hanya atribut dalam daftar izin yang dapat diminta. Untuk permintaan tulis, aplikasi tidak dapat mengatur `ReturnValues` ke `ALL_OLD` atau `ALL_NEW` dan tidak dapat melakukan operasi tulis bersyarat berdasarkan atribut lainnya.

Topik terkait

- [Manajemen Identitas dan Akses untuk Amazon DynamoDB](#)
- [Izin DynamoDB API: Referensi tindakan, sumber daya, dan kondisi](#)

Menggunakan federasi identitas web

Jika Anda menulis aplikasi yang ditargetkan untuk sejumlah besar pengguna, Anda dapat menggunakan federasi identitas web untuk autentikasi dan otorisasi. Federasi identitas web menghilangkan kebutuhan untuk membuat pengguna individual. Sebagai gantinya, pengguna dapat masuk ke penyedia identitas dan kemudian mendapatkan kredensi keamanan sementara dari AWS Security Token Service (AWS STS). Aplikasi kemudian dapat menggunakan kredensi ini untuk mengakses AWS layanan.

Federasi identitas web mendukung penyedia identitas berikut:

- Login with Amazon
- Facebook
- Google

Sumber daya tambahan untuk federasi identitas web

Sumber daya berikut ini dapat membantu Anda mempelajari lebih jauh lagi tentang federasi identitas web:

- Postingan [Web Identity Federation using the AWS SDK for .NET](#) pada blog AWS Developer memberi panduan tentang cara menggunakan federasi identitas web dengan Facebook. Ini termasuk cuplikan kode di C # yang menunjukkan bagaimana mengasumsikan peran IAM dengan identitas web dan cara menggunakan kredensial keamanan sementara untuk mengakses sumber daya. AWS
- [AWS Mobile SDK for iOS](#) dan [AWS Mobile SDK for Android](#) berisi aplikasi sampel. Aplikasi ini mencakup kode yang menunjukkan cara menginvokasi penyedia identitas, kemudian cara menggunakan informasi dari penyedia tersebut untuk mendapatkan dan menggunakan kredensial keamanan sementara.
- Artikel [Federasi Identitas Web dengan Aplikasi Seluler](#) membahas federasi identitas web dan menunjukkan contoh bagaimana menggunakan federasi identitas web untuk mengakses AWS sumber daya.

Contoh kebijakan untuk federasi identitas web

Untuk menunjukkan bagaimana Anda dapat menggunakan federasi identitas web dengan DynamoDB, kunjungi kembali tabel GameScores yang diperkenalkan di [Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci](#). Berikut adalah kunci utama untuk GameScores.

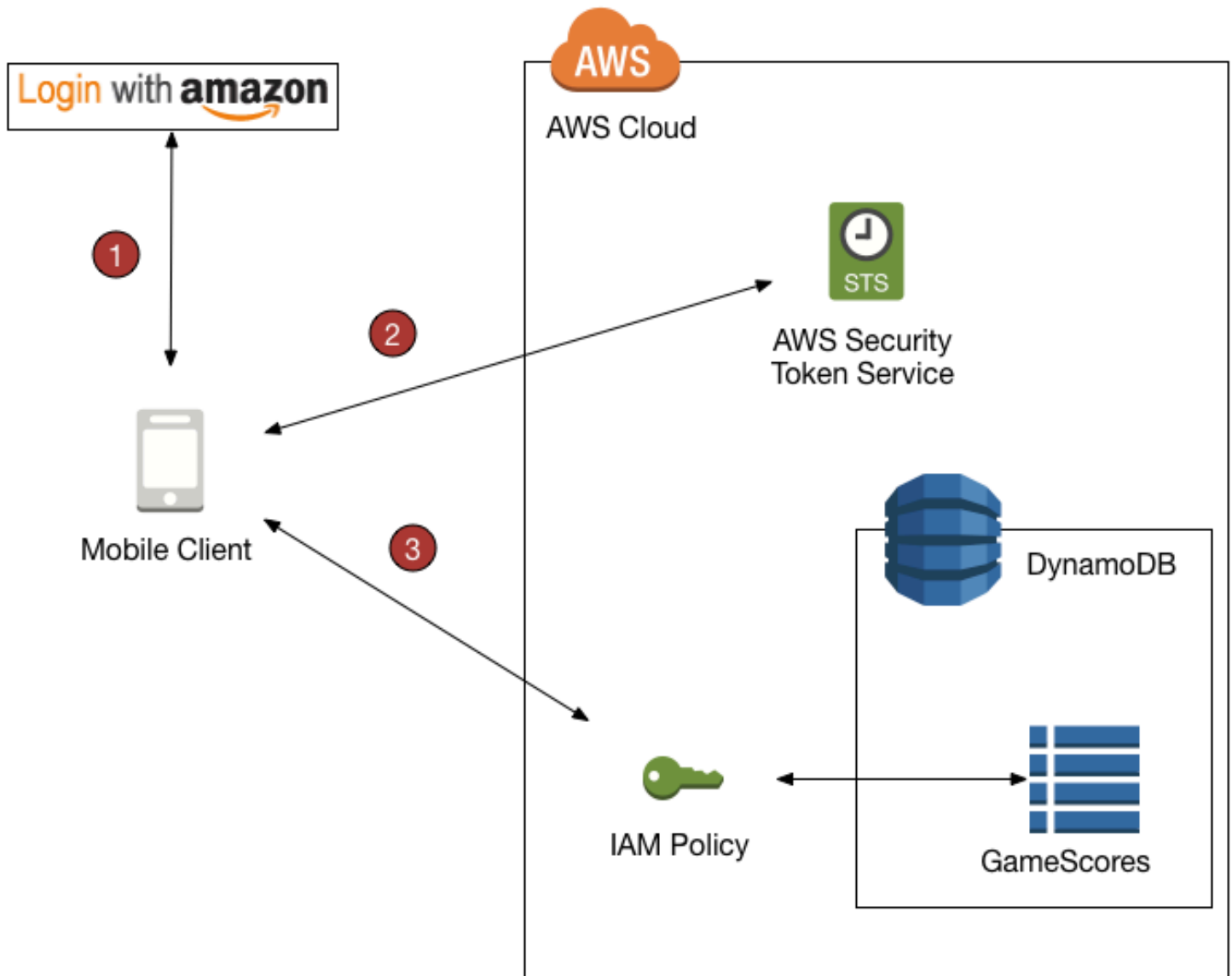
Nama Tabel.	Jenis Kunci Primer	Nama dan Jenis Kunci Partisi	Nama dan Jenis Kunci Urutan
GameScores (<u>UserId</u> , <u>GameTitle</u> , ...)	Komposit	Nama Atribut: UserId Jenis: String	Nama Atribut: GameTitle Jenis: String

Sekarang anggaplah bahwa aplikasi game seluler menggunakan tabel ini, dan aplikasi perlu mendukung ribuan, atau bahkan jutaan, pengguna. Pada skala ini, menjadi sangat sulit untuk mengelola pengguna aplikasi individu, dan untuk menjamin bahwa setiap pengguna hanya dapat mengakses data mereka sendiri dalam GameScorestabel. Untungnya, banyak pengguna sudah memiliki akun dengan penyedia identitas pihak ketiga, seperti Facebook, Google, atau Login with Amazon. Jadi masuk akal untuk menggunakan salah satu penyedia ini untuk tugas autentikasi.

Untuk melakukannya menggunakan federasi identitas web, developer aplikasi harus mendaftarkan aplikasi dengan penyedia identitas (seperti Login with Amazon) dan memperoleh ID aplikasi unik. Selanjutnya, developer perlu membuat peran IAM. (Untuk contoh ini, peran ini diberi nama GameRole.) Peran harus memiliki dokumen kebijakan IAM yang dilampirkan padanya, menentukan kondisi di mana aplikasi dapat mengakses GameScorestabel.

Ketika ingin bermain game, pengguna masuk ke akun Login with Amazon mereka dari dalam aplikasi game. Aplikasi kemudian memanggil AWS Security Token Service (AWS STS), memberikan Login with Amazon app ID dan meminta keanggotaan GameRole. AWS STS mengembalikan AWS kredensi sementara ke aplikasi dan memungkinkannya mengakses GameScorestabel, tunduk pada dokumen GameRolekebijakan.

Diagram berikut menunjukkan bagaimana potongan-potongan ini cocok satu sama lain.



Gambaran umum federasi identitas web

1. Aplikasi memanggil penyedia identitas pihak ketiga untuk mengautentikasi pengguna dan aplikasi. Penyedia identitas mengembalikan token identitas web ke aplikasi.
2. Aplikasi memanggil AWS STS dan meneruskan token identitas web sebagai input. AWS STS mengotorisasi aplikasi dan memberinya kredensi AWS akses sementara. Aplikasi ini diizinkan untuk mengambil peran IAM (GameRole) dan mengakses AWS sumber daya sesuai dengan kebijakan keamanan peran.

3. Aplikasi memanggil DynamoDB untuk mengakses tabel. GameScores Karena telah diasumsikan GameRole, aplikasi tunduk pada kebijakan keamanan yang terkait dengan peran itu. Dokumen kebijakan mencegah aplikasi mengakses data yang bukan milik pengguna.

Sekali lagi, berikut adalah kebijakan keamanan untuk GameRoleitu ditunjukkan di[Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci](#):

```
{
  "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"
        }
      }
    }
  ]
}
```

```
}  
  }  
] }  
}
```

`ConditionKlausa` menentukan item mana `GameScores` yang terlihat oleh aplikasi. Hal ini dilakukan dengan membandingkan ID Login with Amazon untuk nilai kunci partisi `UserId` dalam `GameScores`. Hanya item milik pengguna saat ini yang dapat diproses menggunakan salah satu tindakan DynamoDB yang tercantum dalam kebijakan ini. Item lain dalam tabel tidak dapat diakses. Selain itu, hanya atribut tertentu yang tercantum dalam kebijakan yang dapat diakses.

Mempersiapkan penggunaan federasi identitas web

Jika Anda adalah developer aplikasi dan ingin menggunakan federasi identitas web untuk aplikasi Anda, ikuti langkah-langkah berikut:

1. Daftar sebagai developer dengan penyedia identitas pihak ketiga. Tautan eksternal berikut memberikan informasi tentang mendaftar dengan penyedia identitas yang didukung:
 - [Login dengan Amazon Developer Center](#)
 - [Pendaftaran](#) di situs Facebook
 - [Using OAuth 2.0 to Access Google APIs](#) di situs Google
2. Daftarkan aplikasi Anda dengan penyedia identitas. Saat Anda melakukannya, penyedia memberikan ID yang unik ke aplikasi Anda. Jika Anda ingin aplikasi bekerja dengan beberapa penyedia identitas, Anda perlu memperoleh ID aplikasi dari masing-masing penyedia.
3. Buat satu atau beberapa peran IAM. Anda memerlukan satu peran untuk setiap penyedia identitas untuk setiap aplikasi. Misalnya, Anda dapat membuat peran yang dapat diasumsikan oleh aplikasi saat pengguna masuk menggunakan Login with Amazon, peran kedua untuk aplikasi yang sama di mana pengguna masuk menggunakan Facebook, dan peran ketiga untuk aplikasi di mana pengguna masuk menggunakan Google.

Sebagai bagian dari proses pembuatan peran, Anda harus melampirkan kebijakan IAM ke peran tersebut. Dokumen kebijakan Anda harus menentukan sumber daya DynamoDB yang diperlukan oleh aplikasi Anda, dan izin untuk mengakses sumber daya tersebut.

Untuk informasi selengkapnya, lihat [Tentang Federasi Identitas Web](#) dalam Panduan Pengguna IAM.

Note

Sebagai alternatif AWS Security Token Service, Anda dapat menggunakan Amazon Cognito. Amazon Cognito adalah layanan pilihan untuk mengelola kredensial sementara untuk aplikasi seluler. Untuk informasi selengkapnya, lihat [Mendapatkan kredensial](#) dalam Panduan Developer Amazon Cognito.

Membuat kebijakan IAM menggunakan konsol DynamoDB

Konsol DynamoDB dapat membantu Anda membuat kebijakan IAM untuk digunakan dengan federasi identitas web. Untuk melakukannya, Anda memilih tabel DynamoDB dan menentukan penyedia identitas, tindakan, serta atribut untuk disertakan dalam kebijakan. Konsol DynamoDB kemudian menghasilkan kebijakan yang dapat Anda lampirkan ke peran IAM.

1. [Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/.](https://console.aws.amazon.com/dynamodb/)
2. Di panel navigasi, pilih Tabel.
3. Dalam daftar tabel, pilih tabel yang ingin Anda buat kebijakan IAM.
4. Pilih tombol Tindakan, dan pilih Buat Kebijakan Kontrol Akses.
5. Pilih penyedia identitas, tindakan, dan atribut untuk kebijakan.

Jika pengaturan sudah sesuai keinginan Anda, pilih Buat Kebijakan. Kebijakan yang dihasilkan muncul.

6. Pilih Lihat Dokumentasi, lalu ikuti langkah-langkah yang diperlukan untuk melampirkan kebijakan yang dihasilkan ke peran IAM.

Menulis aplikasi Anda untuk menggunakan federasi identitas web

Untuk menggunakan federasi identitas web, aplikasi Anda harus mengasumsikan peran IAM yang Anda buat. Sejak saat itu, aplikasi menghormati kebijakan akses yang Anda lampirkan pada peran tersebut.

Saat runtime, jika aplikasi Anda menggunakan federasi identitas web, aplikasi harus mengikuti langkah-langkah berikut:

1. Autentikasi dengan penyedia identitas pihak ketiga. Aplikasi Anda harus memanggil penyedia identitas menggunakan antarmuka yang mereka sediakan. Cara yang tepat di mana Anda

mengautentikasi pengguna bergantung pada penyedia dan pada platform apa aplikasi Anda berjalan. Biasanya, jika pengguna belum masuk, penyedia identitas berperan untuk menampilkan halaman masuk untuk penyedia tersebut.

Setelah penyedia identitas mengautentikasi pengguna, penyedia mengembalikan token identitas web ke aplikasi Anda. Format token ini bergantung pada penyedia, tetapi biasanya berupa string karakter yang sangat panjang.

2. Dapatkan kredensi AWS keamanan sementara. Untuk melakukan ini, aplikasi Anda mengirimkan permintaan `AssumeRoleWithWebIdentity` untuk AWS Security Token Service (AWS STS). Permintaan ini berisi hal berikut ini:

- Token identitas web dari langkah sebelumnya
- ID aplikasi dari penyedia identitas
- Amazon Resource Name (ARN) dari peran IAM yang Anda buat untuk penyedia identitas ini untuk aplikasi ini

AWS STS mengembalikan satu set kredensi AWS keamanan yang kedaluwarsa setelah jangka waktu tertentu (3.600 detik, secara default).

Berikut ini adalah sampel permintaan dan respons dari tindakan `AssumeRoleWithWebIdentity` dalam AWS STS. Token identitas web diperoleh dari penyedia identitas Login with Amazon.

```
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-kclvGYM81e...(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+EWUJbbUSHTesjTxqQtM8UKvsM6XAJdA==</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>AR0AJU4SA2VW5SZRF2YMG:web-identity-federation</AssumedRoleId>
  </AssumedRoleUser>
</AssumeRoleWithWebIdentityResult>
<ResponseMetadata>
  <RequestId>c265ac8e-2ae4-11e3-8775-6969323a932d</RequestId>
</ResponseMetadata>
</AssumeRoleWithWebIdentityResponse>

```

3. Akses AWS sumber daya. Respons dari AWS STS berisi informasi yang dibutuhkan aplikasi Anda untuk mengakses sumber daya DynamoDB:

- Bidang `AccessKeyId`, `SecretAccessKey`, dan `SessionToken` berisi kredensial keamanan yang valid hanya untuk pengguna dan aplikasi ini.
- Bidang `Expiration` menandakan batas waktu untuk kredensial ini, setelah itu mereka tidak lagi valid.
- Bidang `AssumedRoleId` berisi nama peran IAM khusus sesi yang telah diasumsikan oleh aplikasi. Aplikasi ini menghormati kontrol akses dalam dokumen kebijakan IAM selama sesi ini.
- Bidang `SubjectFromWebIdentityToken` berisi ID unik yang muncul dalam variabel kebijakan IAM untuk penyedia identitas khusus ini. Berikut ini adalah variabel kebijakan IAM untuk penyedia yang didukung, dan beberapa contoh nilai untuk mereka:

Variabel Kebijakan	Nilai Contoh
<code>\${www.amazon.com:user_id}</code>	amzn1.account.AGJZDKHJKAUUS W6C44CHPEXAMPLE
<code>\${graph.facebook.com:id}</code>	123456789
<code>\${accounts.google.com:sub}</code>	123456789012345678901

Untuk kebijakan IAM contoh di mana variabel kebijakan ini digunakan, lihat [Kebijakan Contoh: Menggunakan ketentuan untuk kontrol parameter terperinci](#).

Untuk informasi selengkapnya tentang cara AWS STS menghasilkan kredensial akses sementara, lihat [Meminta Kredensial Keamanan Sementara di Panduan Pengguna IAM](#).

Izin DynamoDB API: Referensi tindakan, sumber daya, dan kondisi

Saat Anda menyiapkan [Manajemen Identitas dan Akses untuk Amazon DynamoDB](#) dan menulis kebijakan izin yang dapat Anda lampirkan ke identitas IAM (kebijakan berbasis identitas), Anda dapat menggunakan daftar [Kunci tindakan, sumber daya, dan kondisi untuk Amazon DynamoDB](#) dalam Panduan Pengguna IAM sebagai referensi. Halaman ini mencantumkan setiap operasi DynamoDB API, tindakan terkait yang dapat Anda berikan izin untuk melakukan tindakan, dan AWS sumber daya yang dapat Anda berikan izin. Anda menentukan tindakan dalam bidang `Action` kebijakan, dan Anda menentukan nilai sumber daya pada bidang `Resource` kebijakan.

Anda dapat menggunakan kunci kondisi AWS-wide dalam kebijakan DynamoDB Anda untuk menyatakan kondisi. Untuk daftar lengkap kunci AWS-wide, lihat [referensi elemen kebijakan IAM JSON](#) di Panduan Pengguna IAM.

Selain tombol kondisi AWS-wide, DynamoDB memiliki kunci spesifiknya sendiri yang dapat Anda gunakan dalam kondisi. Untuk informasi selengkapnya, lihat [Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci](#).

Topik terkait

- [Manajemen Identitas dan Akses untuk Amazon DynamoDB](#)
- [Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci](#)

Manajemen identitas dan akses di DynamoDB Accelerator

DynamoDB Accelerator (DAX) dirancang untuk bekerja sama dengan DynamoDB guna menambahkan lapisan caching ke aplikasi Anda dengan lancar. Namun, DAX dan DynamoDB memiliki mekanisme kontrol akses terpisah. Kedua layanan menggunakan AWS Identity and Access Management (IAM) untuk menerapkan kebijakan keamanan masing-masing, tetapi model keamanan untuk DAX dan DynamoDB berbeda.

Untuk informasi selengkapnya tentang Manajemen Identitas dan Akses di DAX, lihat [Kontrol akses DAX](#).

Validasi kepatuhan oleh industri untuk DynamoDB

Untuk mempelajari apakah Layanan AWS berada dalam lingkup program kepatuhan khusus, lihat [Layanan AWS di Scope oleh Program](#) Program Kepatuhan yang Anda minati. Untuk informasi umum, lihat [Program Kepatuhan AWS](#).

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat [Mengunduh Laporan di AWS Artifact](#).

Tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS ditentukan berdasarkan sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, serta hukum dan peraturan yang berlaku. AWS menyediakan sumber daya berikut untuk membantu kepatuhan:

- [Panduan Mulai Cepat Keamanan dan Kepatuhan](#) – Panduan deployment ini membahas pertimbangan arsitektur dan menyediakan langkah-langkah untuk melakukan deployment lingkungan dasar di AWS yang menjadi fokus keamanan dan kepatuhan.
- [Merancang Keamanan dan Kepatuhan HIPAA di Amazon Web Services](#) – Laporan resmi ini menjelaskan cara perusahaan dapat menggunakan AWS untuk membuat aplikasi yang memenuhi syarat HIPAA.

Note

Tidak semua Layanan AWS memenuhi syarat HIPAA. Untuk informasi selengkapnya, lihat [Referensi Layanan yang Memenuhi Syarat HIPAA](#).

- [Sumber Daya Kepatuhan AWS](#) – Kumpulan buku kerja dan panduan ini mungkin berlaku untuk industri dan lokasi Anda.
- [Panduan Kepatuhan Pelanggan AWS](#) – Pahami model tanggung jawab bersama melalui lensa kepatuhan. Panduan ini merangkum praktik terbaik untuk mengamankan Layanan AWS dan memetakan panduan kontrol keamanan di banyak kerangka kerja (termasuk National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), dan International Organization for Standardization (ISO)).
- [Mengevaluasi Sumber Daya dengan Aturan](#) dalam Panduan Developer AWS Config – Layanan AWS Config menilai seberapa baik konfigurasi sumber daya Anda dalam mematuhi praktik-praktik internal, pedoman industri, dan regulasi internal.
- [AWS Security Hub](#) – Layanan AWS ini memberikan pandangan komprehensif tentang status keamanan Anda di dalam AWS. Security Hub menggunakan kontrol keamanan untuk

mengevaluasi sumber daya AWS Anda dan memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik. Untuk daftar layanan dan kontrol yang didukung, lihat [Referensi kontrol Security Hub](#).

- [AWS Audit Manager](#) – Layanan AWS ini akan membantu Anda untuk terus-menerus mengaudit penggunaan AWS untuk menyederhanakan cara Anda mengelola risiko dan kepatuhan terhadap regulasi dan standar industri.

Ketahanan dan pemulihan yang Amazon DynamoDB

Infrastruktur global AWS dibangun di sekitar Wilayah AWS dan Availability Zone. AWS Wilayah menyediakan beberapa Availability Zone yang terpisah dan terisolasi secara fisik, yang terhubung dengan jaringan yang memiliki latensi rendah, throughput tinggi, dan sangat berlebihan. Dengan Availability Zone, Anda dapat merancang dan mengoperasikan aplikasi dan basis data yang secara otomatis melakukan fail over di antara Availability Zone tanpa gangguan. Availability Zone lebih tersedia, toleran terhadap kegagalan, dan dapat diskalakan bila dibandingkan dengan satu atau beberapa infrastruktur pusat data tradisional.

Jika Anda harus mereplikasi data atau aplikasi Anda pada jarak geografis yang lebih luas, gunakan Wilayah Lokal AWS. Wilayah Lokal AWS adalah pusat data tunggal yang dirancang untuk melengkapi Wilayah AWS yang sudah ada. Seperti semua Wilayah AWS, Wilayah Lokal AWS benar-benar terisolasi dari Wilayah AWS lainnya.

Untuk informasi selengkapnya tentang Wilayah AWS dan Availability Zone, lihat [Infrastruktur global AWS](#).

Selain infrastruktur global AWS, Amazon DynamoDB menawarkan beberapa fitur untuk membantu mendukung ketahanan data dan kebutuhan cadangan Anda.

Pencadangan dan pemulihan sesuai sesuai sesuai sesuai sesuai sesuai sesuai sesuai sesuai

DynamoDB menyediakan kemampuan pencadangan sesuai permintaan. Hal ini memungkinkan Anda untuk membuat cadangan penuh dari tabel Anda untuk retensi jangka panjang dan arsip. Untuk informasi selengkapnya, lihat [Pencadangan dan pemulihan sesuai sesuai sesuai sesuai sesuai sesuai sesuai sesuai sesuai sesuai sesuai sesuai sesuai sesuai permintaan](#).

P oint-in-time pemulihan

oint-in-timePemulihan P membantu melindungi tabel DynamoDB Anda dari penulisan atau penghapusan operasi yang tidak disengaja. Dengan pemulihan titik waktu, Anda tidak perlu khawatir tentang membuat, memelihara, atau menjadwalkan pencadangan sesuai permintaan. Untuk informasi selengkapnya, lihat [oint-in-timePemulihan P untuk DynamoDB](#).

Tabel global yang disinkronkan di seluruh AWS wilayah

DynamoDB secara otomatis menyebar data dan lalu lintas untuk tabel Anda atas jumlah server yang cukup untuk menangani throughput dan persyaratan penyimpanan Anda, sambil mempertahankan performa yang konsisten dan cepat. Semua data Anda disimpan pada solid-state disk (SSD) dan secara otomatis direplikasi di beberapa Availability Zones di sebuah Wilayah AWS, menyediakan ketersediaan tinggi dan daya tahan data bawaan. Anda dapat menggunakan tabel global untuk menjaga tabel DynamoDB tersinkronisasi di Wilayah AWS.

Keamanan infrastruktur di Amazon DynamoDB

Sebagai layanan terkelola, Amazon DynamoDB dilindungi oleh AWS prosedur keamanan jaringan global yang dijelaskan [dalam perlindungan Infrastruktur yang terletak di Well-Architected Framework](#).
AWS

Anda menggunakan panggilan API yang AWS dipublikasikan untuk mengakses DynamoDB melalui jaringan. Klien dapat menggunakan TLS (Transport Layer Security) versi 1.2 atau 1.3. Klien juga harus mendukung cipher suite dengan perfect forward secrecy (PFS) seperti Ephemeral Diffie-Hellman (DHE) atau Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Sebagian besar sistem modern seperti Java 7 dan sistem yang lebih baru mendukung mode ini. Selain itu, permintaan harus ditandatangani menggunakan ID kunci akses dan kunci akses rahasia yang dikaitkan dengan pengguna utama IAM. Atau Anda dapat menggunakan [AWS Security Token Service](#) (AWS STS) untuk membuat kredensial keamanan sementara untuk menandatangani permintaan.

Anda juga dapat menggunakan titik akhir cloud privat virtual (VPC) untuk DynamoDB guna mengaktifkan instans Amazon EC2 di VPC Anda untuk menggunakan alamat IP privatnya guna mengakses DynamoDB tanpa tersambung ke internet publik. Untuk informasi selengkapnya, lihat [Menggunakan titik akhir Amazon VPC untuk mengakses DynamoDB](#).

Menggunakan titik akhir Amazon VPC untuk mengakses DynamoDB

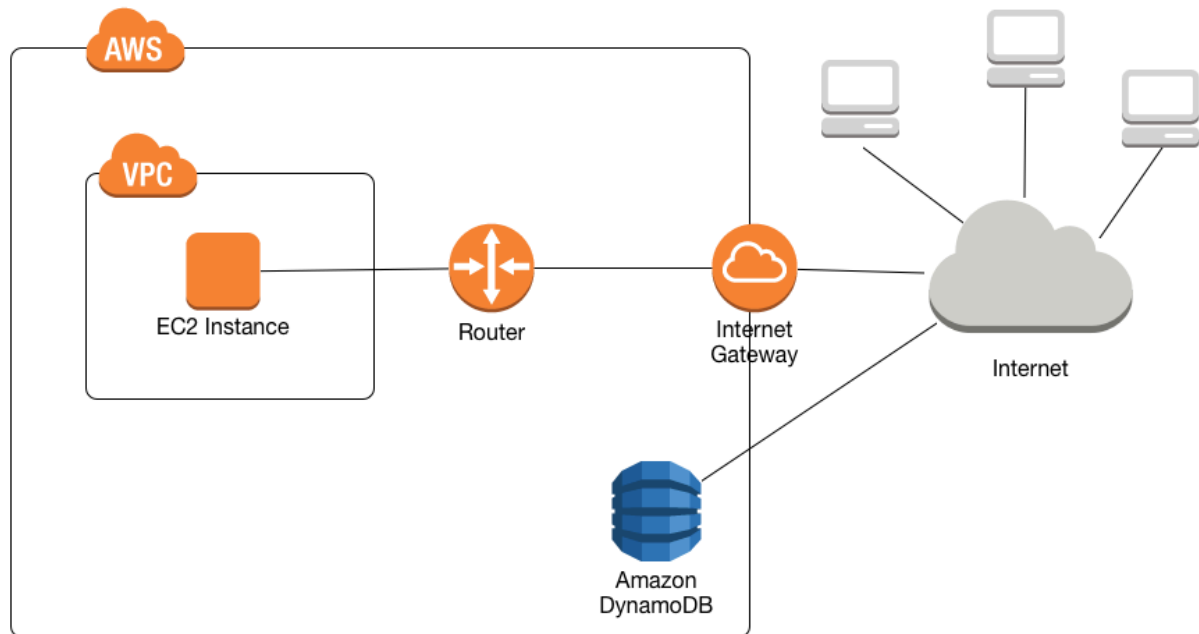
Untuk alasan keamanan, banyak pelanggan AWS menjalankan aplikasinya dalam lingkungan Amazon Virtual Private Cloud (Amazon VPC). Dengan Amazon VPC, Anda dapat meluncurkan instans Amazon EC2 ke virtual private cloud, yang secara logis terisolasi dari jaringan lain—termasuk internet publik. Dengan Amazon VPC, Anda dapat mengontrol rentang alamat IP, subnet, tabel perutean, gateway jaringan, dan pengaturan keamanan.

Note

Jika Anda membuat Akun AWS setelah 4 Desember 2013, maka Anda sudah memiliki VPC default di masing-masing Wilayah AWS VPC default siap untuk Anda gunakan—Anda dapat segera menggunakannya tanpa harus melakukan langkah-langkah konfigurasi tambahan. Untuk informasi selengkapnya, lihat [VPC Default dan Subnet Default](#) di Panduan Pengguna Amazon VPC.

Untuk mengakses internet publik, VPC Anda harus memiliki gateway internet—router virtual yang menghubungkan VPC Anda ke internet. Hal ini memungkinkan aplikasi yang berjalan di Amazon EC2 di VPC Anda mengakses sumber daya internet, seperti Amazon DynamoDB.

Secara default, komunikasi ke dan dari DynamoDB menggunakan protokol HTTPS, yang melindungi lalu lintas jaringan menggunakan enkripsi SSL/TLS. Diagram berikut menunjukkan instans Amazon EC2 dalam VPC yang mengakses DynamoDB, dengan meminta DynamoDB menggunakan gateway internet daripada titik akhir VPC.



Banyak pelanggan memiliki kekhawatiran yang sah mengenai privasi dan keamanan saat mengirim dan menerima data melalui internet publik. Anda dapat mengatasi masalah ini menggunakan jaringan privat virtual (VPN) untuk merutekan semua lalu lintas jaringan DynamoDB melalui infrastruktur jaringan perusahaan Anda. Namun, pendekatan ini dapat menimbulkan tantangan bandwidth dan ketersediaan.

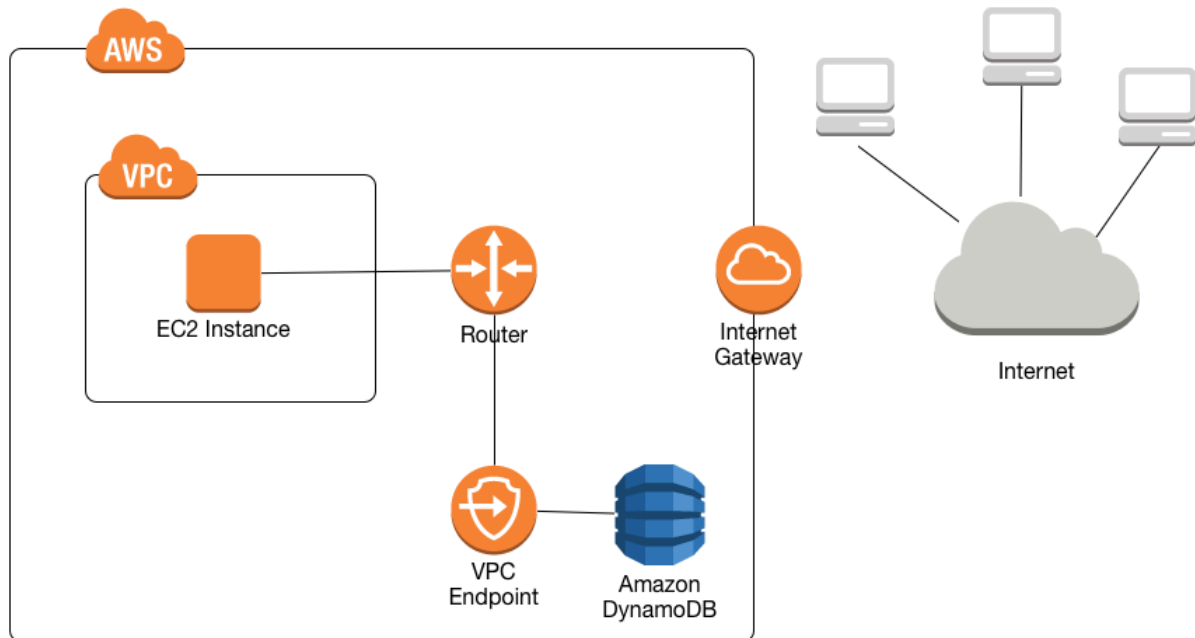
Titik akhir VPC untuk DynamoDB dapat mengatasi tantangan ini. Titik akhir VPC untuk DynamoDB memungkinkan instans Amazon EC2 di VPC menggunakan alamat IP privat untuk mengakses DynamoDB tanpa tersambung ke internet publik. Instans EC2 Anda tidak memerlukan alamat IP publik, dan Anda tidak memerlukan gateway internet, perangkat NAT, atau gateway privat virtual di VPC Anda. Anda menggunakan kebijakan titik akhir untuk mengontrol akses ke DynamoDB. Lalu lintas antara VPC Anda dan layanan AWS tidak meninggalkan jaringan Amazon.

Note

Bahkan ketika Anda menggunakan alamat IP publik, semua komunikasi VPC antara instance dan layanan yang dihosting tetap pribadi di AWS dalam jaringan. AWS Paket yang berasal dari jaringan AWS dengan tujuan di jaringan AWS tetap berada di jaringan global AWS, kecuali lalu lintas ke atau dari Wilayah AWS Tiongkok.

Saat Anda membuat titik akhir VPC untuk DynamoDB, setiap permintaan ke titik akhir DynamoDB dalam Wilayah (misalnya, `dynamodb.us-west-2.amazonaws.com`) dirutekan ke titik akhir DynamoDB privat dalam jaringan Amazon. Anda tidak perlu memodifikasi aplikasi yang berjalan pada instans EC2 di VPC Anda. Nama titik akhir tetap sama, namun rute ke DynamoDB tetap sepenuhnya berada dalam jaringan Amazon, dan tidak mengakses internet publik.

Diagram berikut menunjukkan bagaimana instans EC2 di VPC dapat menggunakan titik akhir VPC untuk mengakses DynamoDB.



Untuk informasi selengkapnya, lihat [the section called “Tutorial: Menggunakan titik akhir VPC untuk DynamoDB”](#).

Berbagi titik akhir Amazon VPC dan DynamoDB

Untuk mengaktifkan akses ke layanan DynamoDB melalui titik akhir gateway subnet VPC, Anda harus memiliki izin akun pemilik untuk subnet VPC tersebut.

Setelah titik akhir gateway subnet VPC diberikan akses ke DynamoDB, akun AWS mana pun yang memiliki akses ke subnet tersebut dapat menggunakan DynamoDB. Hal ini berarti semua pengguna akun dalam subnet VPC dapat menggunakan tabel DynamoDB apa pun yang dapat mereka akses. Ini termasuk tabel DynamoDB yang terkait dengan akun yang berbeda dari subnet VPC. Pemilik

subnet VPC masih dapat membatasi pengguna tertentu dalam subnet untuk menggunakan layanan DynamoDB melalui titik akhir gateway, sesuai kebijakannya.

Tutorial: Menggunakan titik akhir VPC untuk DynamoDB

Bagian ini memandu Anda dalam menyiapkan dan menggunakan titik akhir VPC untuk DynamoDB.

Topik

- [Langkah 1: Luncurkan instans Amazon EC2](#)
- [Langkah 2: Konfigurasi instans Amazon EC2 Anda](#)
- [Langkah 3: Membuat titik akhir VPC untuk DynamoDB](#)
- [Langkah 4: \(Opsional\) Hapus](#)

Langkah 1: Luncurkan instans Amazon EC2

Pada langkah ini, Anda meluncurkan instans Amazon EC2 di Amazon VPC default Anda. Anda kemudian dapat membuat dan menggunakan titik akhir VPC untuk DynamoDB.

1. Buka konsol Amazon EC2 di <https://console.aws.amazon.com/ec2/>.
2. Pilih Luncurkan Instans, dan lakukan hal berikut:

Langkah 1: Pilih Amazon Machine Image (AMI)

- Di bagian atas daftar AMI, buka Amazon Linux AMI, dan pilih Pilih.

Langkah 2: Pilih Jenis Instans

- Di bagian atas daftar jenis instans, pilih t2.micro.
- Pilih Berikutnya: Konfigurasi Detail Instans.

Langkah 3: Konfigurasi Detail Instans

- Buka Jaringan, lalu pilih VPC default Anda.

Pilih Berikutnya: Tambahkan Penyimpanan.

Langkah 4: Tambahkan Penyimpanan

- Lewati langkah ini dengan memilih Selanjutnya: Instans Tag.

Langkah 5: Instans Tag

- Lewati langkah ini dengan memilih Selanjutnya: Konfigurasi Grup Keamanan.

Langkah 6: Konfigurasi Grup Keamanan

- Pilih grup keamanan yang ada.
- Dalam daftar grup keamanan, pilih default. Ini adalah grup keamanan default untuk VPC Anda.
- Pilih Berikutnya: Tinjau dan Luncurkan.

Langkah 7: Tinjau Peluncuran Instans

- Pilih Luncurkan.
3. Di jendela Pilih pasangan kunci yang ada atau buat pasangan kunci baru, lakukan salah satu hal berikut:
 - Jika Anda tidak memiliki pasangan kunci Amazon EC2, pilih Buat pasangan kunci baru dan ikuti petunjuknya. Anda akan diminta untuk mengunduh file kunci privat (file .pem); Anda akan memerlukan file ini nanti saat Anda masuk ke instans Amazon EC2 Anda.
 - Jika Anda sudah memiliki pasangan kunci Amazon EC2 yang sudah ada, buka Pilih pasangan kunci dan pilih pasangan kunci Anda dari daftar. Anda harus sudah memiliki file kunci privat (file .pem) yang tersedia untuk masuk ke instans Amazon EC2 Anda.
 4. Ketika Anda telah mengonfigurasi pasangan kunci, pilih Luncurkan Instans.
 5. Kembali ke halaman beranda konsol Amazon EC2 dan pilih instans yang Anda luncurkan. Pada panel bawah, di tab Deskripsi, cari DNS Publik untuk instans Anda. Misalnya: `ec2-00-00-00-00.us-east-1.compute.amazonaws.com`.

Catat nama DNS publik ini, karena Anda akan memerlukannya di langkah berikutnya dalam tutorial ini ([Langkah 2: Konfigurasi instans Amazon EC2 Anda](#)).

Note

Diperlukan waktu beberapa menit agar instans Amazon EC2 Anda tersedia. Sebelum Anda melanjutkan ke langkah berikutnya, pastikan Status Instans dalam kondisi `running` dan semua Pemeriksaan Status telah lulus.

Langkah 2: Konfigurasi instans Amazon EC2 Anda

Ketika instans Amazon EC2 Anda tersedia, Anda akan dapat masuk ke dalamnya dan mempersiapkannya untuk penggunaan pertama.

Note

Langkah-langkah berikut menganggap bahwa Anda terhubung ke instans Amazon EC2 dari komputer yang menjalankan Linux. Terkait cara lain untuk terhubung, lihat [Menghubungkan ke Instans Linux Anda](#) di Panduan Pengguna Amazon EC2 untuk Instans Linux.

1. Anda perlu mengotorisasi lalu lintas SSH masuk ke instans Amazon EC2 Anda. Untuk melakukan ini, Anda akan membuat grup keamanan EC2 baru, lalu menetapkan grup keamanan tersebut ke instans EC2 Anda.
 - a. Pada panel navigasi, pilih Grup Keamanan.
 - b. Pilih Buat Grup Keamanan. Di jendela Buat Grup Keamanan, lakukan hal berikut:
 - Nama grup keamanan—ketikkan nama untuk grup keamanan Anda. Misalnya: `my-ssh-access`
 - Deskripsi—ketikkan deskripsi singkat untuk grup keamanan.
 - VPC—pilih VPC default Anda.
 - Di bagian Aturan grup keamanan, pilih Tambahkan Aturan dan lakukan hal berikut:
 - Jenis—pilihlah SSH.
 - Sumber—pilih IP Saya.
 - c. Di panel navigasi, pilih Contoh.

Jika pengaturan sudah sesuai keinginan Anda, pilih Buat.

- d. Pilih instans Amazon EC2 yang diluncurkan di [Langkah 1: Luncurkan instans Amazon EC2](#).
 - e. Pilih Tindakan → Jaringan → Ubah Grup Keamanan.
 - f. Di Ubah Grup Keamanan, pilih grup keamanan yang Anda buat sebelumnya dalam prosedur ini (misalnya: my-ssh-access). Saat ini grup keamanan default juga harus dipilih. Bila pengaturan sesuai keinginan Anda, pilih Tetapkan Grup Keamanan.
2. Gunakan perintah ssh untuk masuk ke instans Amazon EC2 Anda, seperti dalam contoh berikut.

```
ssh -i my-keypair.pem ec2-user@public-dns-name
```

Anda perlu menentukan file kunci privat Anda (file .pem) dan nama DNS publik dari instans Anda. (Lihat [Langkah 1: Luncurkan instans Amazon EC2](#)).

ID login adalah ec2-user. Tidak diperlukan kata sandi.

3. Konfigurasi kredensial AWS Anda, seperti yang ditunjukkan di bawahini. Masukkan ID kunci akses, kunci rahasia, dan nama Wilayah default AWS saat diminta.

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]:
```

Anda sekarang siap untuk membuat titik akhir VPC untuk DynamoDB.

Langkah 3: Membuat titik akhir VPC untuk DynamoDB

Pada langkah ini, Anda akan membuat titik akhir VPC untuk DynamoDB dan mengujinya untuk memastikannya berfungsi.

1. Sebelum memulai, verifikasi bahwa Anda dapat berkomunikasi dengan DynamoDB menggunakan titik akhir publiknya.

```
aws dynamodb list-tables
```

Outputnya akan menampilkan daftar tabel DynamoDB yang Anda miliki saat ini. (Jika Anda tidak memiliki tabel apa pun, daftarnya akan kosong.).

2. Verifikasi bahwa DynamoDB adalah layanan yang tersedia untuk membuat titik akhir VPC di Wilayah AWS saat ini. (Perintah ditampilkan dalam teks tebal, diikuti dengan contoh output.)

```
aws ec2 describe-vpc-endpoint-services

{
  "ServiceNames": [
    "com.amazonaws.us-east-1.s3",
    "com.amazonaws.us-east-1.dynamodb"
  ]
}
```

Dalam contoh output, DynamoDB adalah salah satu layanan yang tersedia, sehingga Anda dapat melanjutkan dengan membuat titik akhir VPC untuk layanan tersebut.

3. Tentukan pengidentifikasi VPC Anda.

```
aws ec2 describe-vpcs

{
  "Vpcs": [
    {
      "VpcId": "vpc-0bbc736e",
      "InstanceTenancy": "default",
      "State": "available",
      "DhcpOptionsId": "dopt-8454b7e1",
      "CidrBlock": "172.31.0.0/16",
      "IsDefault": true
    }
  ]
}
```

Dalam contoh output, ID VPC adalah `vpc-0bbc736e`.

4. Buat titik akhir VPC. Untuk parameter `--vpc-id`, tentukan ID VPC dari langkah sebelumnya. Gunakan parameter `--route-table-ids` untuk mengasosiasikan titik akhir dengan tabel rute Anda.

```
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. Verifikasi bahwa Anda dapat mengakses DynamoDB melalui titik akhir VPC.

```
aws dynamodb list-tables
```

Jika mau, Anda dapat mencoba beberapa perintah AWS CLI lainnya untuk DynamoDB. Untuk informasi selengkapnya, lihat [Referensi Perintah AWS CLI](#).

Langkah 4: (Opsional) Hapus

Jika Anda ingin menghapus sumber daya yang Anda buat dalam tutorial ini, ikuti prosedur berikut:

Untuk menghapus titik akhir VPC Anda untuk DynamoDB

1. Masuk ke instans Amazon EC2 Anda.
2. Tentukan ID titik akhir VPC.

```
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"
  }
}
```

```
}  
}
```

Dalam contoh output, ID titik akhir VPC adalah `vpce-9b15e2f2`.

3. Hapus titik akhir VPC.

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2  
  
{  
  "Unsuccessful": []  
}
```

Array kosong `[]` menunjukkan keberhasilan (tidak ada permintaan yang gagal).

Untuk menghentikan instans Amazon EC2 Anda

1. Buka konsol Amazon EC2 di <https://console.aws.amazon.com/ec2/>.
2. Di panel navigasi, pilih Contoh.
3. Pilih instans Amazon EC2 Anda.
4. Pilih Tindakan, Status Instans, Akhiri.
5. Dalam jendela konfirmasi, pilih Ya, Hentikan.

AWS PrivateLink untuk DynamoDB

Dengan AWS PrivateLink untuk DynamoDB, Anda dapat menyediakan antarmuka titik akhir Amazon VPC (titik akhir antarmuka) di cloud pribadi virtual Anda (Amazon VPC). Titik akhir ini dapat diakses langsung dari aplikasi yang ada di tempat melalui VPN dan AWS Direct Connect, atau berbeda dengan peering Wilayah AWS [VPC Amazon](#). Menggunakan AWS PrivateLink dan antarmuka endpoint, Anda dapat menyederhanakan konektivitas jaringan pribadi dari aplikasi Anda ke DynamoDB.

Aplikasi di VPC Anda tidak memerlukan alamat IP publik untuk berkomunikasi dengan titik akhir VPC antarmuka DynamoDB untuk operasi DynamoDB. Endpoint antarmuka diwakili oleh satu atau lebih antarmuka jaringan elastis (ENI) yang diberi alamat IP pribadi dari subnet di VPC Amazon Anda. Permintaan ke DynamoDB melalui titik akhir antarmuka tetap berada di jaringan Amazon. Anda juga dapat mengakses titik akhir antarmuka di VPC Amazon dari aplikasi lokal AWS Direct Connect

melalui AWS Virtual Private Network atau (VPC). Untuk informasi selengkapnya tentang cara menghubungkan VPC Amazon dengan jaringan lokal, lihat [Panduan AWS Direct Connect Pengguna dan Panduan Pengguna AWS Site-to-Site VPN](#)

Untuk informasi umum tentang titik akhir antarmuka, lihat [Antarmuka Amazon VPC endpoints AWS PrivateLink\(\)](#) di [Panduan AWS PrivateLink](#)

Topik

- [Jenis titik akhir Amazon VPC untuk Amazon DynamoDB](#)
- [Pertimbangan saat menggunakan AWS PrivateLink untuk Amazon DynamoDB](#)
- [Membuat titik akhir Amazon VPC](#)
- [Mengakses titik akhir antarmuka Amazon DynamoDB](#)
- [Mengakses tabel DynamoDB dan mengontrol operasi API dari titik akhir antarmuka DynamoDB](#)
- [Memperbarui konfigurasi DNS on-premise](#)
- [Membuat kebijakan endpoint Amazon VPC untuk DynamoDB](#)

Jenis titik akhir Amazon VPC untuk Amazon DynamoDB

Anda dapat menggunakan dua jenis titik akhir Amazon VPC untuk mengakses Amazon DynamoDB: titik akhir gateway dan titik akhir antarmuka (dengan menggunakan AWS PrivateLink). Titik akhir gateway adalah gateway yang Anda tentukan dalam tabel rute untuk mengakses DynamoDB dari VPC Amazon Anda melalui jaringan. AWS Titik akhir antarmuka memperluas fungsionalitas titik akhir gateway dengan menggunakan alamat IP pribadi untuk merutekan permintaan ke DynamoDB dari dalam VPC Amazon Anda, di tempat, atau dari VPC Amazon di tempat lain dengan menggunakan peering Amazon VPC atau Wilayah AWS AWS Transit Gateway. Untuk informasi lebih lanjut, lihat [Apa itu peering Amazon VPC?](#) dan [Transit Gateway vs Pengintip VPC Amazon](#).

Titik akhir antarmuka kompatibel dengan titik akhir gateway. Jika Anda memiliki titik akhir gateway yang ada di VPC Amazon, Anda dapat menggunakan kedua jenis titik akhir di VPC Amazon yang sama.

Titik akhir Gateway untuk DynamoDB

Endpoint antarmuka untuk DynamoDB

Dalam kedua kasus, lalu lintas jaringan Anda tetap berada di AWS jaringan.

Titik akhir Gateway untuk DynamoDB	Endpoint antarmuka untuk DynamoDB
Menggunakan alamat IP publik Amazon DynamoDB	Gunakan alamat IP pribadi dari Amazon VPC Anda untuk mengakses Amazon DynamoDB
Jangan izinkan mengakses dari on-premise	Izinkan mengakses dari on-premise
Jangan izinkan akses dari yang lain Wilayah AWS	Izinkan akses dari titik akhir VPC Amazon di titik lain dengan Wilayah AWS menggunakan peering Amazon VPC atau AWS Transit Gateway
Tidak ditagih	Ditagih

Untuk informasi selengkapnya tentang titik akhir gateway, lihat titik akhir [Gateway Amazon VPC](#) di Panduan.AWS PrivateLink

Pertimbangan saat menggunakan AWS PrivateLink untuk Amazon DynamoDB

Pertimbangan Amazon VPC berlaku untuk Amazon AWS PrivateLink DynamoDB. Untuk informasi selengkapnya, lihat [Pertimbangan antarmuka titik akhir](#) dan [AWS PrivateLink kuota](#) dalam AWS PrivateLink Panduan. Selain itu, larangan berikut juga berlaku.

AWS PrivateLink untuk Amazon DynamoDB tidak mendukung hal berikut:

- [Titik akhir Standar Proses Informasi Federal \(FIPS\)](#)
- Keamanan Lapisan Pengangkutan (TLS) 1.1
- Layanan Sistem Nama Domain Pribadi dan Hybrid (DNS)

AWS PrivateLink saat ini tidak didukung untuk Amazon DynamoDB Streams titik akhir.

Anda dapat mengirimkan hingga 50.000 permintaan per detik untuk setiap AWS PrivateLink titik akhir yang Anda aktifkan.

Note

Batas waktu konektivitas jaringan ke AWS PrivateLink titik akhir tidak berada dalam lingkup respons kesalahan DynamoDB dan perlu ditangani dengan tepat oleh aplikasi Anda yang terhubung ke titik akhir. PrivateLink

Membuat titik akhir Amazon VPC

Untuk membuat titik akhir antarmuka VPC Amazon, lihat [Membuat titik akhir](#) di Panduan.AWS PrivateLink

Mengakses titik akhir antarmuka Amazon DynamoDB

Saat Anda membuat titik akhir antarmuka, DynamoDB menghasilkan dua jenis nama DNS DynamoDB spesifik titik akhir: Regional dan zonal.

- Nama DNS Regional mencakup ID titik akhir VPC Amazon yang unik, pengenalan layanan, Wilayah AWS dan namanya. `vpce.amazonaws.com` Misalnya, untuk ID titik akhir VPC Amazon `vpce-1a2b3c4d`, nama DNS yang dihasilkan mungkin mirip dengan `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com`
- Nama DNS Zonal mencakup Zona Ketersediaan—misalnya, `vpce-1a2b3c4d-5e6f-us-east-1a.dynamodb.us-east-1.vpce.amazonaws.com`. Anda dapat menggunakan opsi ini jika arsitektur Anda mengisolasi Zona Ketersediaan. Contoh, Anda bisa menggunakannya untuk kontainer kesalahan atau untuk mengurangi biaya transfer data Regional.

Nama DNS DynamoDB khusus titik akhir dapat diselesaikan dari domain DNS publik DynamoDB.

Mengakses tabel DynamoDB dan mengontrol operasi API dari titik akhir antarmuka DynamoDB

Anda dapat menggunakan AWS CLI atau AWS SDK untuk mengakses tabel DynamoDB dan mengontrol operasi API melalui titik akhir antarmuka DynamoDB.

AWS CLI contoh

Untuk mengakses tabel DynamoDB atau operasi API kontrol DynamoDB melalui titik akhir antarmuka DynamoDB dalam perintah, gunakan parameter `dan`. AWS CLI `--region --endpoint-url`

Contoh: Buat titik akhir VPC

```
aws ec2 create-vpc-endpoint \  
--region us-east-1 \  
--service-name dynamodb-service-name \  
--vpc-id client-vpc-id \  
--subnet-ids client-subnet-id \  
--vpc-endpoint-type Interface \  
--security-group-ids client-sg-id
```

Contoh: Memodifikasi titik akhir VPC

```
aws ec2 modify-vpc-endpoint \  
--region us-east-1 \  
--vpc-endpoint-id client-vpc-endpoint-id \  
--policy-document policy-document \  
--add-security-group-ids security-group-ids \  
# any additional parameters needed, see Privatelink documentation for more details
```

Contoh: Daftar tabel menggunakan URL endpoint

Dalam contoh berikut, ganti Region `us-east-1` dan nama DNS dari `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` ID endpoint VPC dengan informasi Anda sendiri.

```
aws dynamodb --region us-east-1 --endpoint https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com list-tables
```

AWS Contoh SDK

Untuk mengakses tabel DynamoDB atau operasi API kontrol DynamoDB melalui titik akhir antarmuka DynamoDB saat menggunakan SDK, perbarui SDK Anda ke versi terbaru. AWS Kemudian, konfigurasi klien Anda untuk menggunakan URL titik akhir untuk mengakses tabel atau operasi API kontrol DynamoDB melalui titik akhir antarmuka DynamoDB.

SDK for Python (Boto3)

Contoh: Gunakan URL endpoint untuk mengakses tabel DynamoDB

Dalam contoh berikut, ganti Wilayah `us-east-1` dan ID titik akhir VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` dengan informasi Anda sendiri.

```
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

Contoh: Gunakan URL endpoint untuk mengakses tabel DynamoDB

Dalam contoh berikut, ganti Wilayah `us-east-1` dan ID titik akhir VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` dengan informasi Anda sendiri.

```
//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

Contoh: Gunakan URL titik akhir untuk mengakses bucket S3

Dalam contoh berikut, ganti ID endpoint Region `us-east-1` dan VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` dengan informasi Anda sendiri.

```
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();
```

Memperbarui konfigurasi DNS on-premise

Saat menggunakan nama DNS khusus titik akhir untuk mengakses titik akhir antarmuka DynamoDB, Anda tidak perlu memperbarui penyelesai DNS lokal Anda. Anda dapat menyelesaikan nama DNS spesifik titik akhir dengan alamat IP pribadi titik akhir antarmuka dari domain DNS DynamoDB publik.

Menggunakan titik akhir antarmuka untuk mengakses DynamoDB tanpa titik akhir gateway atau gateway internet di Amazon VPC

Titik akhir antarmuka di VPC Amazon Anda dapat merutekan aplikasi VPC di Amazon dan aplikasi lokal ke DynamoDB melalui jaringan Amazon, seperti yang diilustrasikan dalam diagram berikut.

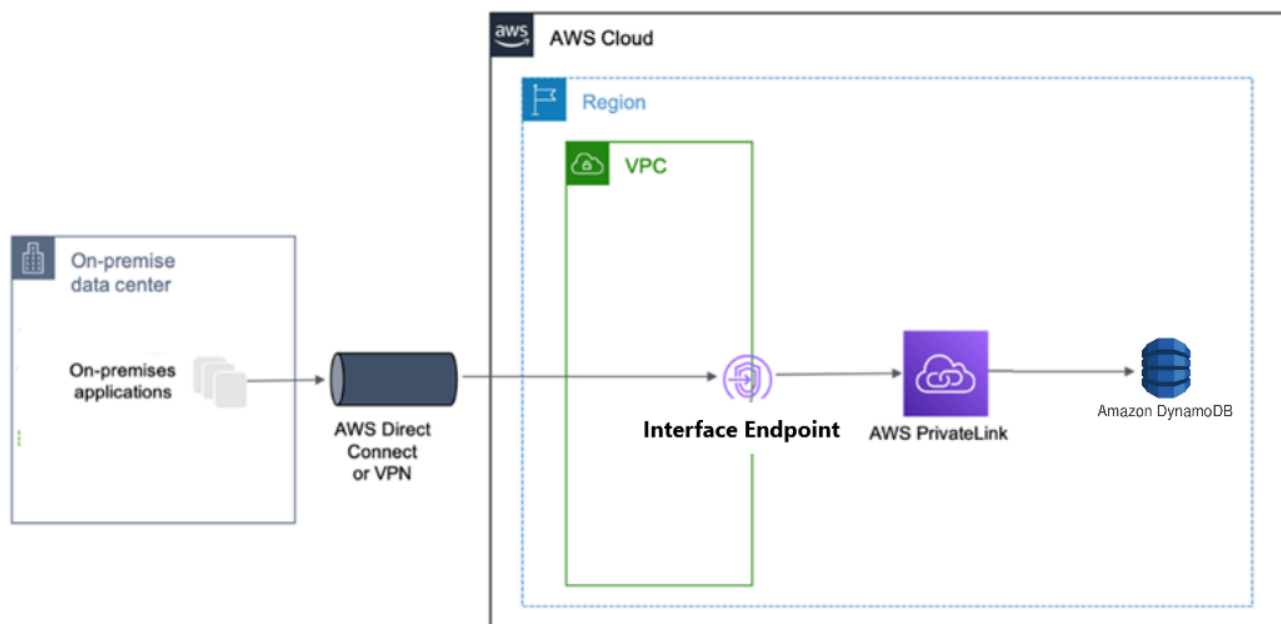


Diagram ini menggambarkan hal sebagai berikut:

- Jaringan lokal Anda menggunakan AWS Direct Connect atau menyambung AWS VPN ke Amazon VPC A.
- Aplikasi Anda lokal dan di Amazon VPC A menggunakan nama DNS khusus titik akhir untuk mengakses DynamoDB melalui titik akhir antarmuka DynamoDB.
- Aplikasi lokal mengirim data ke titik akhir antarmuka di Amazon VPC melalui AWS Direct Connect (atau). AWS VPN AWS PrivateLink memindahkan data dari titik akhir antarmuka ke DynamoDB melalui jaringan. AWS

- Aplikasi VPC di Amazon juga mengirim lalu lintas ke titik akhir antarmuka. AWS PrivateLink memindahkan data dari titik akhir antarmuka ke DynamoDB melalui jaringan. AWS

Menggunakan titik akhir gateway dan titik akhir antarmuka bersama-sama di Amazon VPC yang sama untuk mengakses DynamoDB

Anda dapat membuat titik akhir antarmuka dan mempertahankan titik akhir gateway yang ada di VPC Amazon yang sama, seperti yang ditunjukkan diagram berikut. Dengan mengambil pendekatan ini, Anda mengizinkan aplikasi VPC di Amazon untuk terus mengakses DynamoDB melalui titik akhir gateway, yang tidak ditagih. Kemudian, hanya aplikasi lokal Anda yang akan menggunakan titik akhir antarmuka untuk mengakses DynamoDB. Untuk mengakses DynamoDB dengan cara ini, Anda harus memperbarui aplikasi lokal Anda untuk menggunakan nama DNS khusus titik akhir untuk DynamoDB.

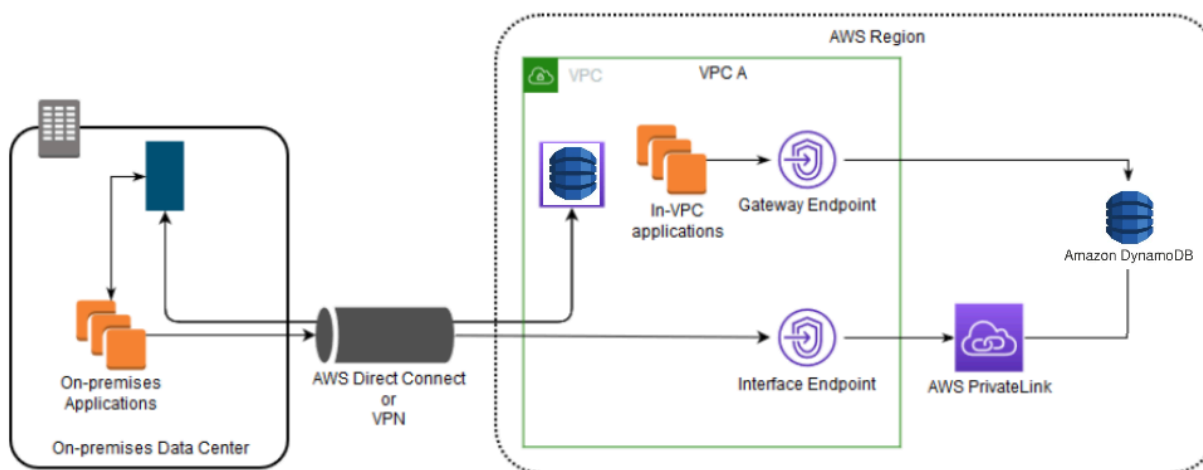


Diagram ini menggambarkan hal sebagai berikut:

- Aplikasi lokal menggunakan nama DNS khusus titik akhir untuk mengirim data ke titik akhir antarmuka dalam VPC Amazon melalui (atau) AWS Direct Connect AWS VPN AWS PrivateLink memindahkan data dari titik akhir antarmuka ke DynamoDB melalui jaringan. AWS
- Menggunakan nama DynamoDB Regional default, aplikasi VPC di Amazon mengirim data ke titik akhir gateway yang terhubung ke DynamoDB melalui jaringan. AWS

Untuk informasi selengkapnya tentang titik akhir gateway, lihat [Gateway titik akhir VPC Amazon](#) di Panduan Pengguna Amazon VPC.

Membuat kebijakan endpoint Amazon VPC untuk DynamoDB

Anda dapat melampirkan kebijakan endpoint ke endpoint Amazon VPC yang mengontrol akses ke DynamoDB. Kebijakan titik akhir menentukan informasi berikut:

- Prinsip AWS Identity and Access Management (IAM) yang dapat melakukan tindakan
- Tindakan-tindakan yang dapat dilakukan
- Sumber daya yang padanya tindakan dapat dilakukan

Topik

- [Contoh: Membatasi akses ke tabel tertentu dari titik akhir VPC Amazon](#)

Contoh: Membatasi akses ke tabel tertentu dari titik akhir VPC Amazon

Anda dapat membuat kebijakan endpoint yang membatasi akses hanya ke tabel DynamoDB tertentu. Jenis kebijakan ini berguna jika Anda memiliki kebijakan lain Layanan AWS di VPC Amazon Anda yang menggunakan tabel. Kebijakan tabel berikut membatasi akses hanya ke file. *DOC-EXAMPLE-TABLE* Untuk menggunakan kebijakan endpoint ini, ganti *DOC-EXAMPLE-TABLE* dengan nama tabel Anda.

```
{
  "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::DOC-EXAMPLE-TABLE",
                  "arn:aws:dynamodb::DOC-EXAMPLE-TABLE/*"]
    }
  ]
}
```


Analisis konfigurasi dan kelemahan di Amazon DynamoDB

AWS menangani tugas-tugas keamanan dasar seperti sistem operasi tamu (OS) dan patching basis data, konfigurasi firewall, dan pemulihan bencana. Prosedur ini telah ditinjau dan disertifikasi oleh pihak ketiga yang sesuai. Untuk detail selengkapnya, lihat sumber daya berikut:

- [Validasi Kepatuhan Amazon DynamoDB](#)
- [Model tanggung jawab bersama](#)
- [Amazon Web Services: Ringkasan proses keamanan](#)(whitepaper)

Praktik terbaik keamanan berikut ini juga membahas analisis konfigurasi dan kerentanan di Amazon DynamoDB:

- [Memantau kepatuhan DynamoDB dengan Aturan AWS Config](#)
- [Memantau konfigurasi DynamoDB dengan AWS Config](#)

Praktik terbaik keamanan untuk Amazon DynamoDB

Amazon DynamoDB menyediakan sejumlah fitur keamanan untuk dipertimbangkan saat Anda mengembangkan dan menerapkan kebijakan keamanan Anda sendiri. Praktik terbaik berikut adalah pedoman umum dan tidak mewakili solusi keamanan yang lengkap. Karena praktik terbaik ini mungkin tidak sesuai atau cukup untuk lingkungan Anda, anggap sebagai pertimbangan yang membantu dan bukan sebagai resep.

Topik

- [Praktik terbaik keamanan pencegahan DynamoDB](#)
- [Praktik terbaik keamanan detektif DynamoDB](#)

Praktik terbaik keamanan pencegahan DynamoDB

Praktik terbaik berikut dapat membantu Anda mengantisipasi dan mencegah insiden keamanan di Amazon DynamoDB.

Enkripsi saat istirahat

DynamoDB mengenkripsi saat istirahat semua data pengguna yang disimpan dalam tabel, indeks, aliran, dan cadangan menggunakan kunci enkripsi yang disimpan dalam [AWS Key Management Service \(AWS KMS\)](#). Ini memberi lapisan perlindungan data tambahan dengan mengamankan data Anda dari akses yang tidak sah ke penyimpanan pokok.

Anda dapat menentukan apakah DynamoDB harus menggunakan Kunci milik AWS (tipe enkripsi default), sebuah Kunci yang dikelola AWS, atau kunci yang dikelola pelanggan untuk mengenkripsi data pengguna. Untuk informasi selengkapnya, lihat [Enkripsi Amazon DynamoDB saat Istirahat](#).

Menggunakan peran IAM untuk mengautentikasi akses ke DynamoDB

Untuk pengguna, aplikasi, dan layanan AWS lainnya agar dapat mengakses DynamoDB, mereka harus menyertakan kredensial AWS yang valid dalam permintaan AWS API mereka. Anda tidak boleh menyimpan kredensial AWS secara langsung di aplikasi atau instans EC2. Ini adalah kredensial jangka panjang yang tidak dirotasi secara otomatis dan dapat menimbulkan dampak bisnis yang signifikan jika dibobol. IAM role memungkinkan Anda memperoleh kunci akses sementara yang dapat digunakan untuk mengakses layanan dan sumber daya AWS.

Untuk informasi selengkapnya, lihat [Manajemen Identitas dan Akses untuk Amazon DynamoDB](#).

Menggunakan kebijakan IAM untuk otorisasi dasar DynamoDB

Saat memberikan izin, Anda memutuskan siapa yang mendapatkannya, API DynamoDB mana yang mereka dapatkan izinnya, dan tindakan khusus yang ingin Anda izinkan pada sumber daya tersebut. Menerapkan akses hak istimewa yang terkecil adalah hal mendasar dalam mengurangi risiko keamanan dan dampak yang dapat disebabkan oleh kesalahan atau niat jahat.

Memberikan kebijakan izin kepada identitas IAM (yaitu, pengguna, grup, dan peran) dan dengan demikian memberikan izin untuk melakukan operasi pada sumber daya DynamoDB.

Anda dapat melakukan hal ini dengan cara berikut:

- [AWSKebijakan terkelola \(yang telah ditentukan\)](#)
- [Kebijakan yang dikelola pelanggan](#)

Gunakan syarat kebijakan IAM untuk kontrol akses terperinci

Ketika memberikan izin di DynamoDB, Anda dapat menetapkan syarat yang menentukan bagaimana kebijakan izin berlaku. Menerapkan akses hak istimewa yang terkecil adalah hal mendasar dalam mengurangi risiko keamanan dan dampak yang dapat disebabkan oleh kesalahan atau niat jahat.

Anda dapat menetapkan syarat saat memberikan izin menggunakan kebijakan IAM. Misalnya, Anda dapat melakukan hal berikut:

- Mengizinkan pengguna mengakses hanya-baca pada item dan atribut tertentu dalam tabel atau indeks sekunder.
- Mengizinkan pengguna mengakses hanya-tulis pada atribut tertentu dalam tabel, berdasarkan identitas pengguna tersebut.

Untuk informasi selengkapnya, lihat [Menggunakan Ketentuan Kebijakan IAM untuk Kontrol Akses Detail](#).

Menggunakan kebijakan dan VPC endpoint untuk mengakses DynamoDB

Jika Anda hanya memerlukan akses ke DynamoDB dari dalam virtual private cloud (VPC), Anda harus menggunakan VPC endpoint untuk membatasi akses hanya dari VPC yang diperlukan. Melakukan hal ini akan mencegah lalu lintas melintasi internet terbuka dan tunduk pada lingkungan tersebut.

Menggunakan VPC endpoint untuk DynamoDB akan memungkinkan Anda untuk mengontrol dan membatasi akses menggunakan hal berikut:

- Kebijakan VPC endpoint – Kebijakan ini diterapkan pada VPC endpoint DynamoDB. Kebijakan tersebut memungkinkan Anda untuk mengontrol dan membatasi akses API ke tabel DynamoDB.
- Kebijakan IAM — Dengan menggunakan `aws:sourceVpce` ketentuan kebijakan yang dilampirkan pada pengguna, grup, atau peran, Anda dapat menegaskan bahwa semua akses ke tabel DynamoDB adalah melalui titik akhir VPC yang ditentukan.

Untuk informasi selengkapnya, lihat [titik akhir Amazon DynamoDB](#).

Pertimbangkan enkripsi sisi klien

Kami menyarankan Anda merencanakan strategi enkripsi sebelum menerapkan tabel Anda di DynamoDB. Jika Anda menyimpan data sensitif atau rahasia di DynamoDB, pertimbangkan untuk menyertakan enkripsi sisi klien dalam paket Anda. Dengan cara ini Anda dapat mengenkripsi data sedekat mungkin dengan asalnya, dan memastikan perlindungannya selama siklus hidupnya. Mengenkripsi data sensitif Anda saat transit dan istirahat membantu memastikan bahwa data plaintext Anda tidak tersedia untuk pihak ketiga mana pun.

[AWS Database Encryption SDK untuk DynamoDB](#) adalah pustaka perangkat lunak yang membantu Anda melindungi data tabel Anda sebelum Anda mengirimkannya ke DynamoDB. Ini

mengkripsi, menandatangani, memverifikasi, dan mendekripsi item tabel DynamoDB Anda. Anda mengontrol atribut mana yang dienkripsi dan ditandatangani.

Praktik terbaik keamanan detektif DynamoDB

Praktik terbaik untuk Amazon DynamoDB berikut dapat membantu mendeteksi potensi kelemahan dan insiden keamanan.

Gunakan AWS CloudTrail untuk memantau penggunaan kunci KMS yang dikelola AWS

Jika Anda menggunakan [Kunci yang dikelola AWS](#) untuk enkripsi saat istirahat, penggunaan kunci ini login ke AWS CloudTrail. CloudTrail memberikan visibilitas ke aktivitas pengguna dengan merekam tindakan yang diambil pada akun Anda. CloudTrail mencatat informasi penting tentang setiap tindakan, termasuk siapa yang membuat permintaan, layanan yang digunakan, tindakan yang dilakukan, parameter untuk tindakan, dan elemen respons yang dikembalikan oleh AWS layanan. Informasi ini membantu Anda melacak perubahan yang diterapkan pada sumber daya AWS Anda dan memecahkan masalah operasional. CloudTrail membuatnya lebih mudah untuk memastikan kepatuhan terhadap kebijakan internal dan standar peraturan.

Anda dapat menggunakan CloudTrail untuk mengaudit penggunaan kunci. CloudTrail membuat file log yang berisi riwayat panggilan AWS API dan peristiwa terkait untuk akun Anda. Berkas log ini mencakup semua permintaan API AWS KMS yang diajukan menggunakan AWS Management Console, SDK AWS, dan alat baris perintah, selain yang dibuat melalui layanan AWS terintegrasi. Anda dapat menggunakan file log ini untuk mendapatkan informasi tentang kapan kunci KMS digunakan, operasi yang diminta, identitas pemohon, alamat IP tempat permintaan itu berasal, dan sebagainya. Untuk informasi selengkapnya, lihat [Pencatatan Panggilan API AWS KMS dengan AWS CloudTrail](#) dan [Panduan Pengguna AWS CloudTrail](#).

Memantau operasi DynamoDB menggunakan CloudTrail

CloudTrail dapat memantau kedua peristiwa pesawat kontrol dan peristiwa pesawat data. Operasi bidang kontrol memungkinkan Anda membuat dan mengelola tabel DynamoDB. Operasi ini juga memungkinkan Anda bekerja dengan indeks, aliran, dan objek lain yang tergantung pada tabel. Operasi bidang data memungkinkan Anda membuat, membaca, memperbarui, dan menghapus tindakan (juga disebut CRUD) pada data dalam tabel. Beberapa operasi bidang data juga memungkinkan Anda membaca data dari indeks sekunder. Untuk mengaktifkan pencatatan peristiwa bidang data CloudTrail, Anda harus mengaktifkan pencatatan aktivitas API bidang data di CloudTrail. Lihat [Pencatatan peristiwa data untuk jejak](#) untuk informasi selengkapnya.

Ketika aktivitas terjadi di DynamoDB, aktivitas tersebut direkam dalam CloudTrail peristiwa bersama dengan peristiwa AWS layanan lainnya dalam riwayat peristiwa. Untuk informasi selengkapnya, lihat [Pencatatan Operasi DynamoDB menggunakan AWS CloudTrail](#). Anda dapat melihat, mencari, dan mengunduh peristiwa terbaru di akun AWS Anda. Untuk informasi selengkapnya, lihat [Melihat CloudTrail Acara dengan Riwayat Peristiwa](#) di Panduan AWS CloudTrail Pengguna.

Untuk catatan berkelanjutan tentang peristiwa di akun AWS Anda, termasuk peristiwa untuk DynamoDB, buatlah [jejak](#). Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon Simple Storage Service (Amazon S3). Secara default, ketika Anda membuat jejak di konsol, jejak akan diterapkan ke semua Wilayah AWS. Jejak mencatat peristiwa dari semua Wilayah di partisi AWS dan mengirimkan berkas log ke bucket S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi layanan AWS lainnya untuk menganalisis lebih lanjut dan bertindak berdasarkan data kejadian yang dikumpulkan di log CloudTrail.

Menggunakan DynamoDB Streams untuk memantau operasi bidang data

DynamoDB terintegrasi dengan AWS Lambda sehingga Anda dapat membuat pemicu – potongan kode yang secara otomatis menanggapi peristiwa di DynamoDB Streams. Dengan pemicu, Anda dapat membangun aplikasi yang bereaksi terhadap modifikasi data dalam tabel DynamoDB.

Jika mengaktifkan DynamoDB Streams pada tabel, Anda dapat menghubungkan aliran Amazon Resource Name (ARN) dengan fungsi Lambda yang Anda tulis. Segera setelah item dalam tabel dimodifikasi, catatan baru muncul di aliran tabel. AWS Lambda menerima aliran dan memanggil fungsi Lambda Anda secara serentak ketika mendeteksi rekaman aliran baru. Fungsi Lambda dapat melakukan tindakan apa pun yang Anda tentukan, seperti mengirim notifikasi atau memulai alur kerja.

Sebagai contoh, lihat [Tutorial: Menggunakan AWS Lambda Dengan Amazon DynamoDB Streams](#). Contoh ini menerima input peristiwa DynamoDB, memproses pesan yang dikandungnya, dan menulis beberapa data peristiwa yang masuk ke Amazon Logs. CloudWatch

Memantau konfigurasi DynamoDB dengan AWS Config

Anda dapat terus memantau dan merekam perubahan konfigurasi pada sumber daya AWS Anda menggunakan [AWS Config](#). Anda juga dapat menggunakan AWS Config untuk menginventaris sumber daya AWS Anda. Ketika terdeteksi perubahan dari keadaan sebelumnya, pemberitahuan Amazon Simple Notification Service (Amazon SNS) dapat dikirimkan untuk Anda guna meninjau dan mengambil tindakan. Ikuti petunjuk di [Menyiapkan AWS Config dengan Konsol](#), memastikan bahwa jenis sumber daya DynamoDB disertakan.

Anda dapat mengonfigurasi AWS Config untuk streaming perubahan konfigurasi dan pemberitahuan untuk topik Amazon SNS. Misalnya, ketika sumber daya diperbarui, Anda dapat memperoleh pemberitahuan yang dikirim ke email Anda, sehingga Anda dapat melihat perubahan tersebut. Anda juga dapat diberi tahu kapan AWS Config mengevaluasi aturan kustom atau terkelola terhadap sumber daya Anda.

Sebagai contoh, lihat [Pemberitahuan yang Dikirimkan AWS Config ke topik Amazon SNS](#) di Panduan Developer AWS Config.

Memantau kepatuhan DynamoDB pada aturan AWS Config

AWS Config terus melacak perubahan konfigurasi yang terjadi di antara sumber daya Anda. Hal itu memeriksa apakah perubahan ini melanggar salah satu syarat dalam aturan Anda. Jika sumber daya melanggar aturan, AWS Config menandai sumber daya dan aturan sebagai tidak patuh.

Dengan menggunakan AWS Config untuk mengevaluasi konfigurasi sumber daya, Anda dapat menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan. AWS Config menyediakan [Aturan terkelola AWS](#), yang telah ditetapkan, aturan yang disesuaikan yang digunakan AWS Config untuk mengevaluasi apakah sumber daya AWS Anda mematuhi praktik terbaik umum.

Tag sumber daya DynamoDB Anda untuk identifikasi dan otomatisasi

Anda dapat menetapkan metadata ke sumber daya AWS Anda dalam bentuk tag. Setiap tag adalah label sederhana yang terdiri dari kunci yang ditetapkan pelanggan dan nilai opsional yang memudahkan untuk mengelola, mencari, dan mem-filter sumber daya.

Tagging memungkinkan implementasi kontrol berkelompok. Meskipun tidak ada jenis tag yang melekat, tagging memungkinkan Anda untuk mengelompokkan sumber daya berdasarkan tujuan, pemilik, lingkungan, atau kriteria lainnya. Berikut ini adalah beberapa contoh:

- Keamanan – Digunakan untuk menentukan persyaratan seperti enkripsi.
- Kerahasiaan – Sebuah pengidentifikasi untuk dukungan sumber daya tingkat kerahasiaan data tertentu.
- Lingkungan – Digunakan untuk membedakan antara pengembangan, pengujian, dan infrastruktur produksi.

Untuk informasi selengkapnya, lihat [Strategi Tagging AWS](#) dan [Tagging untuk DynamoDB](#).

Pantau penggunaan Amazon DynamoDB Anda karena terkait dengan praktik terbaik keamanan dengan menggunakan AWS Security Hub

Hub Keamanan menggunakan kontrol keamanan untuk mengevaluasi konfigurasi sumber daya dan standar keamanan untuk membantu Anda mematuhi berbagai kerangka kerja kepatuhan.

Untuk informasi selengkapnya tentang penggunaan Hub Keamanan untuk mengevaluasi sumber daya DynamoDB, lihat [kontrol Amazon DynamoDB di Panduan Pengguna](#). AWS Security Hub

Pencatatan dan pemantauan

Pemantauan adalah bagian penting dari menjaga keandalan, ketersediaan, dan kinerja DynamoDB dan solusi Anda. AWS Anda harus mengumpulkan data pemantauan dari semua bagian AWS solusi Anda sehingga Anda dapat lebih mudah men-debug kegagalan multi-titik jika terjadi. AWS menyediakan beberapa alat untuk memantau sumber daya DynamoDB Anda dan menanggapi potensi insiden:

Topik

- [Pencatatan dan pemantauan di DynamoDB](#)
- [Pencatatan log dan pemantauan di DynamoDB Accelerator](#)
- [Menganalisis akses data menggunakan wawasan CloudWatch kontributor untuk DynamoDB](#)
- [Menggunakan Pemberitahuan Notifikasi Pengguna AWS Pengguna dengan Amazon DynamoDB](#)

Pencatatan dan pemantauan di DynamoDB

Pemantauan adalah bagian penting dari menjaga keandalan, ketersediaan, dan kinerja DynamoDB dan solusi Anda. AWS Anda harus mengumpulkan data pemantauan dari semua bagian AWS solusi Anda sehingga Anda dapat lebih mudah men-debug kegagalan multi-titik, jika terjadi. Namun, sebelum mulai memantau DynamoDB, Anda harus membuat rencana pemantauan yang mencakup jawaban atas pertanyaan berikut:

- Apa saja sasaran pemantauan Anda?
- Sumber daya apa yang akan Anda pantau?
- Seberapa sering Anda akan memantau sumber daya ini?
- Alat pemantauan apa yang akan Anda gunakan?
- Siapa yang akan melakukan tugas pemantauan?
- Siapa yang harus diberi tahu saat terjadi kesalahan?

Langkah berikutnya adalah menetapkan garis dasar untuk performa DynamoDB normal di lingkungan Anda, dengan mengukur performa di berbagai waktu dan dalam kondisi beban yang berbeda. Saat memantau DynamoDB, Anda harus mempertimbangkan untuk menyimpan data pemantauan historis. Data yang disimpan ini akan memberi Anda dasar untuk membandingkan data performa

saat ini, mengidentifikasi pola performa normal dan anomali performa, dan merancang metode untuk mengatasi masalah.

Untuk menetapkan baseline, Anda harus, setidaknya, menggunakan item berikut:

- Jumlah unit kapasitas baca atau tulis yang digunakan selama periode waktu tertentu, sehingga Anda dapat melacak jumlah penggunaan throughput yang Anda sediakan.
- Permintaan yang melampaui kapasitas baca atau tulis yang disediakan pada tabel selama jangka waktu tertentu, sehingga Anda dapat menentukan permintaan mana yang melampaui kuota throughput yang disediakan pada tabel.
- Kesalahan sistem, sehingga Anda dapat menentukan apakah ada permintaan yang menghasilkan kesalahan.

Topik

- [Alat-alat pemantauan](#)
- [Pemantauan CloudWatch dengan Amazon](#)
- [Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail](#)

Alat-alat pemantauan

AWS menyediakan alat yang dapat Anda gunakan untuk memantau DynamoDB. Anda dapat mengonfigurasi beberapa alat tersebut agar melakukan pemantauan untuk Anda; dan beberapa alat lainnya memerlukan intervensi manual. Sebaiknya Anda mengotomatisasi tugas pemantauan sebanyak mungkin.

Alat pemantauan otomatis

Anda dapat menggunakan alat pemantauan otomatis berikut untuk memantau DynamoDB dan melapor saat terjadi masalah:

- CloudWatch Alarm Amazon — Tonton satu metrik selama periode waktu yang Anda tentukan, dan lakukan satu atau beberapa tindakan berdasarkan nilai metrik relatif terhadap ambang batas tertentu selama beberapa periode waktu. Tindakannya adalah pemberitahuan yang dikirim ke topik Amazon Simple Notification Service (Amazon SNS) atau kebijakan Amazon EC2 Auto Scaling. CloudWatch alarm tidak memanggil tindakan hanya karena mereka berada dalam keadaan tertentu; negara harus telah berubah dan dipertahankan untuk sejumlah periode tertentu. Untuk informasi selengkapnya, lihat [Pemantauan CloudWatch dengan Amazon](#).

- Amazon CloudWatch Logs — Pantau, simpan, dan akses file log Anda dari AWS CloudTrail atau sumber lain. Untuk informasi selengkapnya, lihat [Memantau File Log](#) di Panduan CloudWatch Pengguna Amazon.
- CloudWatch Acara Amazon — Cocokkan acara dan arahkan ke satu atau beberapa fungsi atau aliran target untuk membuat perubahan, menangkap informasi status, dan mengambil tindakan korektif. Untuk informasi selengkapnya, lihat [Apa itu CloudWatch Acara Amazon](#) di Panduan CloudWatch Pengguna Amazon.
- AWS CloudTrail Pemantauan Log - Bagikan file log antar akun, pantau file CloudTrail log secara real time dengan mengirimkannya ke CloudWatch Log, menulis aplikasi pemrosesan log di Java, dan validasi bahwa file log Anda tidak berubah setelah pengiriman oleh CloudTrail. Untuk informasi selengkapnya, lihat [Bekerja dengan File CloudTrail Log](#) di Panduan AWS CloudTrail Pengguna.

Alat pemantauan manual

Bagian penting lainnya dari pemantauan DynamoDB melibatkan pemantauan secara manual item-item yang tidak tercakup oleh CloudWatch alarm. DynamoDB CloudWatch,, Trusted Advisor, dan dasbor konsol AWS lainnya memberikan tampilan at-a-glance status lingkungan Anda. AWS Sebaiknya Anda juga memeriksa file log di Amazon DynamoDB.

- Dasbor DynamoDB menunjukkan:
 - Peringatan terbaru
 - Kapasitas total
 - Kondisi layanan
- CloudWatch halaman rumah menunjukkan:
 - Alarm dan status saat ini
 - Grafik alarm dan sumber daya
 - Status kondisi layanan

Selain itu, Anda dapat menggunakan CloudWatch untuk melakukan hal berikut:

- Membuat [dasbor yang disesuaikan](#) untuk memantau layanan yang penting bagi Anda
- Data metrik grafik untuk memecahkan masalah dan mengungkap tren
- Cari dan telusuri semua metrik AWS sumber daya Anda
- Membuat dan mengedit alarm untuk menerima notifikasi terkait masalah

Pemantauan CloudWatch dengan Amazon

Anda dapat memantau Amazon DynamoDB CloudWatch menggunakan, yang mengumpulkan dan memproses data mentah dari DynamoDB menjadi metrik hampir real-time yang dapat dibaca. Statistik ini disimpan selama jangka waktu tertentu, sehingga Anda dapat mengakses informasi historis untuk perspektif yang lebih baik tentang performa aplikasi web atau layanan Anda. Secara default, data metrik DynamoDB dikirim secara otomatis. CloudWatch Untuk informasi lebih lanjut, lihat [Apa itu Amazon CloudWatch?](#) dan [Retensi metrik](#) di Panduan CloudWatch Pengguna Amazon.

Topik

- [Dimensi dan Metrik DynamoDB](#)
- [Bagaimana cara menggunakan metrik DynamoDB?](#)
- [Memahami dan menganalisis waktu respons DynamoDB](#)
- [Membuat CloudWatch alarm untuk memantau DynamoDB](#)

Dimensi dan Metrik DynamoDB

Ketika Anda berinteraksi dengan DynamoDB, ia mengirimkan metrik dan dimensi berikut ke CloudWatch Anda dapat menggunakan prosedur berikut untuk melihat metrik DynamoDB.

Cara melihat metrik (konsol)

Metrik dikelompokkan terlebih dahulu berdasarkan namespace layanan, lalu berdasarkan berbagai kombinasi dimensi dalam setiap namespace.

1. Buka CloudWatch konsol di <https://console.aws.amazon.com/cloudwatch/>.
2. Pada panel navigasi, silakan pilih Metrik.

Note

Anda juga dapat memilih namespace Penggunaan untuk melihat metrik penggunaan DynamoDB. Untuk informasi selengkapnya tentang metrik penggunaan, lihat [metrik penggunaan AWS](#).

3. Pilih namespace DynamoDB.

Untuk melihat metrik (CLI)

- Pada prompt perintah, gunakan perintah berikut:

```
aws cloudwatch list-metrics --namespace "AWS/DynamoDB"
```

Melihat metrik dan dimensi

CloudWatch menampilkan metrik berikut untuk DynamoDB:

Metrik DynamoDB

Note

Amazon CloudWatch menggabungkan metrik ini pada interval satu menit:

- ConditionalCheckFailedRequests
- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits
- ReadThrottleEvents
- ReturnedBytes
- ReturnedItemCount
- ReturnedRecordsCount
- SuccessfulRequestLatency
- SystemErrors
- TimeToLiveDeletedItemCount
- ThrottledRequests
- TransactionConflict
- UserErrors
- WriteThrottleEvents

Untuk semua metrik DynamoDB lainnya, granularitas agregasi adalah lima menit.

Tidak semua statistik, seperti Average atau Sum, berlaku untuk setiap metrik. Namun, semua nilai ini tersedia melalui konsol Amazon DynamoDB, atau dengan menggunakan konsol, AWS CLI, AWS atau SDK CloudWatch untuk semua metrik.

Dalam daftar berikut, setiap metrik memiliki serangkaian statistik valid yang berlaku metrik tersebut.

Daftar Metrik yang Tersedia

- [AccountMaxReads](#)
- [AccountMaxTableLevelReads](#)
- [AccountMaxTableLevelWrites](#)
- [AccountMaxWrites](#)
- [AccountProvisionedReadCapacityUtilization](#)
- [AccountProvisionedWriteCapacityUtilization](#)
- [AgeOfOldestUnreplicatedRecord](#)
- [ConditionalCheckFailedRequests](#)
- [ConsumedChangeDataCaptureUnits](#)
- [ConsumedReadCapacityUnits](#)
- [ConsumedWriteCapacityUnits](#)
- [FailedToReplicateRecordCount](#)
- [MaxProvisionedTableReadCapacityUtilization](#)
- [MaxProvisionedTableWriteCapacityUtilization](#)
- [OnlineIndexConsumedWriteCapacity](#)
- [OnlineIndexPercentageProgress](#)
- [OnlineIndexThrottleEvents](#)
- [PendingReplicationCount](#)
- [ProvisionedReadCapacityUnits](#)
- [ProvisionedWriteCapacityUnits](#)
- [ReadThrottleEvents](#)
- [ReplicationLatency](#)
- [ReturnedBytes](#)
- [ReturnedItemCount](#)
- [ReturnedRecordsCount](#)

- [SuccessfulRequestLatency](#)
- [SystemErrors](#)
- [TimeToLiveDeletedItemCount](#)
- [ThrottledPutRecordCount](#)
- [ThrottledRequests](#)
- [TransactionConflict](#)
- [UserErrors](#)
- [WriteThrottleEvents](#)

AccountMaxReads

Jumlah maksimum unit kapasitas baca yang dapat digunakan oleh suatu akun. Batas ini tidak berlaku untuk tabel atau indeks sekunder global sesuai permintaan.

Unit: Count

Statistik Valid:

- **Maximum** – Jumlah maksimum unit kapasitas baca yang dapat digunakan oleh suatu akun.

AccountMaxTableLevelReads

Jumlah maksimum unit kapasitas baca yang dapat digunakan oleh tabel atau indeks sekunder global suatu akun. Untuk tabel sesuai permintaan, batas ini membatasi unit permintaan baca maksimum yang dapat digunakan oleh tabel atau indeks sekunder global.

Unit: Count

Statistik Valid:

- **Maximum** – Jumlah maksimum unit kapasitas baca yang dapat digunakan oleh tabel atau indeks sekunder global suatu akun.

AccountMaxTableLevelWrites

Jumlah maksimum unit kapasitas tulis yang dapat digunakan oleh tabel atau indeks sekunder global suatu akun. Untuk tabel sesuai permintaan, batas ini membatasi unit permintaan tulis maksimum yang dapat digunakan oleh tabel atau indeks sekunder global.

Unit: Count

Statistik Valid:

- **Maximum** – Jumlah maksimum unit kapasitas tulis yang dapat digunakan oleh tabel atau indeks sekunder global suatu akun.

AccountMaxWrites

Jumlah maksimum unit kapasitas tulis yang dapat digunakan oleh suatu akun. Batas ini tidak berlaku untuk tabel atau indeks sekunder global sesuai permintaan.

Unit: Count

Statistik Valid:

- **Maximum** – Jumlah maksimum unit kapasitas tulis yang dapat digunakan oleh suatu akun.

AccountProvisionedReadCapacityUtilization

Persentase unit kapasitas baca yang disediakan dan digunakan oleh suatu akun.

Unit: Percent

Statistik Valid:

- **Maximum** – Persentase maksimum unit kapasitas baca yang disediakan dan digunakan oleh suatu akun.
- **Minimum** – Persentase minimum unit kapasitas baca yang disediakan dan digunakan oleh suatu akun.
- **Average** – Persentase rata-rata unit kapasitas baca yang disediakan dan digunakan oleh suatu akun. Metrik diterbitkan selama interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas baca yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

AccountProvisionedWriteCapacityUtilization

Persentase unit kapasitas tulis yang disediakan dan digunakan oleh suatu akun.

Unit: Percent

Statistik Valid:

- **Maximum** – Persentase maksimum unit kapasitas tulis yang disediakan dan digunakan oleh suatu akun.
- **Minimum** – Persentase minimum unit kapasitas tulis yang disediakan dan digunakan oleh suatu akun.
- **Average** – Persentase rata-rata unit kapasitas tulis yang disediakan dan digunakan oleh suatu akun. Metrik diterbitkan selama interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas tulis yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

AgeOfOldestUnreplicatedRecord

Waktu yang berlalu sejak catatan akan direplikasi ke Kinesis data stream yang muncul pertama kali di tabel DynamoDB.

Unit: Milliseconds


Dimensi: `TableName`, `DelegatedOperation`

Statistik Valid:

- **Maximum**.
- **Minimum**.
- **Average**.

ConditionalCheckFailedRequests

Jumlah upaya gagal untuk melakukan penulisan bersyarat. Operasi `PutItem`, `UpdateItem`, dan `DeleteItem` memungkinkan Anda memberikan syarat logis yang harus bernilai `true` sebelum operasi dapat dilanjutkan. Jika syarat ini bernilai `false`, `ConditionalCheckFailedRequests` bertambah satu. `ConditionalCheckFailedRequests` juga bertambah satu untuk pernyataan `Perbarui` dan `Hapus PartiQL`, dengan syarat logis disediakan dan syarat tersebut bernilai `false`.

 Note

Penulisan bersyarat yang gagal akan menghasilkan kesalahan HTTP 400 (Permintaan Buruk). Peristiwa ini tercermin dalam metrik `ConditionalCheckFailedRequests`, tapi tidak di metrik `UserErrors`.

Unit: Count

Dimensi: `TableName`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

`ConsumedChangeDataCaptureUnits`

Jumlah unit tangkapan data perubahan yang digunakan.

Unit: Count

Dimensi: `TableName`, `DelegatedOperation`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`

`ConsumedReadCapacityUnits`

Jumlah unit kapasitas baca yang digunakan selama jangka waktu tertentu untuk kapasitas yang tersedia dan sesuai permintaan, sehingga Anda dapat melacak berapa banyak throughput yang digunakan. Anda dapat mengambil total kapasitas baca yang digunakan untuk tabel dan

semua indeks sekunder globalnya, atau untuk indeks sekunder global tertentu. Untuk informasi selengkapnya, lihat [Mode Kapasitas Baca/Tulis](#).

Dimensi `TableName` mengembalikan `ConsumedReadCapacityUnits` untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat `ConsumedReadCapacityUnits` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

Note

Di Amazon DynamoDB, metrik kapasitas yang dikonsumsi dilaporkan CloudWatch pada interval satu menit sebagai nilai rata-rata. Ini berarti bahwa lonjakan konsumsi kapasitas yang pendek dan intens yang berlangsung hanya satu detik mungkin tidak tercermin secara akurat dalam CloudWatch grafik, yang berpotensi mengarah ke tingkat konsumsi yang lebih rendah untuk menit itu.

Gunakan statistik `Sum` untuk menghitung throughput yang digunakan. Misalnya, dapatkan nilai `Sum` dalam rentang waktu satu menit, dan bagi nilai tersebut dengan jumlah detik dalam satu menit (60) untuk menghitung rata-rata `ConsumedReadCapacityUnits` per detik. Anda dapat membandingkan nilai terhitung dengan nilai throughput yang disediakan yang Anda berikan kepada DynamoDB.

Unit: Count

Dimensi: `TableName`, `GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum` – Jumlah minimum unit kapasitas baca yang digunakan oleh setiap permintaan individu pada tabel atau indeks.
- `Maximum` – Jumlah maksimum unit kapasitas baca yang digunakan oleh setiap permintaan individu pada tabel atau indeks.
- `Average` – Rata-rata kapasitas baca per permintaan yang digunakan.

Note

Nilai `Average` dipengaruhi oleh periode tidak aktif dengan nilai sampel akan menjadi nol.

- `Sum` – Total unit kapasitas baca yang digunakan. Ini adalah statistik yang paling berguna untuk metrik `ConsumedReadCapacityUnits`.

- `SampleCount` – Jumlah permintaan baca ke DynamoDB, meskipun tidak ada kapasitas baca yang digunakan.

Note

Nilai `SampleCount` dipengaruhi oleh periode tidak aktif di mana nilai sampel akan menjadi nol.

`ConsumedWriteCapacityUnits`

Jumlah unit kapasitas tulis yang digunakan selama jangka waktu tertentu untuk kapasitas yang tersedia dan sesuai permintaan, sehingga Anda dapat melacak berapa banyak throughput yang digunakan. Anda dapat mengambil total kapasitas tulis yang digunakan untuk tabel dan semua indeks sekunder globalnya, atau untuk indeks sekunder global tertentu. Untuk informasi selengkapnya, lihat [Mode Kapasitas Baca/Tulis](#).

Dimensi `TableName` mengembalikan `ConsumedWriteCapacityUnits` untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat `ConsumedWriteCapacityUnits` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

Note

Gunakan statistik `Sum` untuk menghitung throughput yang digunakan. Sebagai contoh, dapatkan nilai `Sum` dalam rentang waktu satu menit, dan bagi nilai tersebut dengan jumlah detik dalam satu menit (60) untuk menghitung rata-rata `ConsumedWriteCapacityUnits` per detik (memahami bahwa rata-rata ini tidak menyoroti lonjakan besar tapi singkat dalam aktivitas tulis yang terjadi selama menit itu). Anda dapat membandingkan nilai terhitung dengan nilai throughput yang disediakan yang Anda berikan kepada DynamoDB.

Unit: Count

Dimensi: `TableName`, `GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum` – Jumlah minimum unit kapasitas tulis yang digunakan oleh setiap permintaan individu pada tabel atau indeks.

- **Maximum** – Jumlah maksimum unit kapasitas tulis yang digunakan oleh setiap permintaan individu pada tabel atau indeks.
- **Average** – Rata-rata kapasitas tulis per permintaan yang digunakan.

Note

Nilai **Average** dipengaruhi oleh periode tidak aktif dengan nilai sampel akan menjadi nol.

- **Sum** – Total unit kapasitas tulis yang digunakan. Ini adalah statistik yang paling berguna untuk metrik `ConsumedWriteCapacityUnits`.
- **SampleCount** – Jumlah permintaan tulis ke DynamoDB, meskipun tidak ada kapasitas tulis yang digunakan.

Note

Nilai **SampleCount** dipengaruhi oleh periode tidak aktif dengan nilai sampel akan menjadi nol.

FailedToReplicateRecordCount

Jumlah catatan yang gagal direplikasi DynamoDB ke aliran data Kinesis Anda.

Unit: `Count`

Dimensions: `TableName`, `DelegatedOperation`

Statistik Valid:

- **Sum**

MaxProvisionedTableReadCapacityUtilization

Persentase kapasitas baca yang disediakan dan digunakan oleh tabel atau indeks sekunder global baca tertinggi yang disediakan dari sebuah akun.

Unit: `Percent`

Statistik Valid:

- **Maximum** – Persentase maksimum unit kapasitas baca yang disediakan dan digunakan oleh tabel atau indeks sekunder global baca tertinggi yang disediakan dari sebuah akun.
- **Minimum** – Persentase minimum unit kapasitas baca yang disediakan dan digunakan oleh tabel atau indeks sekunder global baca tertinggi yang disediakan dari sebuah akun.
- **Average** – Persentase rata-rata unit kapasitas baca yang disediakan dan digunakan oleh tabel atau indeks sekunder global baca tertinggi yang disediakan dari akun. Metrik diterbitkan selama interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas baca yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

MaxProvisionedTableWriteCapacityUtilization

Persentase kapasitas tulis yang disediakan dan digunakan oleh tabel atau indeks sekunder global penulisan tertinggi yang disediakan dari sebuah akun.

Unit: Percent

Statistik Valid:


- **Maximum** – Persentase maksimum unit kapasitas tulis yang disediakan dan digunakan oleh tabel atau indeks sekunder global tulis tertinggi yang disediakan dari sebuah akun.
- **Minimum** – Persentase minimum unit kapasitas tulis yang disediakan dan digunakan oleh tabel atau indeks sekunder global tulis tertinggi yang disediakan dari sebuah akun.
- **Average** – Persentase rata-rata unit kapasitas tulis yang disediakan dan digunakan oleh tabel atau indeks sekunder global tulis tertinggi yang disediakan dari akun. Metrik diterbitkan selama interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas tulis yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

OnlineIndexConsumedWriteCapacity

Jumlah unit kapasitas tulis digunakan saat menambahkan indeks sekunder global baru ke tabel. Jika kapasitas tulis indeks terlalu rendah, aktivitas tulis masuk selama fase backfill mungkin akan mengalami throttling. Hal ini dapat meningkatkan waktu yang dibutuhkan untuk membuat indeks. Anda harus memantau statistik ini saat indeks sedang dibuat untuk menentukan apakah kapasitas tulis indeks kurang tersedia.

Anda dapat menyesuaikan kapasitas tulis indeks menggunakan operasi `UpdateTable`, meskipun indeks sedang dibuat.

Metrik `ConsumedWriteCapacityUnits` untuk indeks tidak menyertakan throughput tulis yang digunakan selama pembuatan indeks.

 Note

Metrik ini mungkin tidak ditampilkan jika fase backfill indeks sekunder global yang baru selesai dengan cepat (kurang dari beberapa menit), yang mungkin terjadi jika tabel dasar memiliki sedikit atau tidak ada item untuk di-backfill dalam indeks.

Unit: Count

Dimensi: `TableName`, `GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

`OnlineIndexPercentageProgress`

Persentase penyelesaian ketika indeks sekunder global baru sedang ditambahkan ke tabel. DynamoDB pertama-tama harus mengalokasikan sumber daya untuk indeks baru, lalu melakukan backfill atribut dari tabel ke indeks. Untuk tabel besar, proses ini mungkin memerlukan waktu lama. Anda harus memantau statistik ini untuk melihat kemajuan relatif saat DynamoDB membuat indeks.

Unit: Count

Dimensi: `TableName`, `GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum`
- `Maximum`

- Average
- SampleCount
- Sum

OnlineIndexThrottleEvents

Jumlah peristiwa pelambatan menulis yang terjadi saat menambahkan indeks sekunder global baru ke tabel. Peristiwa ini menunjukkan bahwa pembuatan indeks akan memakan waktu lebih lama, karena aktivitas tulis masuk melampaui throughput tulis yang disediakan pada indeks.

Anda dapat menyesuaikan kapasitas tulis indeks menggunakan operasi `UpdateTable`, meskipun indeks sedang dibuat.

Metrik `WriteThrottleEvents` untuk indeks tidak mencakup peristiwa throttling yang terjadi selama pembuatan indeks.

Unit: Count

Dimensi: `TableName`, `GlobalSecondaryIndexName`

Statistik Valid:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

PendingReplicationCount

Metrik untuk [Versi tabel global 2017.11.29 \(Legacy\)](#) (hanya tabel global). Jumlah pembaruan item yang ditulis ke satu tabel replika, tetapi belum ditulis ke replika lain dalam tabel global.

Unit: Count

Dimensi: `TableName`, `ReceivingRegion`

Statistik Valid:

- Average
- Sample Count
- Sum

ProvisionedReadCapacityUnits

Jumlah unit kapasitas baca yang disediakan untuk tabel atau indeks sekunder global. Dimensi `TableName` mengembalikan `ProvisionedReadCapacityUnits` untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat `ProvisionedReadCapacityUnits` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

Unit:Count

Dimensi: `TableName`, `GlobalSecondaryIndexName`

Statistik Valid:

- Minimum – Pengaturan terendah untuk kapasitas baca yang disediakan. Jika Anda menggunakan `UpdateTable` untuk menambah kapasitas baca, metrik ini menunjukkan nilai terendah `ReadCapacityUnits` yang disediakan selama periode waktu ini.
- Maximum – Pengaturan tertinggi untuk kapasitas baca yang disediakan. Jika Anda menggunakan `UpdateTable` untuk mengurangi kapasitas baca, metrik ini menunjukkan nilai tertinggi `ReadCapacityUnits` yang disediakan selama periode waktu ini.
- Average – Rata-rata kapasitas baca yang disediakan. Metrik `ProvisionedReadCapacityUnits` diterbitkan pada interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas baca yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

ProvisionedWriteCapacityUnits

Jumlah unit kapasitas tulis yang disediakan untuk tabel atau indeks sekunder global.

Dimensi `TableName` mengembalikan `ProvisionedWriteCapacityUnits` untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat `ProvisionedWriteCapacityUnits` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

Unit: Count

Dimensi: `TableName`, `GlobalSecondaryIndexName`

Statistik Valid:

- **Minimum** – Pengaturan terendah untuk kapasitas tulis yang disediakan. Jika Anda menggunakan `UpdateTable` untuk menambah kapasitas tulis, metrik ini menunjukkan nilai terendah `WriteCapacityUnits` yang disediakan selama periode waktu ini.
- **Maximum** – Pengaturan tertinggi untuk kapasitas tulis yang disediakan. Jika Anda menggunakan `UpdateTable` untuk mengurangi kapasitas tulis, metrik ini menunjukkan nilai tertinggi `WriteCapacityUnits` yang disediakan selama periode waktu ini.
- **Average** – Rata-rata kapasitas tulis yang disediakan. Metrik `ProvisionedWriteCapacityUnits` diterbitkan pada interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas tulis yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

`ReadThrottleEvents`

Permintaan untuk DynamoDB yang melampaui unit kapasitas baca yang disediakan untuk tabel atau indeks sekunder global.

Suatu permintaan dapat mengakibatkan beberapa peristiwa. Sebagai contoh, `BatchGetItem` yang membaca 10 item diproses sebagai 10 peristiwa `GetItem`. Untuk setiap peristiwa, `ReadThrottleEvents` bertambah satu jika peristiwa tersebut mengalami throttling. Metrik `ThrottledRequests` untuk seluruh `BatchGetItem` tidak bertambah kecuali jika 10 peristiwa `GetItem` mengalami throttling.

Dimensi `TableName` mengembalikan `ReadThrottleEvents` untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat `ReadThrottleEvents` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

Unit: Count

Dimensi: `TableName`, `GlobalSecondaryIndexName`

Statistik Valid:

- `SampleCount`
- `Sum`

ReplicationLatency

(Metrik ini adalah untuk tabel global DynamoDB.) Waktu yang berlalu antara item yang diperbarui yang muncul di aliran DynamoDB untuk satu tabel replika, dan item tersebut muncul di replika lain di tabel global.

Unit: Milliseconds

Dimensi: TableName, ReceivingRegion

Statistik Valid:

- Average
- Minimum
- Maximum

ReturnedBytes

Jumlah byte yang dikembalikan oleh operasi GetRecords (Amazon DynamoDB Streams) selama periode waktu yang ditentukan.

Unit: Bytes

Dimensi: Operation, StreamLabel, TableName

Statistik Valid:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

ReturnedItemCount

Jumlah item yang dikembalikan oleh operasi Query, Scan atau ExecuteStatement (pilih) selama periode waktu yang ditentukan.

Jumlah item yang dikembalikan belum tentu sama dengan jumlah item yang dievaluasi. Sebagai contoh, anggaplah Anda meminta Scan pada suatu tabel atau indeks yang memiliki 100 item,

tetapi menentukan `FilterExpression` yang mempersempit hasil sehingga hanya 15 item yang dikembalikan. Dalam hal ini, respons dari `Scan` akan berisi `ScanCount` sebanyak 100 dan `Count` sebesar 15 item yang dikembalikan.

Unit: `Count`

Dimensi: `TableName`, `Operation`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

`ReturnedRecordsCount`

Jumlah catatan aliran yang dikembalikan oleh operasi `GetRecords` (Amazon DynamoDB Streams) selama periode waktu yang ditentukan.

Unit: `Count`

Dimensi: `Operation`, `StreamLabel`, `TableName`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

`SuccessfulRequestLatency`

Latensi permintaan yang berhasil untuk DynamoDB atau Amazon DynamoDB Streams selama jangka waktu tertentu. `SuccessfulRequestLatency` dapat memberikan dua jenis informasi:

- Waktu berlalu untuk permintaan yang berhasil (Minimum, Maximum, Sum, atau Average).
- Jumlah permintaan yang berhasil (SampleCount).

SuccessfulRequestLatency hanya mencerminkan aktivitas dalam DynamoDB atau Amazon DynamoDB Streams, dan tidak memperhitungkan latensi jaringan akun atau aktivitas sisi klien.

Unit: Milliseconds

Dimensi: TableName, Operation, StreamLabel

Statistik Valid:

- Minimum
- Maximum
- Average
- SampleCount

SystemErrors

Permintaan untuk DynamoDB atau Amazon DynamoDB Streams yang menghasilkan kode status HTTP 500 selama jangka waktu tertentu. HTTP 500 biasanya menunjukkan kesalahan layanan internal.

Unit: Count

Dimensi: TableName, Operation

Statistik Valid:

- Sum
- SampleCount

TimeToLiveDeletedItemCount

Jumlah item yang dihapus oleh Waktu untuk Beroperasi (TTL) selama jangka waktu yang ditentukan. Metrik ini membantu Anda memantau laju penghapusan TTL di tabel Anda.

Unit: Count

Dimensi: TableName

Statistik Valid:

- Sum

ThrottledPutRecordCount

Jumlah catatan yang dibatasi oleh aliran data Kinesis Anda karena kapasitas Kinesis Data Streams tidak memadai.

Unit: Count

Dimensi: TableName, DelegatedOperation

Statistik Valid:

- Minimum
- Maximum
- Average
- SampleCount

ThrottledRequests

Permintaan untuk DynamoDB yang melampaui batas throughput yang disediakan pada sumber daya (seperti tabel atau indeks).

`ThrottledRequests` bertambah satu jika ada peristiwa dalam permintaan yang melampaui batas throughput yang disediakan. Misalnya, jika Anda memperbarui item dalam tabel dengan indeks sekunder global, ada beberapa peristiwa—penulisan ke tabel, dan penulisan ke setiap indeks. Jika satu atau beberapa peristiwa ini mengalami throttling, maka `ThrottledRequests` bertambah satu.


Note

Dalam permintaan batch (`BatchGetItem` atau `BatchWriteItem`), `ThrottledRequests` bertambah hanya jika setiap permintaan dalam batch mengalami throttling.

Jika setiap permintaan individual dalam batch mengalami throttling, salah satu metrik berikut bertambah:

- `ReadThrottleEvents` – Untuk peristiwa `GetItem` yang mengalami throttling dalam `BatchGetItem`.
- `WriteThrottleEvents` – Untuk peristiwa `PutItem` atau `DeleteItem` yang mengalami throttling dalam `BatchWriteItem`.

Untuk mendapatkan wawasan tentang peristiwa mana yang melakukan throttling terhadap permintaan, bandingkan `ThrottledRequests` dengan `ReadThrottleEvents` dan `WriteThrottleEvents` untuk tabel dan indeksinya.

 Note

Permintaan yang mengalami throttling akan menghasilkan kode status HTTP 400. Semua peristiwa tersebut tercermin dalam metrik `ThrottledRequests`, tapi tidak di metrik `UserErrors`.

Unit: Count

Dimensi: `TableName`, `Operation`

Statistik Valid:

- `Sum`
- `SampleCount`

`TransactionConflict`

Permintaan tingkat item ditolak karena konflik transaksional antara permintaan serentak pada item yang sama. Untuk informasi selengkapnya, lihat [Penanganan Konflik Transaksi di DynamoDB](#).

Unit: Count

Dimensi: `TableName`

Statistik Valid:

- `Sum` – Jumlah permintaan tingkat item yang ditolak karena konflik transaksi.

Note

Jika beberapa permintaan tingkat item dalam suatu panggilan ke `TransactWriteItems` atau `TransactGetItems` ditolak, `Sum` bertambah satu untuk setiap permintaan `Put`, `Update`, `Delete`, atau `Get` tingkat item.

- `SampleCount` – Jumlah permintaan yang ditolak karena konflik transaksi.

Note

Jika beberapa permintaan tingkat item dalam suatu panggilan ke `TransactWriteItems` atau `TransactGetItems` ditolak, `SampleCount` hanya bertambah satu.

- `Min` – Jumlah minimum permintaan tingkat item yang ditolak dalam panggilan ke `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem`, atau `DeleteItem`.
- `Max` – Jumlah maksimum permintaan tingkat item yang ditolak dalam panggilan ke `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem`, atau `DeleteItem`.
- `Average` – Jumlah rata-rata permintaan tingkat item yang ditolak dalam panggilan ke `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem`, atau `DeleteItem`.

UserErrors

Permintaan ke DynamoDB atau Amazon DynamoDB Streams yang menghasilkan kode status HTTP 400 selama jangka waktu tertentu. HTTP 400 biasanya menunjukkan kesalahan di sisi klien, seperti kombinasi parameter yang tidak valid, upaya untuk memperbarui tabel tidak ada, atau tanda tangan permintaan yang salah.

Beberapa contoh pengecualian yang akan mencatat metrik yang terkait dengan `UserErrors` adalah:

- `ResourceNotFoundException`
- `ValidationException`
- `TransactionConflict`

Semua peristiwa tersebut tercermin dalam metrik `UserErrors`, kecuali yang berikut ini:

- `ProvisionedThroughputExceededException`— Lihat `ThrottledRequests` metrik di bagian ini.
- `ConditionalCheckFailedException`— Lihat `ConditionalCheckFailedRequests` metrik di bagian ini.

`UserErrors` mewakili agregat kesalahan HTTP 400 untuk permintaan DynamoDB atau Amazon DynamoDB Streams untuk Wilayah saat ini dan akun saat ini. AWS AWS

Unit: Count

Statistik Valid:

- `Sum`
- `SampleCount`

`WriteThrottleEvents`

Permintaan untuk DynamoDB yang melampaui unit kapasitas tulis yang disediakan untuk tabel atau indeks sekunder global.

Suatu permintaan dapat mengakibatkan beberapa peristiwa. Misalnya, permintaan `PutItem` pada tabel dengan tiga indeks sekunder global akan menghasilkan empat peristiwa—penulisan tabel, dan masing-masing dari tiga penulisan indeks. Untuk setiap peristiwa, metrik `WriteThrottleEvents` bertambah satu jika peristiwa tersebut mengalami throttling. Untuk satu permintaan `PutItem`, jika salah satu peristiwa mengalami throttling, `ThrottledRequests` juga bertambah satu. Untuk `BatchWriteItem`, metrik `ThrottledRequests` untuk seluruh `BatchWriteItem` tidak bertambah kecuali jika semua `PutItem` individual atau peristiwa `DeleteItem` mengalami throttling.

Dimensi `TableName` mengembalikan `WriteThrottleEvents` untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat `WriteThrottleEvents` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

Unit: Count

Dimensi: `TableName`, `GlobalSecondaryIndexName`

Statistik Valid:

- `Sum`
- `SampleCount`

Metrik penggunaan

Metrik penggunaan CloudWatch memungkinkan Anda mengelola penggunaan secara proaktif dengan memvisualisasikan metrik di CloudWatch konsol, membuat dasbor khusus, mendeteksi perubahan aktivitas dengan deteksi CloudWatch anomali, dan mengonfigurasi alarm yang mengingatkan Anda saat penggunaan mendekati ambang batas.

DynamoDB juga mengintegrasikan metrik penggunaan ini dengan Kuota Layanan. Anda dapat menggunakan CloudWatch untuk mengelola penggunaan akun Anda atas kuota layanan Anda. Untuk informasi selengkapnya, lihat [Memvisualisasikan kuota layanan dan mengatur alarm](#)

Daftar Metrik Penggunaan yang Tersedia

- [AccountProvisionedWriteCapacityUnits](#)
- [AccountProvisionedReadCapacityUnits](#)
- [TableCount](#)

AccountProvisionedWriteCapacityUnits

Jumlah unit kapasitas tulis yang disediakan untuk semua tabel dan indeks sekunder global pada account.

Unit: Count

Statistik Valid:

- **Minimum** – Jumlah terendah unit kapasitas tulis yang disediakan selama jangka waktu tertentu.
- **Maximum** – Jumlah tertinggi unit kapasitas tulis yang disediakan selama jangka waktu tertentu.
- **Average** – Jumlah rata-rata unit kapasitas tulis yang disediakan pada akun selama jangka waktu tertentu.

Metrik ini diterbitkan pada interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas tulis yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

AccountProvisionedReadCapacityUnits

Jumlah unit kapasitas baca yang disediakan untuk semua tabel dan indeks sekunder global pada account.

Unit: Count

Statistik Valid:

- **Minimum** – Jumlah terendah unit kapasitas baca yang disediakan selama jangka waktu tertentu.
- **Maximum** – Jumlah tertinggi unit kapasitas baca yang disediakan selama jangka waktu tertentu.
- **Average** – Jumlah rata-rata unit kapasitas baca yang disediakan pada akun selama jangka waktu tertentu.

Metrik ini diterbitkan pada interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas baca yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

TableCount

Jumlah tabel aktif pada akun.

Unit: Count

Statistik Valid:

- **Minimum** – Jumlah tabel terendah selama periode waktu tertentu.
- **Maximum** – Jumlah tabel tertinggi selama periode waktu tertentu.
- **Average** – Jumlah tabel rata-rata selama periode waktu tertentu.

Memahami metrik dan dimensi DynamoDB

Metrik untuk DynamoDB dikualifikasikan berdasarkan nilai akun, nama tabel, nama indeks sekunder global, atau operasi. Anda dapat menggunakan CloudWatch konsol untuk mengambil data DynamoDB sepanjang salah satu dimensi dalam tabel di bawah ini.

Daftar Dimensi yang Tersedia

- [DelegatedOperation](#)
- [GlobalSecondaryIndexName](#)
- [Operation](#)
- [OperationType](#)
- [Verb](#)
- [ReceivingRegion](#)
- [StreamLabel](#)

- [TableName](#)

DelegatedOperation

Dimensi ini membatasi data pada operasi yang dilakukan DynamoDB atas nama Anda. Operasi berikut ditangkap:

- Ubah penangkapan data untuk Kinesis Data Streams.

GlobalSecondaryIndexName

Dimensi ini membatasi data untuk indeks sekunder global pada tabel. Jika menentukan `GlobalSecondaryIndexName`, Anda juga harus menentukan `TableName`.

Operation

Dimensi ini membatasi data ke salah satu operasi DynamoDB berikut:

- `PutItem`
- `DeleteItem`
- `UpdateItem`
- `GetItem`
- `BatchGetItem`
- `Scan`
- `Query`
- `BatchWriteItem`
- `TransactWriteItems`
- `TransactGetItems`
- `ExecuteTransaction`
- `BatchExecuteStatement`
- `ExecuteStatement`

Selain itu, Anda dapat membatasi data untuk operasi Amazon DynamoDB Streams berikut:

- `GetRecords`

OperationType

Dimensi ini membatasi data ke salah satu jenis operasi berikut:

- Read
- Write

Dimensi ini diteruskan untuk permintaan `ExecuteTransaction` dan `BatchExecuteStatement`.

Verb

Dimensi ini membatasi data ke salah satu kata kerja DynamoDB PartiQL berikut:

- Sisipkan: `PartiQLInsert`
- Pilih: `PartiQLSelect`
- Perbarui: `PartiQLUpdate`
- Hapus: `PartiQLDelete`

Dimensi ini diteruskan untuk operasi `ExecuteStatement`.

ReceivingRegion

Dimensi ini membatasi data ke AWS wilayah tertentu. Ini digunakan dengan metrik yang berasal dari tabel replika dalam tabel global DynamoDB.

StreamLabel


Dimensi ini membatasi data untuk label aliran tertentu. Ini digunakan dengan metrik yang berasal dari operasi `GetRecords` Amazon DynamoDB Streams.

TableName

Dimensi ini membatasi data ke tabel tertentu. Nilai ini dapat berupa nama tabel apa pun di wilayah saat ini dan AWS akun saat ini.

Bagaimana cara menggunakan metrik DynamoDB?

Metrik yang dilaporkan oleh DynamoDB memberikan informasi yang dapat Anda analisis dengan berbagai cara. Daftar berikut menunjukkan beberapa penggunaan umum untuk metrik. Daftar ini berisi saran agar Anda dapat memulai, bukan daftar komprehensif.

Bagaimana saya dapat melakukannya?	Metrik terkait
<p>How can I monitor the rate of TTL deletions on my table?</p>	<p>Anda dapat memantau <code>TimeToLiveDeletedItemCount</code> selama periode waktu yang ditentukan, untuk melacak tingkat penghapusan TTL di tabel Anda. Untuk contoh aplikasi tanpa server yang menggunakan <code>TimeToLiveDeletedItemCount</code> metrik, lihat Mengarsipkan item secara otomatis ke S3 menggunakan DynamoDB time to live (TTL) with dan Amazon Data Firehose. AWS Lambda</p>
<p>How can I determine how much of my provisioned throughput is being used?</p>	<p>Anda dapat memantau <code>ConsumedReadCapacityUnits</code> atau <code>ConsumedWriteCapacityUnits</code> selama periode waktu yang ditentukan, untuk melacak jumlah throughput yang disediakan yang sedang digunakan.</p>
<p>How can I determine which requests exceed the provisioned throughput quotas of a table?</p>	<p><code>ThrottledRequests</code> bertambah satu jika ada peristiwa dalam permintaan yang melampaui kuota throughput yang disediakan. Untuk mendapatkan wawasan tentang peristiwa mana yang melakukan throttling terhadap permintaan, bandingkan <code>ThrottledRequests</code> dengan metrik <code>ReadThrottleEvents</code> dan <code>WriteThrottleEvents</code> untuk tabel dan indeksinya.</p>
<p>How can I determine if any system errors occurred?</p>	<p>Anda dapat memantau <code>SystemErrors</code> untuk menentukan apakah setiap permintaan menghasilkan kode HTTP 500 (kesalahan server). Biasanya, metrik ini sama dengan nol. Jika tidak, Anda mungkin harus menyelidikinya.</p> <div data-bbox="594 1482 1507 1793" style="border: 1px solid #00a0e3; border-radius: 10px; padding: 10px;"><p> Note</p><p>Anda mungkin mengalami kesalahan server internal saat bekerja dengan item. Ini diduga terjadi selama masa aktif tabel. Setiap permintaan yang gagal dapat segera dicoba kembali.</p></div>

Memahami dan menganalisis waktu respons DynamoDB

Saat menganalisis latensi, praktik terbaiknya adalah memeriksa rata-ratanya. Lonjakan latensi sesekali tidak perlu dikhawatirkan. Namun, jika latensi rata-rata tinggi maka mungkin ada masalah mendasar yang menjadi penyebabnya.

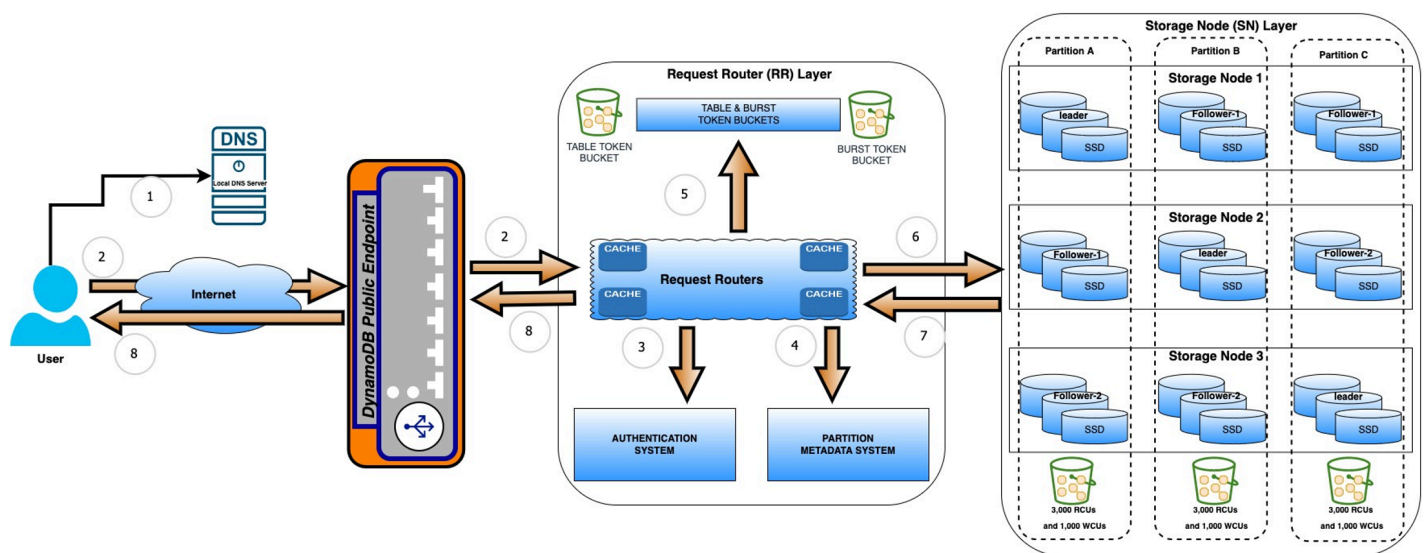
Ada dua kategori latensi, latensi API dan latensi sisi layanan. Latensi API DynamoDB diukur dari langkah 1 hingga 11 dalam proses di bawah ini. Latensi sisi layanan diukur dari saat API mencapai Router Permintaan (langkah 4) hingga waktu yang dibutuhkan RR untuk mengirimkan hasil kembali ke pengguna aplikasi (langkah 11). Anda dapat menganalisis latensi sisi layanan dengan CloudWatch metrik `SuccessfulRequestLatency` Amazon.

Ketika aplikasi membuat panggilan API DynamoDB ke DynamoDB, seperti mengeluarkan operasi `GetItem` ke tabel DynamoDB, langkah-langkah berikut terjadi.

1. Aplikasi ini menyelesaikan titik akhir publik DynamoDB menggunakan server DNS lokal.
2. Aplikasi ini terhubung ke alamat IP yang diselesaikan pada langkah pertama, dan membuat panggilan API.
3. Titik akhir publik DynamoDB mengambil permintaan dan meneruskannya ke komponen yang disebut Router Permintaan (RR).
4. Setelah permintaan API mencapai Router Permintaan (RR), RR melakukan autentikasi dan otorisasi panggilan API. RR juga melakukan pemeriksaan throttling pada tahap ini.
5. Setelah menyelesaikan semua pemeriksaan, Router Permintaan (RR) membuat hash dari nilai kunci partisi yang diperoleh dari permintaan API. Berdasarkan nilai hash, Router Permintaan (RR) menemukan detail informasi partisi (simpul penyimpanan).
6. Simpul Penyimpanan mewakili server tempat data tabel pelanggan disimpan. Sebuah partisi tunggal (jangan bingung dengan kunci primer atau kunci partisi), terdiri dari satu set 3 Simpul Penyimpanan. Dari ketiga Simpul Penyimpanan ini, satu simpul berfungsi sebagai pemimpin untuk partisi itu dan dua simpul lainnya berfungsi sebagai pengikut.
7. Jika panggilan API adalah permintaan tulis atau permintaan baca sangat konsisten, maka Router Permintaan (RR) akan menemukan simpul pemimpin untuk partisi itu dan meneruskan permintaan API ke simpul tertentu. Dalam kasus permintaan bacaan akhir konsisten, Router Permintaan (RR) meneruskan permintaan ke simpul pemimpin atau simpul pengikut mana pun untuk partisi itu secara acak.
8. Dalam keadaan normal, Router Permintaan mencapai Simpul Penyimpanan pada upaya pertama. Jika upaya ini gagal, RR melakukan beberapa percobaan ulang untuk mencapai

simpul penyimpanan. Saat menghubungkan ke simpul penyimpanan, RR selalu memberikan waktu yang cukup untuk upaya yang ada dan kemudian mencoba menghubungkan ke SN yang berbeda. Ini adalah percobaan ulang layanan mikro internal, dan bukan percobaan ulang SDK yang dapat dikonfigurasi.

9. Pada tahap ini, permintaan API mencapai lapisan Simpul Penyimpanan (SN) dan SN mulai memprosesnya, membaca atau menulis data tergantung pada panggilan API.
10. Setelah berhasil memproses permintaan API, Simpul Penyimpanan (SN) mengembalikan hasil atau kode respons ke RR yang berasal dari permintaan.
11. Akhirnya, Router Permintaan meneruskan hasil ke aplikasi pelanggan.



Note

- Untuk sebagian besar operasi atom, seperti `GetItem` dan `PutItem`, Anda dapat mengharapkan latensi rata-rata dalam satu digit milidetik. Latensi untuk operasi non-atom seperti `Query` dan `Scan` tergantung pada banyak faktor, termasuk ukuran set hasil dan kompleksitas filter dan ketentuan kueri.
- DynamoDB tidak mengukur jumlah waktu yang dibutuhkan aplikasi untuk terhubung dengan titik akhir publik DynamoDB, atau jumlah waktu yang dibutuhkan aplikasi untuk mengunduh hasil dari titik akhir publik.

Membuat CloudWatch alarm untuk memantau DynamoDB

Anda dapat membuat CloudWatch alarm yang mengirimkan pesan Amazon SNS saat alarm berubah status. Alarm mengawasi satu metrik selama jangka waktu yang Anda tentukan, dan melakukan satu atau beberapa tindakan berdasarkan nilai metrik relatif terhadap ambang batas tertentu selama jangka waktu tertentu. Tindakan ini adalah notifikasi yang dikirim ke topik Amazon SNS atau kebijakan Penskalaan Otomatis. Alarm memanggil tindakan untuk perubahan status berkelanjutan saja. CloudWatch alarm tidak memanggil tindakan hanya karena mereka berada dalam keadaan tertentu; negara harus telah berubah dan dipertahankan untuk sejumlah periode tertentu.

Note

Anda harus menentukan semua dimensi yang diperlukan saat membuat CloudWatch alarm Anda, karena tidak CloudWatch akan menggabungkan metrik untuk dimensi yang hilang. Membuat CloudWatch alarm dengan dimensi yang hilang tidak akan menghasilkan kesalahan, saat membuat alarm.

Untuk daftar metrik yang didukung dan dimensi yang diperlukannya di DynamoDB, lihat [Melihat metrik dan dimensi](#).

Bagaimana saya bisa mendapatkan pemberitahuan sebelum menggunakan seluruh kapasitas baca saya?

1. Buat topik Amazon SNS, `arn:aws:sns:us-east-1:123456789012:capacity-alarm`.

Untuk informasi selengkapnya, lihat [Menyiapkan Amazon Simple Notification Service](#).

2. Buat alarm. Dalam contoh ini, kita mengasumsikan kapasitas yang disediakan sebanyak lima unit kapasitas baca.

```
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. Uji alarm.

```
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
```

Note

Alarm diaktifkan setiap kali kapasitas baca digunakan setidaknya 4 unit per detik (80% dari kapasitas baca yang disediakan sebanyak 5) selama 1 menit (60 detik). Jadi, threshold adalah 240 unit kapasitas baca (4 unit/detik * 60 detik). Setiap kali kapasitas baca diperbarui, Anda harus memperbarui penghitungan alarm dengan tepat. Anda dapat menghindari proses ini dengan membuat alarm melalui konsol DynamoDB. Dengan cara ini, alarm diperbarui secara otomatis untuk Anda.

Bagaimana saya bisa mendapatkan pemberitahuan jika ada permintaan yang melampaui kuota throughput yang disediakan pada tabel?

1. Buat topik Amazon SNS, `arn:aws:sns:us-east-1:123456789012:requests-exceeding-throughput`.

Untuk informasi selengkapnya, lihat [Menyiapkan Amazon Simple Notification Service](#).

2. Buat alarm.

```
aws cloudwatch put-metric-alarm \  
--alarm-name RequestsExceedingThroughputAlarm\  
--alarm-description "Alarm when my requests are exceeding provisioned  
throughput quotas of a table" \  
--namespace AWS/DynamoDB \  
--metric-name ThrottledRequests \  
--dimensions Name=TableName,Value=myTable  
Name=Operation,Value=aDynamoDBOperation \  
--statistic Sum \  

```

```
--threshold 0 \  
--comparison-operator GreaterThanThreshold \  
--period 300 \  
--unit Count \  
--evaluation-periods 1 \  
--treat-missing-data notBreaching \  
--alarm-actions arn:aws:sns:us-east-1:123456789012:requests-exceeding-  
throughput
```

3. Uji alarm.

```
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
```

Bagaimana saya dapat diberi tahu jika terjadi kesalahan sistem?

1. Buat topik Amazon SNS, `arn:aws:sns:us-east-1:123456789012:notify-on-system-errors`.

Untuk informasi selengkapnya, lihat [Menyiapkan Amazon Simple Notification Service](#).

2. Buat alarm.

```
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. Uji alarm.

```
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
```

Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail

DynamoDB terintegrasi AWS CloudTrail dengan, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau AWS layanan di DynamoDB. CloudTrail menangkap semua panggilan API untuk DynamoDB sebagai peristiwa. Panggilan yang ditangkap mencakup panggilan dari konsol DynamoDB dan panggilan kode ke operasi API DynamoDB, menggunakan PartiQL dan API klasik. Jika Anda membuat jejak, Anda dapat mengaktifkan pengiriman acara secara terus menerus ke bucket Amazon S3, termasuk CloudTrail peristiwa untuk DynamoDB. Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru di CloudTrail konsol dalam Riwayat acara. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk DynamoDB, alamat IP dari mana permintaan dibuat, siapa yang membuat permintaan, kapan dibuat, dan detail tambahan.

Untuk pemantauan dan peringatan yang kuat, Anda juga dapat mengintegrasikan CloudTrail peristiwa dengan [Amazon CloudWatch Logs](#). [Untuk meningkatkan analisis aktivitas layanan DynamoDB dan mengidentifikasi perubahan aktivitas untuk AWS akun, Anda dapat melakukan AWS CloudTrail kueri log menggunakan Amazon Athena](#). Misalnya, Anda dapat menggunakan kueri untuk mengidentifikasi tren dan mengisolasi aktivitas lebih lanjut berdasarkan atribut seperti alamat IP sumber atau pengguna.

Untuk mempelajari selengkapnya CloudTrail, termasuk cara mengonfigurasi dan mengaktifkannya, lihat [Panduan AWS CloudTrail Pengguna](#).

Topik

- [Informasi DynamoDB di CloudTrail](#)
- [Memahami entri file log DynamoDB](#)

Informasi DynamoDB di CloudTrail

CloudTrail diaktifkan di AWS akun Anda saat Anda membuat akun. Ketika aktivitas peristiwa yang didukung terjadi di DynamoDB, aktivitas tersebut direkam dalam CloudTrail suatu peristiwa bersama dengan peristiwa layanan AWS lainnya dalam riwayat Acara. Anda dapat melihat, mencari, dan mengunduh acara terbaru di AWS akun Anda. Untuk informasi selengkapnya, lihat [Melihat peristiwa dengan riwayat CloudTrail acara](#).

Untuk catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk acara untuk DynamoDB, buat jejak. Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon S3. Secara default, saat Anda membuat jejak di konsol, jejak tersebut berlaku untuk semua AWS Wilayah. Jejak mencatat peristiwa dari semua Wilayah di AWS partisi dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi AWS layanan lain untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi selengkapnya, lihat berikut:

- [Gambaran umum untuk membuat jejak](#)
- [CloudTrail layanan dan integrasi yang didukung](#)
- [Mengonfigurasi notifikasi Amazon SNS untuk CloudTrail](#)
- [Menerima file CloudTrail log dari beberapa wilayah](#) dan [Menerima file CloudTrail log dari beberapa akun](#)

Kontrol peristiwa pesawat di CloudTrail

Tindakan API berikut dicatat secara default sebagai peristiwa dalam CloudTrail file:

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](#)

Peristiwa bidang data DynamoDB di CloudTrail

Untuk mengaktifkan pencatatan tindakan API berikut dalam CloudTrail file, Anda harus mengaktifkan pencatatan aktivitas API bidang data di CloudTrail. Lihat [Pencatatan log peristiwa data untuk jejak](#) untuk informasi selengkapnya.

Peristiwa bidang data dapat difilter berdasarkan jenis sumber daya, untuk kontrol terperinci atas panggilan DynamoDB API yang ingin Anda log dan bayar secara selektif. CloudTrail Misalnya, dengan menentukan `AWS::DynamoDB::Stream` sebagai jenis sumber daya, Anda hanya dapat mencatat panggilan ke API aliran DynamoDB. Untuk tabel dengan aliran yang diaktifkan, bidang sumber daya dalam peristiwa bidang data berisi `AWS::DynamoDB::Stream` dan `AWS::DynamoDB::Table`. Jika Anda menentukan `AWS::DynamoDB::Table` sebagai jenis sumber daya, tabel DynamoDB dan peristiwa aliran DynamoDB akan dicatat secara default. Anda dapat menambahkan [filter](#) tambahan untuk mengecualikan peristiwa aliran, jika tidak ingin peristiwa aliran

dicatat. Untuk informasi selengkapnya, lihat [DataResource](#) di [referensi CloudTrail AWS CloudTrail API](#).

Amazon DynamoDB

- [BatchExecuteStatement](#)
- [BatchGetItem](#)
- [BatchWriteItem](#)
- [DeleteItem](#)
- [ExecuteStatement](#)
- [ExecuteTransaction](#)
- [GetItem](#)
- [PutItem](#)
- [Query](#)
- [Scan](#)
- [TransactGetItems](#)
- [TransactWriteItems](#)
- [UpdateItem](#)

Note

DynamoDB Time to Live tindakan pesawat data tidak dicatat oleh CloudTrail

DynamoDB Streams

- [GetRecords](#)
- [GetShardIterator](#)

Memahami entri file log DynamoDB

Trail adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail file log berisi satu atau lebih entri log. Sebuah peristiwa mewakili

permintaan tunggal dari sumber apa pun dan mencakup informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya.

Setiap entri kejadian atau log berisi informasi tentang siapa yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan hal berikut:

- Baik permintaan tersebut dibuat dengan kredensial pengguna root atau pengguna.
- Apakah permintaan dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan itu dibuat oleh AWS layanan lain.

Note

Nilai atribut non kunci akan disunting dalam CloudTrail log tindakan menggunakan API PartiQL, dan tidak akan muncul di log tindakan menggunakan API klasik.

Untuk informasi selengkapnya, lihat elemen [CloudTrail UserIdentity](#).

Contoh berikut menunjukkan CloudTrail log dari jenis acara ini:

Amazon DynamoDB

- [UpdateTable](#)
- [DeleteTable](#)
- [CreateCluster](#)
- [PutItem \(Berhasil\)](#)
- [UpdateItem \(Gagal\)](#)
- [TransactWriteItems \(Berhasil\)](#)
- [TransactWriteItems \(Dengan TransactionCanceledException\)](#)
- [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": "AALNP0J2L244N5015PKISJ1KUFVV4KQNS05AEMVJF66Q9ASUAAJG",
"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": "4KBNVRGD25RG1KE09UT4V3FQDJVV4KQNS05AEMVJF66Q9ASUAAJG",
```

```

    "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 boto3/1.13.40",
      "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 (Berhasil)

```

{
  "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": "4KBNVRGD25RG1KE09UT4V3FQDJVV4KQNS05AEMVJF66Q9ASUAAJG",
"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 (Gagal)

```
{
  "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": "4KBNVRGD25RG1KE09UT4V3FQDJVV4KQNS05AEMVJF66Q9ASUAAJG",
    "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 (Berhasil)

```
{
  "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": "45EN320M6TQSMV2MI6504L5TNFVV4KQNS05AEMVJF66Q9ASUAAJG",
"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 (Dengan TransactionCanceledException)

```

{
  "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 boto3/1.12.83",
"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": "A0GTQEKLB9VD8E05REA5A3E1VVV4KQNS05AEMVJF66Q9ASUAAJG",

```

```
"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 boto3/1.20.7",
    "requestParameters": {
        "statement": "SELECT * FROM Music WHERE Artist = 'No One You Know' AND
SongTitle = 'Call Me Today' AND nonKeyAttr = ***(Redacted)"
    },
    "responseElements": null,
    "requestID": "V7G2KCSFLP830RB7MMFG6RIAD3VV4KQNS05AEMVJF66Q9ASUAAJG",
    "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 boto3/1.20.7",
  "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": "23PE7ED291UD65P9SMS6TISNVBVV4KQNS05AEMVJF66Q9ASUAAJG",
  "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 boto3/1.20.50",

```

```

    "requestParameters": {
      "shardIterator": "arn:aws:dynamodb:us-west-2:123456789012:table/
Music/stream/2021-04-15T04:02:47.428|1|AAAAAAAAAAAH7HF3xwDQHBrvk2UBZ1PKh8bX3F
+JeH0rFwHCE7dz4VGv1ZoJ5bMxQwkmerA3wzCTL+zSseGLdSXNJP14EwrjLNvDNoZeRSJ/
n6xc3I4NYOptR4zR8d7VrjMAD6h5nR12NtxGIgJ/
dVsUpLuWsHyCW3PPbKsMLJSruVRWoitRhSd3S6s1EWEPEB0bDC7+
+ISH5mXrCH0nvyezQK1qNshTSPZ5jWwqRj2VNSXCMTGXv9P01/
U0bp0UI2cuRTchgUpPSe3ur2sQrRj3K1bmIyCz7P
+H3CY1ugafi8fQ5kipDSkESkIWS605ejzibWKg/3izms1eVIm/
zLFdEeihCYJ7G8fpHUSLX5JAK3ab68aUXGSFEZL0NntgNIhQkcMo00/
mJ1aIgkEdBUyqvZ01vtKUBH5YonIrZqSUhv8Coc+mh24v0g1YI+SPIX1r
+Ln154BG6AjrmaScjHACVXoPDxPsXSJXC4c9HjoC3YSskCPV7uWi0f65/
n7JAT3cskcX2ISaLHwYzJPaMBSftxOgeRLm3BnisL32nT8uTj2gF/
PUrEjdyoqTX7EerQpcaekXm0gay5Kh8n4T2uPdM83f356vRpar/
DDp8pLFD0ddb6Yvz7zU2zGdAvTod3IScC1GpTqcjRxaMh1BVZy1TnI9Cs
+7fXMdUF6xYScjR2725icFBNLojSFVDmsfHabXaCEpmeuXZsLbp5CjcPAHa66R8mQ5tSoFjrz0EzeB4uconEXAMPLE=="
    },
    "responseElements": null,
    "requestID": "1M0U1Q80P4LDPT7A7N1A758N2VVV4KQNS05AEMVJF66Q9EXAMPLE",
    "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"
  }
}

```

Pencatatan log dan pemantauan di DynamoDB Accelerator

Pemantauan merupakan bagian penting dalam menjaga keandalan, ketersediaan, dan kinerja Amazon DynamoDB Accelerator (DAX) dan solusi Anda. AWS Anda harus mengumpulkan data

pemantauan dari semua bagian AWS solusi Anda sehingga Anda dapat lebih mudah men-debug kegagalan multi-titik, jika terjadi.

Untuk informasi selengkapnya tentang pencatatan log dan pemantauan di DAX, lihat [Pemantauan DAX](#).

Menganalisis akses data menggunakan wawasan CloudWatch kontributor untuk DynamoDB

Amazon CloudWatch Contributor Insights untuk Amazon DynamoDB adalah alat diagnostik untuk mengidentifikasi kunci yang paling sering diakses dan dibatasi dalam tabel atau indeks Anda secara sekilas. Alat ini menggunakan [wawasan CloudWatch kontributor](#).

Dengan mengaktifkan CloudWatch Contributor Insights for DynamoDB pada tabel atau indeks sekunder global, Anda dapat melihat item yang paling banyak diakses dan dibatasi dalam sumber daya tersebut.

Note

CloudWatch biaya berlaku untuk Contributor Insights untuk DynamoDB. Untuk informasi selengkapnya tentang harga, lihat [CloudWatch harga Amazon](#).

Topik

- [CloudWatch wawasan kontributor untuk DynamoDB: Cara kerjanya](#)
- [Memulai dengan wawasan CloudWatch kontributor untuk DynamoDB](#)
- [Menggunakan IAM dengan wawasan CloudWatch kontributor untuk DynamoDB](#)

CloudWatch wawasan kontributor untuk DynamoDB: Cara kerjanya

Amazon DynamoDB terintegrasi dengan [CloudWatch Amazon Contributor Insights](#) untuk memberikan informasi tentang item yang paling banyak diakses dan dibatasi dalam tabel atau indeks sekunder global. DynamoDB memberikan informasi ini kepada Anda CloudWatch melalui aturan Contributor [Insights, laporan](#), dan grafik data laporan.

Untuk informasi selengkapnya tentang Wawasan CloudWatch Kontributor, lihat [Menggunakan Wawasan Kontributor untuk Menganalisis Data Kardinalitas Tinggi](#) di Panduan Pengguna Amazon CloudWatch.

Bagian berikut menjelaskan konsep inti dan perilaku CloudWatch Contributor Insights untuk DynamoDB.

Topik

- [CloudWatch wawasan kontributor untuk aturan DynamoDB](#)
- [Memahami wawasan CloudWatch kontributor untuk grafik DynamoDB](#)
- [Interaksi dengan fitur DynamoDB lain](#)
- [CloudWatch wawasan kontributor untuk penagihan DynamoDB](#)

CloudWatch wawasan kontributor untuk aturan DynamoDB

[Saat Anda mengaktifkan CloudWatch Contributor Insights for DynamoDB pada tabel atau indeks sekunder global, DynamoDB akan membuat aturan berikut atas nama Anda:](#)

- Item yang paling sering diakses (kunci partisi) — Mengidentifikasi kunci partisi dari item yang paling sering diakses di tabel Anda atau indeks sekunder global.

CloudWatch format nama aturan: `DynamoDBContributorInsights-PKC-[resource_name]-[creationtimestamp]`

- Item yang paling sering mengalami throttling (kunci partisi) — Mengidentifikasi kunci partisi dari item yang paling sering mengalami throttling di tabel atau indeks sekunder global Anda.

CloudWatch format nama aturan: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

Note

Saat mengaktifkan Contributor Insights di tabel DynamoDB, Anda masih tunduk pada batas aturan Contributor Insights. Untuk informasi selengkapnya, lihat [CloudWatch service quotas](#).

Jika tabel atau indeks sekunder global Anda memiliki kunci urutan, DynamoDB juga membuat aturan berikut khusus untuk kunci urutan:

- Kunci yang paling sering diakses (kunci partisi dan urutan) — Mengidentifikasi kunci partisi dan urutan dari item yang paling sering diakses di tabel atau indeks sekunder global Anda.

CloudWatch format nama aturan: `DynamoDBContributorInsights-SKC-[resource_name]-[creationtimestamp]`

- Kunci yang paling sering mengalami throttling (kunci partisi dan urutan) — Mengidentifikasi kunci partisi dan urutan dari item yang paling sering mengalami throttling di tabel atau indeks sekunder global Anda.

CloudWatch format nama aturan: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

Note

- Anda tidak dapat menggunakan CloudWatch konsol atau API untuk secara langsung mengubah atau menghapus aturan yang dibuat oleh CloudWatch Contributor Insights for DynamoDB. Menonaktifkan CloudWatch Contributor Insights untuk DynamoDB pada tabel atau indeks sekunder global secara otomatis menghapus aturan yang dibuat untuk tabel tersebut atau indeks sekunder global.
- Bila Anda menggunakan [GetInsightRuleReport](#) operasi dengan aturan CloudWatch Contributor Insights yang dibuat oleh DynamoDB, hanya dan mengembalikan statistik yang berguna. `MaxContributorValue` Maximum Statistik lain dalam daftar ini tidak mengembalikan nilai yang berarti.
- CloudWatch Contributor Insights untuk DynamoDB memiliki batas 25 kontributor. Meminta lebih dari 25 kontributor akan mengembalikan kesalahan.

[Anda dapat membuat CloudWatch Alarm menggunakan CloudWatch Contributor Insights for DynamoDB rules](#). Hal ini memungkinkan Anda menerima pemberitahuan jika ada item yang melebihi atau mencapai ambang batas tertentu untuk `ConsumedThroughputUnits` atau `ThrottleCount`. Untuk informasi selengkapnya, lihat [Mengatur Alarm pada Data Metrik Wawasan Kontributor](#).

Memahami wawasan CloudWatch kontributor untuk grafik DynamoDB

CloudWatch Contributor Insights for DynamoDB menampilkan dua jenis grafik pada DynamoDB dan konsol: Item Paling Banyak Diakses dan Item Paling Dibatasi. CloudWatch

Item paling sering diakses

Gunakan grafik ini untuk mengidentifikasi item yang paling sering diakses dalam tabel atau indeks sekunder global. Grafik ini menampilkan `ConsumedThroughputUnits` pada sumbu y dan waktu pada sumbu x. Setiap kunci N teratas ditampilkan dalam warnanya sendiri, dengan keterangan ditampilkan di bawah sumbu x.

DynamoDB mengukur frekuensi akses utama menggunakan `ConsumedThroughputUnits`, yang mengukur gabungan lalu lintas baca dan tulis. `ConsumedThroughputUnits` didefinisikan sebagai berikut:

- Disediakan — (3 x unit kapasitas tulis yang disediakan) + unit kapasitas tulis yang digunakan
- Sesuai permintaan — (3 x unit permintaan tulis) + unit permintaan baca

Pada konsol DynamoDB, setiap titik data dalam grafik mewakili `ConsumedThroughputUnits` maksimum selama periode 1 menit. Misalnya, nilai grafik 180.000 `ConsumedThroughputUnits` menunjukkan bahwa item diakses terus-menerus pada throughput maksimum per item sebanyak 1.000 permintaan unit tulis atau 3.000 unit permintaan baca untuk rentang 60 detik dalam periode 1 menit tersebut (3.000 x 60 detik). Dengan kata lain, nilai bergrafik mewakili menit lalu lintas tertinggi dalam setiap periode 1 menit. Anda dapat mengubah perincian waktu `ConsumedThroughputUnits` metrik (misalnya, untuk melihat metrik 5 menit, bukan 1 menit) di konsol. CloudWatch

Jika Anda melihat beberapa baris berklaster rapat tanpa outlier jelas, ini menunjukkan bahwa beban kerja Anda relatif seimbang di item pada periode waktu tertentu. Jika Anda melihat titik terisolasi dalam grafik, bukan garis terhubung, ini menunjukkan item yang sering diakses hanya untuk jangka waktu singkat.

Jika tabel atau indeks sekunder global Anda memiliki kunci urutan, DynamoDB membuat dua grafik: satu untuk kunci partisi yang paling sering diakses dan satu untuk pasangan kunci partisi + urutan yang paling sering diakses. Anda dapat melihat lalu lintas pada tingkat kunci partisi di grafik khusus kunci partisi. Anda dapat melihat lalu lintas pada tingkat item di grafik kunci partisi + urutan.

Item yang paling sering mengalami throttling

Gunakan grafik ini untuk mengidentifikasi item yang paling sering mengalami throttling dalam tabel atau indeks sekunder global. Grafik ini menampilkan `ThrottleCount` pada sumbu y dan waktu pada sumbu x. Setiap kunci N teratas ditampilkan dalam warnanya sendiri, dengan keterangan ditampilkan di bawah sumbu x.

DynamoDB mengukur frekuensi throttling menggunakan `ThrottleCount`, yang merupakan jumlah kesalahan `ProvisionedThroughputExceededException`, `ThrottlingException`, dan `RequestLimitExceeded`.

Penulisan throttling yang disebabkan oleh kapasitas tulis yang tidak memadai untuk indeks sekunder global tidak diukur. Anda dapat menggunakan grafik Item Paling Sering Diakses pada indeks sekunder global untuk mengidentifikasi pola akses tidak seimbang yang dapat menyebabkan throttling tulis. Untuk informasi selengkapnya, lihat [Pertimbangan Throughput yang Disediakan untuk Indeks Sekunder Global](#).

Pada konsol DynamoDB, setiap titik data dalam grafik menunjukkan jumlah peristiwa throttling selama periode 1 menit.

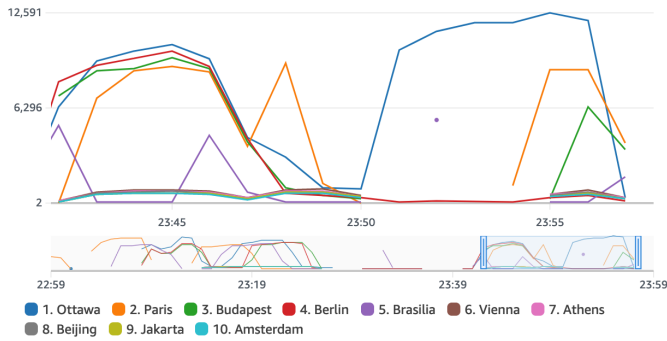
Jika Anda tidak melihat data dalam grafik ini, artinya permintaan Anda tidak mengalami throttling. Jika Anda melihat titik terisolasi dalam grafik, bukan garis yang terhubung, ini menunjukkan bahwa item sering mengalami throttling untuk jangka waktu singkat.

Jika tabel atau indeks sekunder global Anda memiliki kunci urutan, DynamoDB membuat dua grafik: satu untuk kunci partisi yang paling sering mengalami throttling dan satu untuk pasangan kunci partisi + urutan yang paling sering mengalami throttling. Anda dapat melihat jumlah throttling pada tingkat kunci partisi pada grafik khusus kunci partisi, dan jumlah throttling pada tingkat item pada grafik kunci partisi + urutan

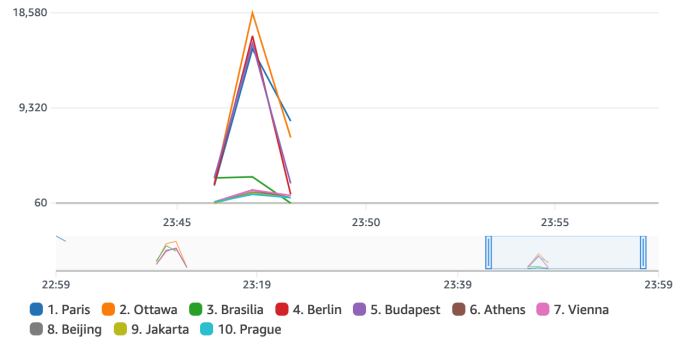
Contoh laporan

Berikut ini adalah contoh laporan yang dihasilkan untuk tabel dengan kunci partisi dan kunci urutan.

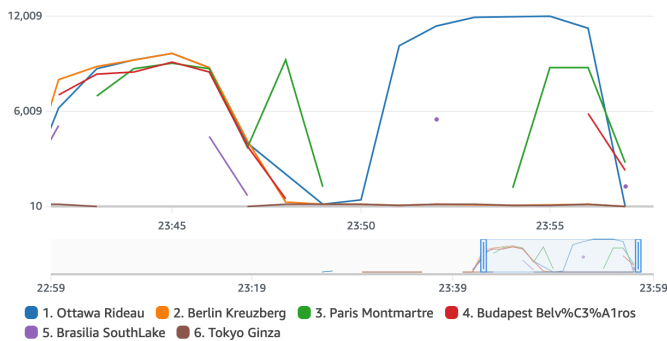
Most accessed keys (partition key)



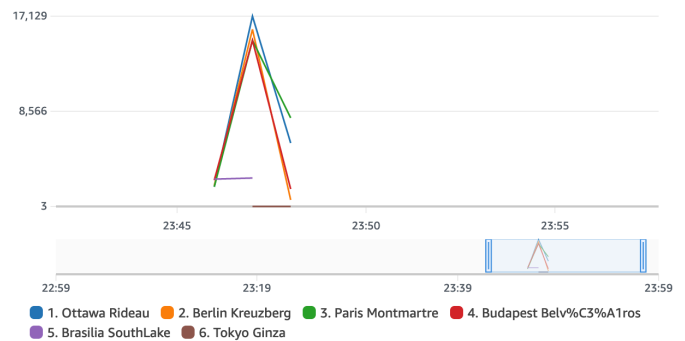
Most throttled keys (partition key)



Most accessed keys (partition and sort keys)



Most throttled keys (partition and sort keys)



Interaksi dengan fitur DynamoDB lain

Bagian berikut menjelaskan bagaimana CloudWatch Contributor Insights for DynamoDB berperilaku dan berinteraksi dengan beberapa fitur lain di DynamoDB.

Tabel global

CloudWatch Wawasan Kontributor untuk DynamoDB memantau replika tabel global sebagai tabel yang berbeda. Grafik Contributor Insights untuk replika di satu AWS Wilayah mungkin tidak menunjukkan pola yang sama dengan Wilayah lain. Hal ini karena data tulis direplikasi di semua replika dalam tabel global, tetapi setiap replika dapat melayani lalu lintas baca yang terikat Wilayah.

DynamoDB Accelerator (DAX)

CloudWatch Wawasan Kontributor untuk DynamoDB tidak menampilkan respons cache DAX. Wawasan Kontributor tersebut hanya menampilkan respons untuk mengakses tabel atau indeks sekunder global.

 Note

DynamoDB CCI tidak mendukung permintaan PartiQL.

Enkripsi diam

CloudWatch Contributor Insights untuk DynamoDB tidak memengaruhi cara kerja enkripsi di DynamoDB. Data kunci utama yang dipublikasikan di CloudWatch dienkripsi dengan file. Kunci milik AWS Namun, DynamoDB juga mendukung dan kunci Kunci yang dikelola AWS yang dikelola pelanggan.

CloudWatch Wawasan Kontributor untuk grafik DynamoDB menampilkan kunci partisi dan kunci sortir (jika ada) dari item yang sering diakses dan item yang sering dibatasi dalam teks biasa. Jika Anda memerlukan penggunaan AWS Key Management Service (KMS) untuk mengenkripsi kunci partisi tabel ini dan mengurutkan data kunci dengan Kunci yang dikelola AWS atau kunci yang dikelola pelanggan, Anda tidak boleh mengaktifkan CloudWatch Contributor Insights for DynamoDB untuk tabel ini.

Jika Anda memerlukan data kunci utama Anda untuk dienkripsi dengan Kunci yang dikelola AWS atau kunci yang dikelola pelanggan, Anda tidak boleh mengaktifkan CloudWatch Contributor Insights for DynamoDB untuk tabel tersebut.

Kontrol akses detail

CloudWatch Contributor Insights for DynamoDB tidak berfungsi secara berbeda untuk tabel dengan kontrol akses halus (FGAC). Dengan kata lain, setiap pengguna yang memiliki CloudWatch izin yang sesuai dapat melihat kunci utama yang dilindungi FGAC dalam grafik Contributor Insights. CloudWatch

Jika kunci utama tabel berisi data yang dilindungi FGAC yang tidak ingin Anda publikasikan CloudWatch, Anda tidak boleh mengaktifkan CloudWatch Contributor Insights for DynamoDB untuk tabel tersebut.

Kontrol akses

Anda mengontrol akses ke CloudWatch Contributor Insights for DynamoDB menggunakan (AWS Identity and Access Management IAM) dengan membatasi izin bidang kontrol DynamoDB dan izin bidang data. CloudWatch Untuk informasi selengkapnya lihat, [Menggunakan IAM dengan CloudWatch Contributor Insights for DynamoDB](#).

CloudWatch wawasan kontributor untuk penagihan DynamoDB

Biaya untuk Wawasan CloudWatch Kontributor untuk DynamoDB muncul di bagian [CloudWatch](#) tagihan bulanan Anda. Biaya ini dihitung berdasarkan jumlah peristiwa DynamoDB yang diproses. [Untuk tabel dan indeks sekunder global dengan CloudWatch Contributor Insights for DynamoDB diaktifkan, setiap item yang ditulis atau dibaca melalui operasi bidang data mewakili satu peristiwa.](#)

Jika tabel atau indeks sekunder global menyertakan kunci urutan, setiap item yang dibaca atau ditulis mewakili dua peristiwa. Hal ini karena DynamoDB mengidentifikasi kontributor teratas dari rangkaian waktu terpisah: satu untuk kunci partisi saja, dan satu untuk pasangan kunci partisi dan urutan.

Sebagai contoh, misalkan aplikasi Anda melakukan operasi DynamoDB berikut: `GetItem`, `PutItem`, dan `BatchWriteItem` yang memasukkan lima item

- Jika tabel atau indeks sekunder global Anda hanya memiliki kunci partisi, hasilnya adalah 7 peristiwa (1 untuk `GetItem`, 1 untuk `PutItem`, dan 5 untuk `BatchWriteItem`).
- Jika tabel atau indeks sekunder global Anda memiliki kunci partisi dan kunci urutan, hasilnya adalah 14 peristiwa (2 untuk `GetItem`, 2 untuk `PutItem`, dan 10 untuk `BatchWriteItem`).
- Operasi `Query` selalu menghasilkan 1 peristiwa, terlepas dari jumlah item yang dikembalikan.

Tidak seperti fitur DynamoDB lainnya CloudWatch , Contributor Insights untuk penagihan DynamoDB tidak bervariasi berdasarkan hal berikut:

- [Mode kapasitas](#) (disediakan vs. sesuai permintaan)
- Baik jika Anda melakukan permintaan baca atau tulis
- Ukuran (KB) item yang dibaca atau ditulis

Memulai dengan wawasan CloudWatch kontributor untuk DynamoDB

Bagian ini menjelaskan cara menggunakan Amazon CloudWatch Contributor Insights dengan konsol Amazon DynamoDB atau (). AWS Command Line Interface AWS CLI

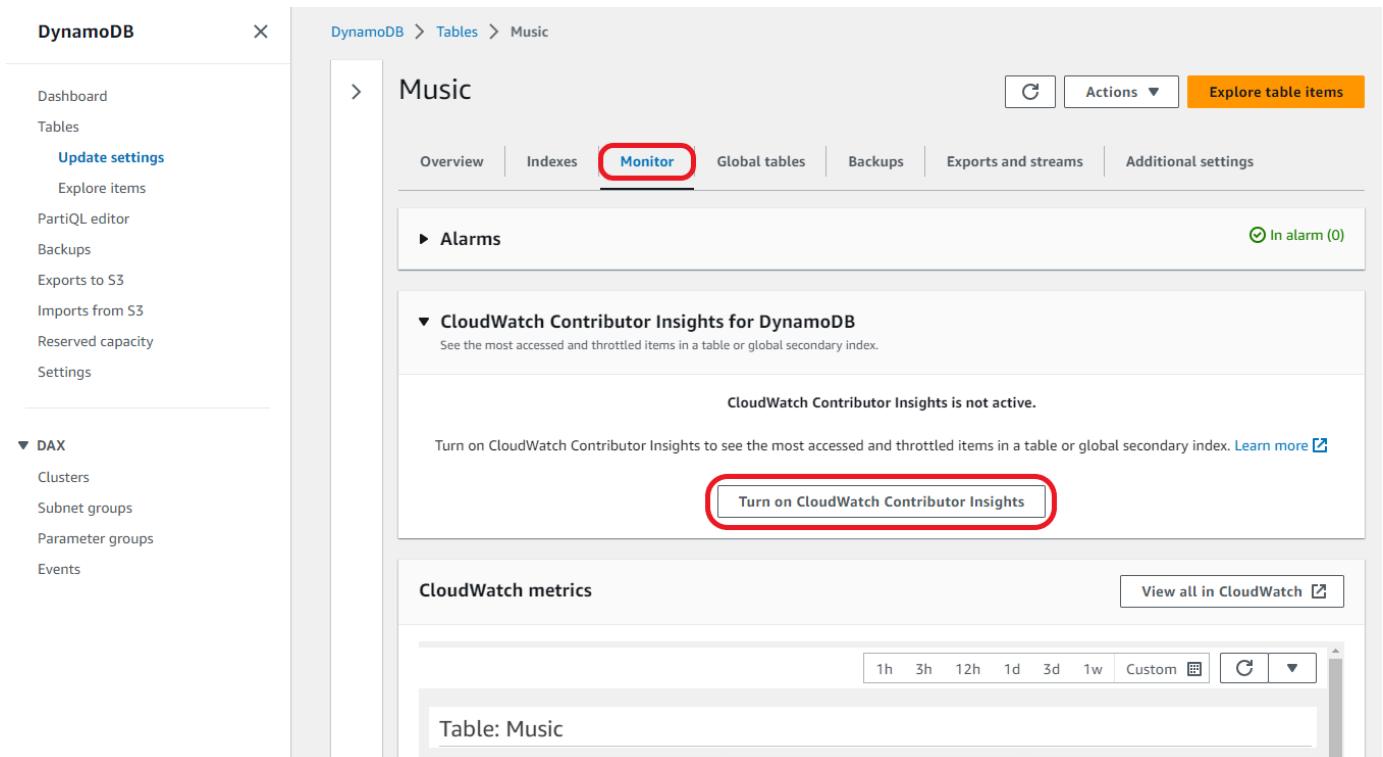
Dalam contoh berikut, Anda menggunakan tabel DynamoDB yang didefinisikan dalam tutorial [Mulai menggunakan DynamoDB](#).

Topik

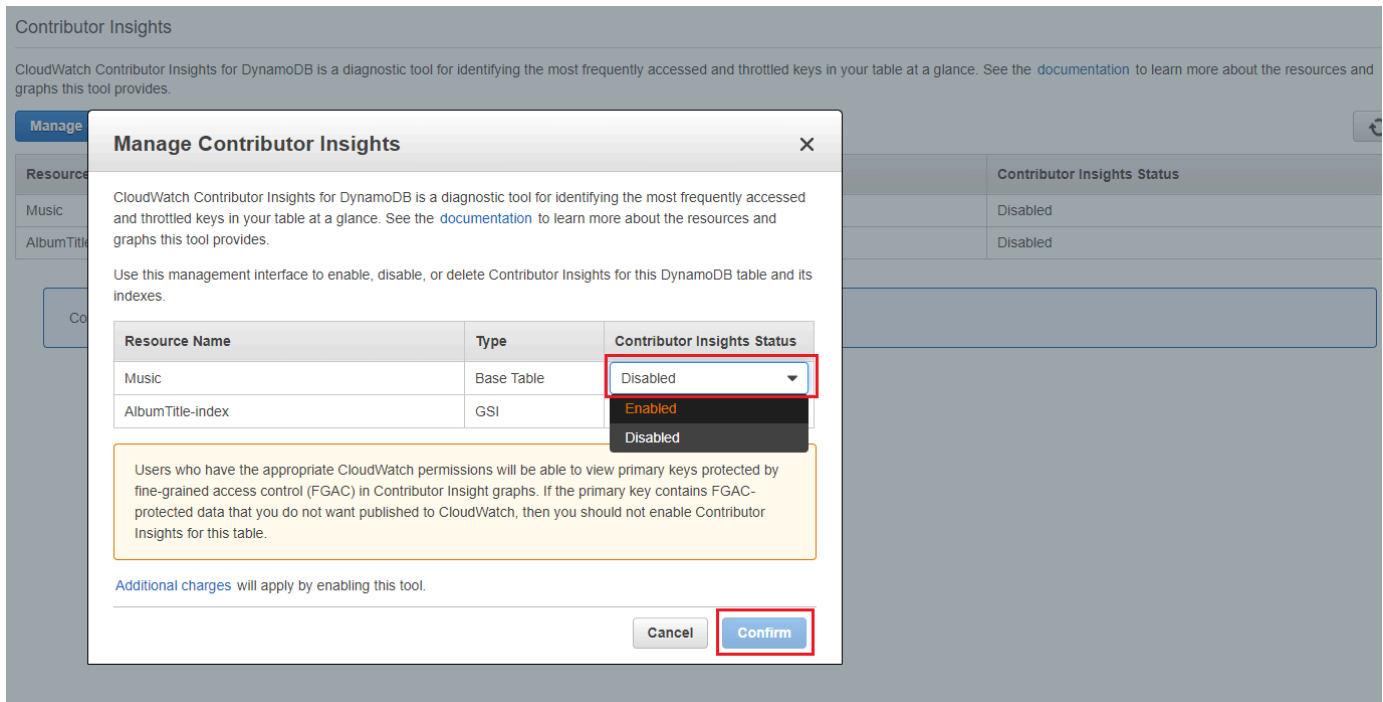
- [Menggunakan wawasan kontributor \(konsole\)](#)
- [Menggunakan wawasan kontributor \(AWS CLI\)](#)

Menggunakan wawasan kontributor (konsole)

1. [Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/.](https://console.aws.amazon.com/dynamodb/)
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tabel Music.
4. Pilih tab Pantau.
5. Pilih Aktifkan Wawasan CloudWatch Kontributor.

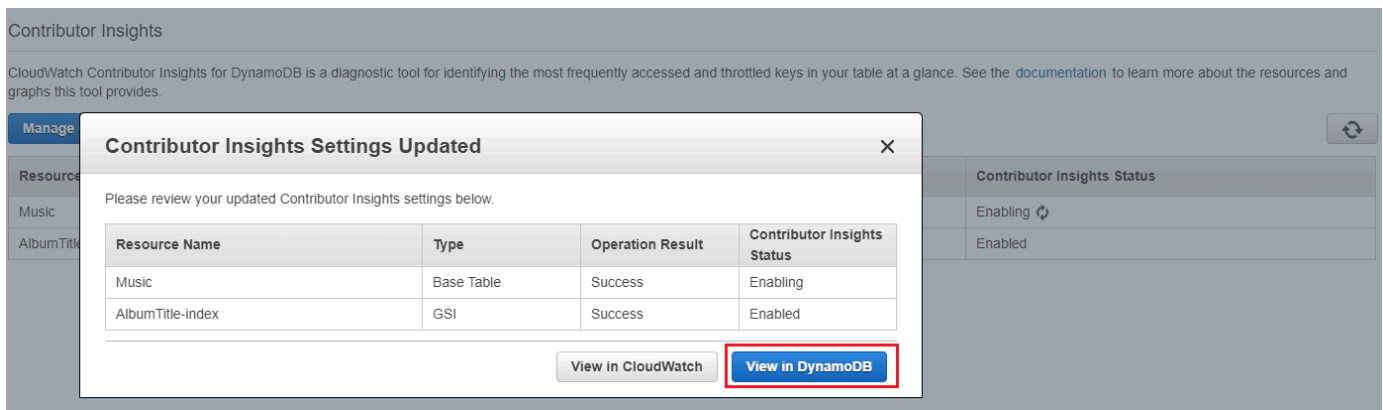


6. Di kotak dialog Kelola Wawasan Kontributor, di bagian Status Wawasan Kontributor, pilih Diaktifkan untuk tabel dasar Music dan indeks sekunder global AlbumTitle-index. Kemudian pilih Konfirmasi.



Jika operasi gagal, lihat [DescribeContributorInsights FailureException](#) di Referensi Amazon DynamoDB API untuk alasan yang mungkin.

7. Pilih Lihat di DynamoDB.



8. Grafik Wawasan Kontributor sekarang ditampilkan di tab Wawasan Kontributor untuk tabel Music.



Membuat CloudWatch alarm

Ikuti langkah-langkah ini untuk membuat CloudWatch alarm dan diberi tahu ketika kunci partisi apa pun menghabiskan lebih dari 50.000. [ConsumedThroughputUnits](#)

1. Masuk ke AWS Management Console dan buka CloudWatch konsol di <https://console.aws.amazon.com/cloudwatch/>
2. Di panel navigasi di sisi kiri konsol, pilih Wawasan Kontributor.
3. Pilih aturan DynamoDB ContributorInsights -PKC-Music.
4. Pilih drop down Tindakan.
5. Pilih Lihat dalam metrik.
6. Pilih Nilai Kontributor Maks.

Note

Hanya Max Contributor Value dan Maximum yang mengembalikan statistik yang berguna. Statistik lain dalam daftar ini tidak mengembalikan nilai yang berarti.

The screenshot shows the AWS Management Console interface. On the left, the navigation menu is visible, with 'Contributor Insights' highlighted. In the main content area, the 'Actions' dropdown menu is open, and 'View in metrics' is selected. The 'Max Contributor Value' metric is highlighted in the dropdown. The main area shows a graph for 'DynamoDBContributorInsights-PKC-Music-1580235665872' with 'No data available' message.

7. Di kolom Tindakan, Pilih Buat Alarm.

The screenshot shows the AWS Management Console interface. In the main content area, the 'Actions' dropdown menu is open, and 'Create alarm' is selected. The 'Math expression' field is visible, and the 'Create alarm' button is highlighted.

8. Masukkan nilai 50000 untuk ambang batas dan pilih Selanjutnya.

The screenshot shows the AWS CloudWatch 'Create Alarm' wizard. The 'Graph' section displays a line graph with a red horizontal threshold line at 50.0k. The 'Conditions' section is configured with 'Static' threshold type, 'Greater' condition, and a threshold value of 50000. The 'Label' is 'DynamoDBContributorInsights-PKC-Music-158749' and the 'Period' is '1 minute'. The 'Next' button is highlighted in orange.

9. Lihat [Menggunakan CloudWatch alarm Amazon](#) untuk detail tentang cara mengonfigurasi notifikasi untuk alarm.

Menggunakan wawasan kontributor (AWS CLI)

1. Aktifkan CloudWatch Contributor Insights untuk DynamoDB pada tabel dasar. Music

```
aws dynamodb update-contributor-insights --table-name Music --contributor-insights-action=ENABLE
```

2. Aktifkan Wawasan Kontributor untuk DynamoDB di indeks sekunder global AlbumTitle-index.

```
aws dynamodb update-contributor-insights --table-name Music --index-name AlbumTitle-index --contributor-insights-action=ENABLE
```

3. Dapatkan status dan aturan untuk tabel Music dan semua indeksnya.

```
aws dynamodb describe-contributor-insights --table-name Music
```

4. Nonaktifkan CloudWatch Contributor Insights untuk DynamoDB pada indeks sekunder global. `AlbumTitle-index`

```
aws dynamodb update-contributor-insights --table-name Music --index-name
AlbumTitle-index --contributor-insights-action=DISABLE
```

5. Dapatkan status tabel `Music` dan semua indeksinya.

```
aws dynamodb list-contributor-insights --table-name Music
```

Menggunakan IAM dengan wawasan CloudWatch kontributor untuk DynamoDB

Pertama kali Anda mengaktifkan Amazon CloudWatch Contributor Insights untuk Amazon DynamoDB, DynamoDB secara otomatis membuat peran terkait layanan (IAM) untuk Anda. AWS Identity and Access Management Peran ini, `AWSServiceRoleForDynamoDBCloudWatchContributorInsights`, memungkinkan DynamoDB CloudWatch mengelola aturan Contributor Insights atas nama Anda. Jangan hapus peran tertaut layanan ini. Jika Anda menghapusnya, semua aturan terkelola tidak akan lagi dibersihkan saat Anda menghapus tabel atau indeks sekunder global.

Untuk informasi selengkapnya tentang peran tertaut layanan, lihat [Menggunakan peran tertaut layanan](#) di Panduan Pengguna IAM.

Izin berikut diperlukan:

- Untuk mengaktifkan atau menonaktifkan CloudWatch Contributor Insights for DynamoDB, Anda harus memiliki `dynamodb:UpdateContributorInsights` izin pada tabel atau indeks.
- Untuk melihat CloudWatch Contributor Insights untuk grafik DynamoDB, Anda harus memiliki izin. `cloudwatch:GetInsightRuleReport`
- Untuk menjelaskan CloudWatch Contributor Insights for DynamoDB untuk tabel atau indeks DynamoDB tertentu, Anda harus memiliki izin. `dynamodb:DescribeContributorInsights`
- Untuk mencantumkan CloudWatch Contributor Insights untuk status DynamoDB untuk setiap tabel dan indeks sekunder global, Anda harus memiliki izin. `dynamodb>ListContributorInsights`

Contoh: Mengaktifkan atau menonaktifkan wawasan CloudWatch kontributor untuk DynamoDB

Kebijakan IAM berikut memberikan izin untuk mengaktifkan atau menonaktifkan CloudWatch Contributor Insights for DynamoDB.

```
{
  "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/*"
    }
  ]
}
```

Untuk tabel yang dienkripsi oleh kunci KMS, pengguna harus memiliki izin kms:Decrypt untuk dapat memperbarui Wawasan Kontributor.

```
{
  "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"
      ],
    }
  ]
}

```

Contoh: Ambil laporan aturan wawasan CloudWatch kontributor

Kebijakan IAM berikut memberikan izin untuk mengambil laporan aturan CloudWatch Contributor Insights.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetInsightRuleReport"
      ],
      "Resource": "arn:aws:cloudwatch:*:*:insight-rule/
DynamoDBContributorInsights*"
    }
  ]
}

```

Contoh: Terapkan wawasan CloudWatch kontributor secara selektif untuk izin DynamoDB berdasarkan sumber daya

Kebijakan IAM berikut memberikan izin untuk memperbolehkan tindakan `ListContributorInsights` dan `DescribeContributorInsights` serta menolak tindakan `UpdateContributorInsights` untuk indeks sekunder global tertentu.

```
{
  "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"
    }
  ]
}
```

Menggunakan peran terkait layanan untuk CloudWatch Contributor Insights untuk DynamoDB

CloudWatch [Wawasan Kontributor untuk DynamoDB menggunakan AWS Identity and Access Management peran terkait layanan \(IAM\)](#). Peran terkait layanan adalah jenis unik peran IAM yang ditautkan langsung ke CloudWatch Contributor Insights for DynamoDB. Peran terkait layanan telah ditentukan sebelumnya oleh CloudWatch Contributor Insights for DynamoDB dan menyertakan semua izin yang diperlukan layanan untuk memanggil layanan lain atas nama Anda. AWS

Peran terkait layanan membuat pengaturan CloudWatch Contributor Insights untuk DynamoDB lebih mudah karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. CloudWatch Contributor Insights for DynamoDB mendefinisikan izin dari peran terkait layanannya, dan kecuali

ditentukan lain, hanya Contributor Insights untuk DynamoDB yang dapat mengambil perannya. CloudWatch Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, dan kebijakan izin tersebut tidak dapat dilampirkan ke entitas IAM lainnya.

Untuk informasi tentang layanan lain yang mendukung peran terkait layanan, lihat [Layanan AWS yang Berfungsi dengan IAM](#) dan cari layanan yang memiliki Ya di kolom Peran Terkait Layanan. Pilih Ya dengan sebuah tautan untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Izin peran terkait layanan untuk CloudWatch Contributor Insights untuk DynamoDB

CloudWatch Contributor Insights untuk DynamoDB menggunakan peran terkait layanan bernama. `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` Tujuan dari peran terkait layanan adalah untuk mengizinkan Amazon DynamoDB mengelola CloudWatch aturan Amazon Contributor Insights yang dibuat untuk tabel DynamoDB dan indeks sekunder global, atas nama Anda.

Peran tertaut layanan `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` memercayai layanan berikut untuk mengambil peran tersebut:

- `contributorinsights.dynamodb.amazonaws.com`

Kebijakan izin peran memungkinkan CloudWatch Contributor Insights for DynamoDB menyelesaikan tindakan berikut pada sumber daya yang ditentukan:

- Tindakan: `Create and manage Insight Rules` pada `DynamoDBContributorInsights`

Anda harus mengonfigurasi izin untuk mengizinkan entitas IAM (seperti pengguna, grup, atau peran) untuk membuat, menyunting, atau menghapus peran terhubung dengan layanan. Untuk informasi lebih lanjut, lihat [Izin Peran Tertaut Layanan](#) di Panduan Pengguna IAM.

Membuat peran terkait layanan untuk CloudWatch Contributor Insights untuk DynamoDB

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda mengaktifkan Contributor Insights di AWS Management Console, the, atau AWS API AWS CLI, CloudWatch Contributor Insights for DynamoDB akan membuat peran terkait layanan untuk Anda.

Jika Anda menghapus peran terkait layanan ini, dan ingin membuatnya lagi, Anda dapat mengulangi proses yang sama untuk membuat kembali peran tersebut di akun Anda. Saat Anda mengaktifkan Contributor Insights, CloudWatch Contributor Insights for DynamoDB membuat peran terkait layanan untuk Anda lagi.

Mengedit peran terkait layanan untuk CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB tidak mengizinkan Anda mengedit peran terkait layanan. `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin merujuk peran tersebut. Namun, Anda dapat mengedit deskripsi peran ini menggunakan IAM. Untuk informasi selengkapnya, silakan lihat [Mengedit peran tertaut layanan](#) dalam Panduan Pengguna IAM.

Menghapus peran terkait layanan untuk CloudWatch Contributor Insights for DynamoDB

Anda tidak perlu menghapus peran

`AWSServiceRoleForDynamoDBCloudWatchContributorInsights` secara manual. Saat Anda menonaktifkan Contributor Insights di AWS Management Console, the, atau AWS API AWS CLI, CloudWatch Contributor Insights for DynamoDB membersihkan sumber daya.

Anda juga dapat menggunakan konsol IAM, AWS CLI atau AWS API untuk menghapus peran terkait layanan secara manual. Untuk melakukannya, Anda harus membersihkan sumber daya untuk peran tertaut layanan terlebih dahulu, lalu Anda dapat menghapusnya secara manual.

Note

Jika layanan CloudWatch Contributor Insights for DynamoDB menggunakan peran saat Anda mencoba menghapus sumber daya, maka penghapusan mungkin gagal. Jika hal itu terjadi, tunggu beberapa menit dan coba mengoperasikannya lagi.

Untuk menghapus peran terkait layanan secara manual menggunakan IAM

Gunakan konsol IAM, the AWS CLI, atau AWS API untuk menghapus peran `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` terkait layanan. Untuk informasi selengkapnya, silakan lihat [Menghapus Peran Terkait Layanan](#) di Panduan Pengguna IAM.

Menggunakan Pemberitahuan Notifikasi Pengguna AWS Pengguna dengan Amazon DynamoDB

Anda dapat menggunakan [Notifikasi Pengguna](#) untuk menyiapkan saluran pengiriman agar mendapatkan pemberitahuan tentang peristiwa DynamoDB. Anda akan menerima notifikasi saat ada sebuah peristiwa yang cocok dengan sebuah aturan yang Anda tentukan. Anda dapat menerima

notifikasi untuk peristiwa melalui beberapa saluran, termasuk email, notifikasi obrolan [AWS Chatbot](#), atau notifikasi push [AWS Console Mobile Application](#). Anda juga dapat melihat notifikasi di [Pusat Notifikasi Konsol](#). Notifikasi Pengguna mendukung agregasi, hal ini akan dapat mengurangi jumlah notifikasi yang Anda terima selama peristiwa-peristiwa tertentu.

Praktik terbaik untuk mendesain dan merancang dengan DynamoDB

Gunakan bagian ini untuk menemukan rekomendasi dengan cepat untuk memaksimalkan performa dan meminimalkan biaya throughput saat menggunakan Amazon DynamoDB.

Topik

- [Desain NoSQL untuk DynamoDB](#)
- [Menggunakan perlindungan penghapusan untuk melindungi tabel Anda](#)
- [Menggunakan DynamoDB Well-Architected Lens untuk mengoptimalkan beban kerja DynamoDB Anda](#)
- [Praktik Terbaik untuk merancang dan menggunakan kunci partisi secara efektif](#)
- [Praktik terbaik untuk menggunakan kunci urutan untuk mengatur data](#)
- [Praktik terbaik untuk menggunakan indeks sekunder di DynamoDB](#)
- [Praktik terbaik untuk menyimpan atribut dan item besar](#)
- [Praktik terbaik untuk penanganan data deret waktu di DynamoDB](#)
- [Praktik terbaik untuk mengelola many-to-many hubungan](#)
- [Praktik terbaik untuk mengimplementasikan sistem basis data hibrida](#)
- [Praktik terbaik untuk memodelkan data relasional di DynamoDB](#)
- [Praktik terbaik untuk mengkueri dan memindai data](#)
- [Praktik terbaik untuk desain tabel DynamoDB](#)
- [Praktik terbaik untuk desain tabel global DynamoDB](#)
- [Praktik terbaik untuk mengelola bidang kontrol di DynamoDB](#)
- [Praktik Terbaik untuk Memahami Laporan AWS Penagihan dan Penggunaan Anda](#)

- [Pertimbangan saat menggunakan AWS PrivateLink untuk Amazon DynamoDB](#)

Desain NoSQL untuk DynamoDB

Sistem basis data NoSQL seperti Amazon DynamoDB menggunakan model alternatif untuk manajemen data, seperti pasangan nilai kunci atau penyimpanan dokumen. Saat Anda beralih dari

sistem manajemen basis data relasional ke sistem basis data NoSQL seperti DynamoDB, penting untuk memahami perbedaan utama dan pendekatan desain tertentu.

Topik

- [Perbedaan antara desain data relasional dan NoSQL](#)
- [Dua konsep utama untuk desain NoSQL](#)
- [Mendekati desain NoSQL](#)
- [NoSQL Workbench untuk DynamoDB](#)

Perbedaan antara desain data relasional dan NoSQL

Sistem basis data relasional (RDBMS) dan basis data NoSQL memiliki keunggulan dan kelemahan yang berbeda:

- Di RDBMS, data dapat dikueri secara fleksibel, tetapi kueri relatif mahal dan tidak dapat diskalakan dengan baik dalam situasi lalu lintas tinggi (lihat [Langkah pertama untuk memodelkan data relasional di DynamoDB](#)).
- Dalam basis data NoSQL seperti DynamoDB, data dapat dikueri secara efisien dalam sejumlah cara terbatas, di luar itu kueri bisa jadi mahal dan lambat.

Perbedaan ini membuat desain basis data menjadi berbeda di antara kedua sistem:

- Di RDBMS, Anda mendesain untuk fleksibilitas tanpa perlu mengkhawatirkan detail penerapan atau performa. Optimasi kueri umumnya tidak memengaruhi desain skema, tetapi normalisasi itu penting.
- Di DynamoDB, Anda mendesain skema secara spesifik untuk membuat kueri yang paling umum dan penting seefisien dan seterjangkau mungkin. Struktur data Anda disesuaikan dengan kebutuhan spesifik kasus penggunaan bisnis Anda.

Dua konsep utama untuk desain NoSQL

Desain NoSQL membutuhkan pola pikir yang berbeda dari desain RDBMS. Untuk RDBMS, Anda dapat melanjutkan dan membuat model data yang dinormalisasi tanpa memikirkan pola akses. Anda kemudian dapat memperluasnya nanti ketika ada pertanyaan dan persyaratan kueri baru. Anda dapat mengatur setiap jenis data ke dalam tabelnya sendiri.

Perbedaan desain NoSQL

- Sebaliknya, Anda tidak boleh mulai merancang skema untuk DynamoDB sampai Anda mengetahui pertanyaan yang perlu dijawab. Memahami masalah bisnis dan kasus penggunaan aplikasi di awal sangat penting.
- Anda harus mempertahankan tabel sesedikit mungkin dalam aplikasi DynamoDB. Memiliki lebih sedikit tabel membuat segala sesuatunya lebih terukur, memerlukan lebih sedikit manajemen izin, dan mengurangi overhead untuk aplikasi DynamoDB Anda. Hal ini juga dapat membantu menjaga biaya pencadangan tetap rendah secara keseluruhan.

Mendekati desain NoSQL

Langkah pertama dalam mendesain aplikasi DynamoDB adalah mengidentifikasi pola kueri tertentu yang harus dipenuhi oleh sistem.

Secara khusus, penting untuk memahami tiga properti dasar dari pola akses aplikasi Anda sebelum memulai:

- Ukuran data: Mengetahui jumlah data yang akan disimpan dan diminta pada satu waktu akan membantu menentukan cara paling efektif untuk mempartisi data.
- Bentuk data: Alih-alih membentuk kembali data saat kueri diproses (seperti yang dilakukan sistem RDBMS), basis data NoSQL mengatur data sehingga bentuknya dalam basis data tersebut sesuai dengan apa yang akan dikueri. Ini adalah faktor kunci dalam meningkatkan kecepatan dan skalabilitas.
- Kecepatan data: DynamoDB menskalakan dengan meningkatkan jumlah partisi fisik yang tersedia untuk memproses kueri, dan dengan mendistribusikan data secara efisien ke seluruh partisi tersebut. Mengetahui berapa beban kueri puncak di awal mungkin akan membantu menentukan cara mempartisi data agar dapat menggunakan kapasitas I/O dengan sebaik-baiknya.

Setelah mengidentifikasi persyaratan kueri tertentu, Anda bisa mengatur data menurut prinsip umum yang mengatur performa:

- Menyimpan data terkait bersama-sama. Penelitian tentang optimasi tabel perutean 20 tahun yang lalu menemukan bahwa "lokalitas referensi" adalah satu-satunya faktor terpenting dalam mempercepat waktu respons: menyimpan data terkait di satu tempat. Ini juga berlaku dalam sistem NoSQL saat ini, penyimpanan data terkait dalam jarak dekat memiliki dampak besar pada

biaya dan performa. Alih-alih mendistribusikan item data terkait di beberapa tabel, Anda harus menyimpan item terkait di sistem NoSQL Anda sedekat mungkin.

Aturan umumnya, Anda harus mempertahankan tabel sesedikit mungkin dalam aplikasi DynamoDB.

Pengecualian adalah kasus yang melibatkan data deret waktu bervolume tinggi, atau set data yang memiliki pola akses yang sangat berbeda. Tabel tunggal dengan indeks terbalik biasanya dapat mengaktifkan kueri sederhana untuk membuat dan mengambil struktur data hierarki kompleks yang diperlukan oleh aplikasi Anda.

- Menggunakan urutan. Item terkait dapat dikelompokkan bersama dan dikueri secara efisien jika desain utamanya menyebabkan item tersebut disortir bersama. Ini adalah strategi desain NoSQL yang penting.
- Mendistribusikan kueri. Penting juga agar kueri dalam jumlah besar tidak terfokus pada satu bagian basis data, yang dapat melebihi kapasitas I/O. Sebagai gantinya, Anda harus mendesain kunci data untuk mendistribusikan lalu lintas secara merata di seluruh partisi sebanyak mungkin, menghindari "hot spot".
- Menggunakan indeks sekunder global. Dengan membuat indeks sekunder global tertentu, Anda dapat mengaktifkan kueri yang berbeda dari yang dapat didukung oleh tabel utama Anda, dan itu masih cepat dan relatif murah.

Prinsip-prinsip umum ini diterjemahkan ke dalam beberapa pola desain umum yang dapat Anda gunakan untuk memodelkan data secara efisien di DynamoDB.

NoSQL Workbench untuk DynamoDB

[NoSQL Workbench untuk DynamoDB](#) adalah aplikasi GUI sisi klien lintas platform yang dapat Anda gunakan untuk pengembangan dan operasi basis data modern. Ini tersedia untuk Windows, macOS, dan Linux. NoSQL Workbench adalah alat pengembangan visual yang menyediakan fitur pemodelan data, visualisasi data, pembuatan data sampel, dan pengembangan kueri untuk membantu Anda mendesain, membuat, mengkueri, dan mengelola tabel DynamoDB. Dengan NoSQL Workbench untuk DynamoDB, Anda dapat membuat model data baru dari, atau mendesain model berdasarkan, model data yang sudah ada yang memenuhi pola akses data aplikasi Anda. Anda juga dapat mengimpor dan mengeksport model data yang didesain pada akhir proses. Lihat informasi yang lebih lengkap di [Membangun model data dengan NoSQL Workbench](#)

Menggunakan perlindungan penghapusan untuk melindungi tabel Anda

Perlindungan penghapusan dapat menjaga tabel Anda agar tidak terhapus secara tidak sengaja. Bagian ini menjelaskan beberapa praktik terbaik untuk menggunakan perlindungan penghapusan.

- Untuk semua tabel produksi yang aktif, praktik terbaiknya adalah mengaktifkan pengaturan perlindungan penghapusan dan melindungi tabel ini dari penghapusan yang tidak disengaja. Ini juga berlaku untuk replika global.
- Saat menyajikan kasus penggunaan pengembangan aplikasi, jika alur kerja manajemen tabel menyertakan sering menghapus serta membuat ulang tabel pengembangan dan penahanan, maka pengaturan perlindungan penghapusan dapat dimatikan. Ini akan memungkinkan penghapusan tabel tersebut dengan sengaja oleh pengguna utama IAM yang berwenang.

Untuk informasi selengkapnya tentang perlindungan penghapusan, lihat [Menggunakan perlindungan penghapusan](#).

Menggunakan DynamoDB Well-Architected Lens untuk mengoptimalkan beban kerja DynamoDB Anda

Bagian ini menjelaskan Amazon DynamoDB Well-Architected Lens, kumpulan prinsip desain, dan panduan untuk mendesain beban kerja DynamoDB yang dirancang dengan baik.

Mengoptimalkan biaya pada tabel DynamoDB

Bagian ini mencakup praktik terbaik tentang cara mengoptimalkan biaya untuk tabel DynamoDB yang ada. Anda harus melihat strategi berikut untuk mengetahui strategi pengoptimalan biaya yang paling sesuai dengan kebutuhan Anda dan mendekatinya secara berulang. Setiap strategi akan memberikan gambaran umum tentang hal-hal yang mungkin memengaruhi biaya Anda, tanda-tanda yang harus dicari, dan panduan preskriptif tentang cara mengurangnya.

Topik

- [Evaluasi biaya Anda di tingkat tabel](#)
- [Evaluasi mode kapasitas tabel Anda](#)
- [Evaluasi pengaturan penskalaan otomatis tabel Anda](#)

- [Evaluasi pilihan kelas tabel Anda](#)
- [Mengidentifikasi sumber daya yang tidak terpakai](#)
- [Evaluasi pola penggunaan tabel Anda](#)
- [Evaluasi penggunaan Streams](#)
- [Evaluasi kapasitas yang disediakan untuk penyediaan ukuran yang tepat](#)

Evaluasi biaya Anda di tingkat tabel

Alat Cost Explorer yang ditemukan di dalamnya AWS Management Console memungkinkan Anda melihat biaya yang dipecah berdasarkan jenisnya, seperti biaya baca, tulis, penyimpanan, dan cadangan. Anda juga dapat melihat biaya-biaya ini dirangkum berdasarkan periode seperti bulan atau hari.

Salah satu tantangan yang dapat dihadapi administrator adalah ketika biaya hanya pada satu tabel tertentu perlu ditinjau. Beberapa data ini tersedia melalui konsol DynamoDB atau melalui panggilan ke `DescribeTable` API, tetapi Cost Explorer tidak, secara default, memungkinkan Anda untuk memfilter atau mengelompokkan berdasarkan biaya yang terkait dengan tabel tertentu. Bagian ini akan menunjukkan cara menggunakan pemberian tag untuk melakukan analisis biaya tabel individual di Cost Explorer.

Topik

- [Cara melihat biaya tabel DynamoDB tunggal](#)
- [Tampilan default Cost Explorer](#)
- [Cara menggunakan dan menerapkan tag tabel di Cost Explorer](#)

Cara melihat biaya tabel DynamoDB tunggal

Baik Amazon AWS Management Console DynamoDB dan `DescribeTable` API akan menampilkan informasi tentang satu tabel, termasuk skema kunci utama, indeks apa pun di tabel, dan ukuran dan jumlah item tabel dan indeks apa pun. Ukuran tabel, ditambah ukuran indeks, dapat digunakan untuk menghitung biaya penyimpanan bulanan untuk tabel Anda. Misalnya, \$0,25 per GB di wilayah us-east-1.

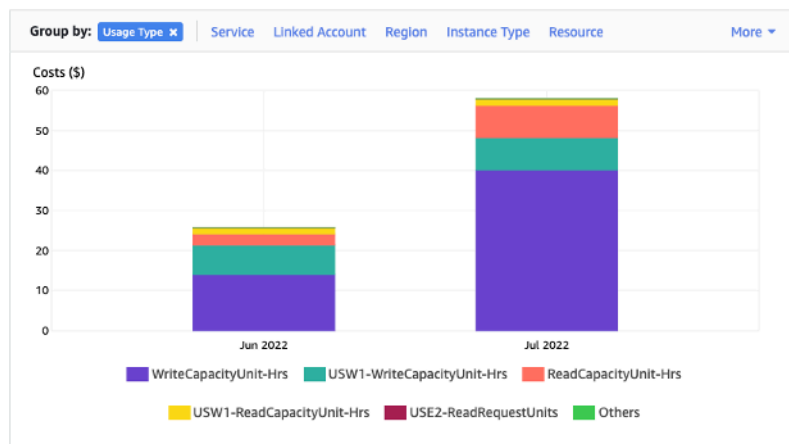
Jika tabel dalam mode kapasitas yang disediakan, pengaturan RCU dan WCU saat ini juga dikembalikan. Ini dapat digunakan untuk menghitung biaya baca dan tulis saat ini untuk tabel, tetapi biaya ini dapat berubah, terutama jika tabel telah dikonfigurasi dengan Penskalaan Otomatis.

Note

Jika tabel berada dalam mode kapasitas sesuai permintaan, maka `DescribeTable` tidak akan membantu memperkirakan biaya throughput, karena penagihannya berdasarkan penggunaan aktual, tidak disediakan dalam satu periode.

Tampilan default Cost Explorer

Tampilan default Cost Explorer menyediakan bagan yang menunjukkan biaya sumber daya yang digunakan seperti throughput dan penyimpanan. Anda dapat memilih untuk mengelompokkan biaya berdasarkan periode, seperti total berdasarkan bulan atau hari. Biaya penyimpanan, baca, tulis, dan fitur lainnya dapat dipecah dan dibandingkan juga.



Cara menggunakan dan menerapkan tag tabel di Cost Explorer

Secara default, Cost Explorer tidak menyediakan ringkasan biaya untuk satu tabel tertentu, karena Cost Explorer akan menggabungkan biaya beberapa tabel menjadi total. Namun, Anda dapat menggunakan [pemberian tag sumber daya AWS](#) untuk mengidentifikasi setiap tabel dengan tag metadata. Tag adalah pasangan nilai kunci yang dapat Anda gunakan untuk berbagai tujuan, seperti mengidentifikasi semua sumber daya milik proyek atau departemen. Untuk contoh ini, kita akan menganggap Anda memiliki tabel bernama MyTable.

1. Tetapkan tag dengan kunci `table_name` dan nilai. MyTable
2. [Aktifkan tag di dalam Cost Explorer](#) lalu filter pada nilai tag untuk mendapatkan lebih banyak visibilitas ke setiap biaya tabel.

Note

Tag mungkin akan mulai muncul di Cost Explorer setelah satu atau dua hari

Anda dapat mengatur sendiri tag metadata di konsol, atau melalui otomatisasi seperti AWS CLI atau SDK. AWS Pertimbangkan untuk mewajibkan tag `table_name` ditetapkan sebagai bagian dari proses pembuatan tabel baru organisasi Anda. Untuk tabel yang ada, tersedia utilitas Python yang akan menemukan dan menerapkan tag ini ke semua tabel yang ada di wilayah tertentu di akun Anda. Lihat [Tagger Tabel Eponymous GitHub untuk lebih jelasnya](#).

Evaluasi mode kapasitas tabel Anda

Bagian ini memberikan gambaran umum tentang cara memilih mode kapasitas yang sesuai untuk tabel DynamoDB Anda. Setiap mode disesuaikan untuk memenuhi kebutuhan beban kerja yang berbeda dalam hal respons terhadap perubahan throughput, serta cara penagihan penggunaan tersebut. Anda harus menyeimbangkan faktor-faktor ini ketika membuat keputusan.

Topik

- [Mode kapasitas tabel yang tersedia](#)
- [Kapan harus memilih mode kapasitas sesuai permintaan](#)
- [Kapan harus mode kapasitas yang disediakan](#)
- [Faktor lain yang perlu dipertimbangkan saat memilih mode kapasitas tabel](#)

Mode kapasitas tabel yang tersedia

Saat membuat tabel DynamoDB, Anda harus memilih mode kapasitas sesuai permintaan atau yang disediakan. Anda dapat beralih di antara mode kapasitas baca/tulis satu kali setiap 24 jam. Satu-satunya pengecualian untuk ini adalah jika Anda mengalihkan tabel mode yang disediakan ke mode sesuai permintaan: Anda dapat beralih kembali ke mode yang disediakan dalam periode 24 jam yang sama.

Edit read/write capacity

Capacity mode [Info](#)

On-demand
Simplify billing by paying for the actual reads and writes your application performs.

Provisioned
Manage and optimize the price by allocating read/write capacity in advance.

[Cancel](#) [Save changes](#)

Mode kapasitas sesuai permintaan

Mode kapasitas sesuai permintaan didesain untuk menghilangkan kebutuhan merencanakan atau menyediakan kapasitas tabel DynamoDB Anda. Dalam mode ini, tabel Anda akan langsung mengakomodasi permintaan ke tabel Anda tanpa perlu menaikkan atau menurunkan skala sumber daya apa pun (hingga dua kali lipat throughput puncak tabel sebelumnya).

Tabel sesuai permintaan ditagih dengan menghitung jumlah permintaan sebenarnya terhadap tabel tersebut, sehingga Anda hanya akan membayar untuk yang Anda gunakan, bukan yang telah disediakan.

Tabel kapasitas yang disediakan

Mode kapasitas yang disediakan adalah model yang lebih tradisional di mana Anda dapat menentukan berapa banyak kapasitas tabel yang tersedia untuk permintaan baik secara langsung atau dengan bantuan penskalaan otomatis. Karena kapasitas tertentu disediakan untuk tabel pada waktu tertentu, penagihan didasarkan pada kapasitas yang disediakan, bukan jumlah permintaan. Melebihi kapasitas yang dialokasikan juga dapat menyebabkan tabel menolak permintaan dan mengurangi pengalaman pengguna aplikasi Anda.

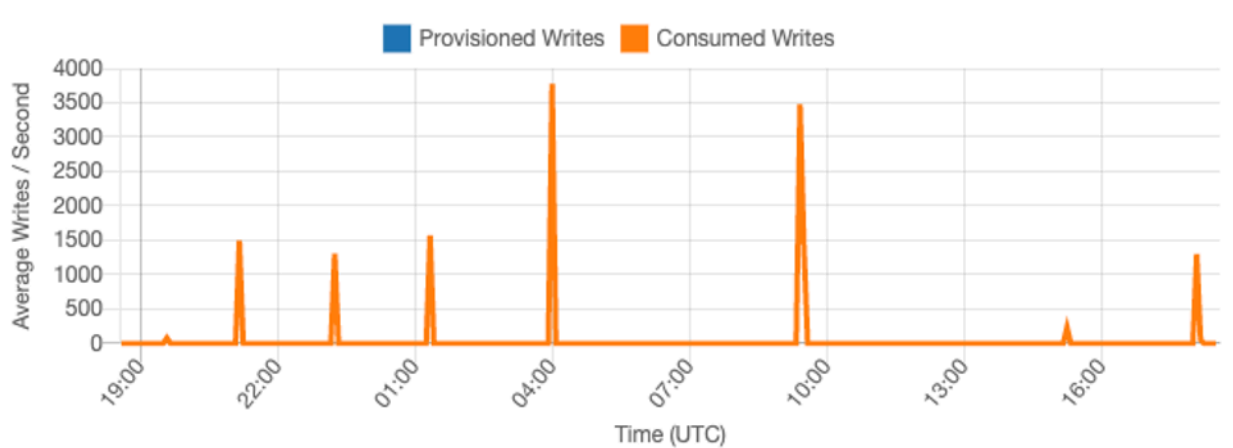
Mode kapasitas yang disediakan memerlukan keseimbangan antara tidak menyediakan tabel secara berlebihan atau kurang untuk menjaga throttling tetap rendah dan biaya disesuaikan.

Kapan harus memilih mode kapasitas sesuai permintaan

Saat mengoptimalkan biaya, mode sesuai permintaan adalah pilihan terbaik ketika Anda memiliki beban kerja yang mirip dengan grafik berikut.

Faktor-faktor berikut berkontribusi pada jenis beban kerja ini:

- Waktu permintaan yang tidak dapat diprediksi (mengakibatkan lonjakan lalu lintas)
- Volume permintaan variabel (dihasilkan dari beban kerja batch)
- Turun ke nol atau di bawah 18% dari puncak pada jam tertentu (akibat dari lingkungan pengembangan atau pengujian)



Untuk beban kerja dengan faktor-faktor di atas, menggunakan penskalaan otomatis untuk mempertahankan kapasitas yang cukup di atas meja untuk merespons lonjakan lalu lintas kemungkinan akan menyebabkan tabel dilebih-lebihkan dan biaya lebih dari yang diperlukan atau tabel sedang disediakan dan permintaan dibatasi secara tidak perlu.

Karena tabel sesuai permintaan ditagih berdasarkan permintaan, kita tidak perlu melakukan tindakan apa pun di tingkat tabel untuk mengoptimalkan biaya. Tabel sesuai permintaan harus dievaluasi secara berkala untuk memverifikasi apakah beban kerja masih memiliki faktor-faktor di atas. Jika beban kerja telah stabil, pertimbangkan untuk mengubah ke mode yang disediakan untuk lebih mengoptimalkan biaya.

Kapan harus mode kapasitas yang disediakan

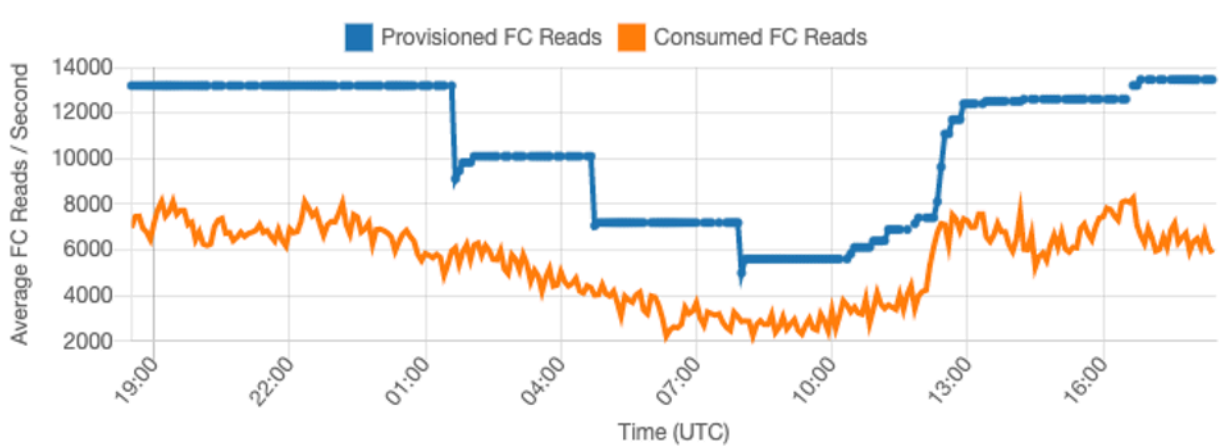
Beban kerja yang ideal untuk mode kapasitas yang disediakan adalah beban kerja dengan pola penggunaan yang lebih dapat diprediksi seperti grafik di bawah ini.

Note

Sebaiknya tinjau metrik pada periode yang berbutir halus, seperti 14 hari atau 24 jam, sebelum mengambil tindakan atas kapasitas yang Anda berikan.

Faktor-faktor berikut berkontribusi pada jenis beban kerja ini:

- Lalu lintas yang dapat diprediksi/bersiklus untuk jam atau hari tertentu
- Lonjakan lalu lintas jangka pendek terbatas



Karena volume lalu lintas pada jam atau hari tertentu lebih stabil, kita dapat mengatur kapasitas tabel yang disediakan relatif dekat dengan kapasitas tabel yang digunakan sebenarnya. Pengoptimalan biaya pada kapasitas tabel yang disediakan pada akhirnya merupakan upaya untuk membuat kapasitas yang disediakan (garis biru) sedekat mungkin dengan kapasitas yang digunakan (garis oranye) tanpa meningkatkan `ThrottledRequests` pada tabel. Jarak antara kedua garis tersebut merupakan kapasitas yang terbuang serta jaminan terhadap pengalaman pengguna yang buruk akibat throttling.

DynamoDB menyediakan penskalaan otomatis untuk kapasitas tabel yang disediakan yang akan menyeimbangkan kapasitas tersebut secara otomatis atas nama Anda. Ini memungkinkan Anda memantau kapasitas yang digunakan sepanjang hari dan mengatur kapasitas tabel berdasarkan beberapa variabel.

On-demand
Simplify billing by paying for the actual reads and writes your application performs.

Provisioned
Manage and optimize the price by allocating read/write capacity in advance.

Table capacity

Read capacity

Auto scaling [Info](#)
Dynamically adjusts provisioned throughput capacity on your behalf in response to actual traffic patterns.

On
 Off

Minimum capacity units	Maximum capacity units	Target utilization (%)
<input type="text" value="100"/>	<input type="text" value="500"/>	<input type="text" value="70"/>

Initial provisioned units [Info](#)

Unit kapasitas minimum

Anda dapat mengatur kapasitas tabel minimum untuk membatasi throttling, tetapi tindakan ini tidak akan mengurangi biaya tabel. Jika tabel Anda memiliki periode penggunaan rendah yang diikuti dengan lonjakan penggunaan tinggi secara tiba-tiba, mengatur ke kapasitas minimum dapat mencegah penskalaan otomatis mengatur kapasitas tabel terlalu rendah.

Unit kapasitas maksimum

Anda dapat mengatur kapasitas tabel maksimum untuk membatasi penskalaan tabel yang lebih tinggi dari yang dimaksudkan. Pertimbangkan untuk menerapkan jumlah maksimum untuk tabel Pengembangan atau Pengujian dengan pengujian beban skala besar tidak diinginkan. Anda dapat menetapkan maksimum untuk tabel apa pun, tetapi pastikan untuk mengevaluasi pengaturan ini secara berkala terhadap garis dasar tabel saat menggunakannya dalam Produksi untuk mencegah throttling yang tidak disengaja.

Pemanfaatan target

Menetapkan pemanfaatan target pada tabel adalah cara utama pengoptimalan biaya untuk kapasitas tabel yang disediakan. Menetapkan nilai persentase yang lebih rendah di sini akan meningkatkan jumlah tabel yang disediakan secara berlebihan, sehingga meningkatkan biaya, tetapi mengurangi

risiko throttling. Menetapkan nilai persentase yang lebih tinggi akan mengurangi jumlah tabel yang disediakan secara berlebihan, tetapi meningkatkan risiko throttling.

Faktor lain yang perlu dipertimbangkan saat memilih mode kapasitas tabel

Saat memutuskan antara dua mode, ada beberapa faktor lain yang perlu dipertimbangkan.

Kapasitas terpesan

Untuk kapasitas tabel yang disediakan, DynamoDB menawarkan kemampuan untuk membeli kapasitas terpesan untuk kapasitas baca dan tulis Anda (unit kapasitas tulis yang direplikasi (RWCU) dan tabel Standard-IA saat ini tidak memenuhi syarat). Jika memilih membeli reservasi untuk kapasitas ini, Anda dapat mengurangi biaya tabel hingga persentase yang signifikan.

Saat memutuskan antara dua mode tabel, pertimbangkan seberapa besar pengaruh diskon tambahan ini terhadap biaya tabel. Dalam banyak kasus, beban kerja yang relatif tidak dapat diprediksi bisa lebih murah jika dijalankan pada kapasitas tabel yang disediakan secara berlebihan dengan kapasitas terpesan.

Meningkatkan prediktabilitas beban kerja Anda

Dalam beberapa situasi, beban kerja tampaknya memiliki pola yang dapat diprediksi dan tidak dapat diprediksi. Meskipun ini dapat dengan mudah didukung dengan tabel sesuai permintaan, biaya kemungkinan akan lebih terjangkau jika pola beban kerja yang tidak dapat diprediksi dapat ditingkatkan.

Salah satu penyebab paling umum dari pola ini adalah impor batch. Jenis lalu lintas ini sering kali dapat melebihi kapasitas dasar tabel sedemikian rupa sehingga throttling akan terjadi jika tabel tersebut dijalankan. Agar beban kerja seperti ini tetap berjalan pada kapasitas tabel yang disediakan, pertimbangkan opsi berikut:

- Jika batch terjadi pada waktu yang dijadwalkan, Anda dapat menjadwalkan peningkatan kapasitas penskalaan otomatis sebelum batch dijalankan
- Jika batch terjadi secara acak, pertimbangkan untuk memperpanjang waktu berjalan daripada mengeksekusi secepat mungkin
- Tambahkan periode kenaikan ke impor di mana kecepatan impor mulai kecil tetapi perlahan-lahan meningkat selama beberapa menit sampai penskalaan otomatis memiliki kesempatan untuk mulai menyesuaikan kapasitas tabel

Evaluasi pengaturan penskalaan otomatis tabel Anda

Bagian ini memberikan gambaran umum tentang cara mengevaluasi pengaturan penskalaan otomatis pada tabel DynamoDB Anda. [Penskalaan otomatis Amazon DynamoDB](#) adalah fitur yang mengelola tabel dan throughput indeks sekunder global (GSI) berdasarkan lalu lintas aplikasi dan metrik pemanfaatan target Anda. Fitur ini akan memastikan tabel atau GSI Anda memiliki kapasitas yang diperlukan untuk pola aplikasi Anda.

Layanan penskalaan AWS otomatis akan memantau pemanfaatan tabel Anda saat ini dan membandingkannya dengan nilai pemanfaatan target. `TargetValue` Fitur ini akan memberi tahu Anda jika sudah waktunya untuk menambah atau mengurangi kapasitas yang dialokasikan.

Topik

- [Memahami pengaturan penskalaan otomatis Anda](#)
- [Cara mengidentifikasi tabel dengan pemanfaatan target rendah \(\$\leq 50\%\$ \)](#)
- [Cara mengatasi beban kerja dengan varian musiman](#)
- [Cara mengatasi lonjakan beban kerja dengan pola yang tidak diketahui](#)
- [Cara mengatasi beban kerja dengan aplikasi tertaut](#)

Memahami pengaturan penskalaan otomatis Anda

Mendefinisikan nilai yang benar untuk pemanfaatan target, langkah awal, dan nilai akhir adalah aktivitas yang memerlukan keterlibatan dari tim operasi Anda. Hal ini memungkinkan Anda untuk menentukan nilai dengan benar berdasarkan penggunaan aplikasi historis, yang akan digunakan untuk memicu kebijakan penskalaan otomatis AWS. Target pemanfaatan adalah persentase total kapasitas yang perlu dicapai selama jangka waktu tertentu sebelum aturan penskalaan otomatis berlaku.

Ketika Anda menetapkan target pemanfaatan yang tinggi (target sekitar 90%), artinya lalu lintas Anda harus lebih tinggi dari 90% untuk jangka waktu tertentu sebelum penskalaan otomatis dimulai. Jangan menggunakan target pemanfaatan yang tinggi kecuali aplikasi Anda sangat konstan dan tidak menerima lonjakan lalu lintas.

Ketika Anda menetapkan pemanfaatan yang sangat rendah (target kurang dari 50%), artinya aplikasi Anda harus mencapai 50% dari kapasitas yang disediakan sebelum memicu kebijakan penskalaan otomatis. Kecuali lalu lintas aplikasi Anda tumbuh pada tingkat yang sangat agresif, ini biasanya diterjemahkan ke dalam kapasitas yang tidak terpakai dan sumber daya yang terbuang.

Cara mengidentifikasi tabel dengan pemanfaatan target rendah ($\leq 50\%$)

Anda dapat menggunakan AWS CLI atau AWS Management Console untuk memantau dan mengidentifikasi kebijakan penskalaan otomatis di sumber daya DynamoDB Anda: TargetValues

AWS CLI

1. Kembalikan seluruh daftar sumber daya dengan menjalankan perintah berikut:

```
aws application-autoscaling describe-scaling-policies --service-namespace dynamodb
```

Perintah ini akan mengembalikan seluruh daftar kebijakan penskalaan otomatis yang dikeluarkan ke sumber daya DynamoDB apa pun. Jika hanya ingin mengambil sumber daya dari tabel tertentu, Anda dapat menambahkan `-resource-id` parameter. Sebagai contoh:

```
aws application-autoscaling describe-scaling-policies --service-namespace dynamodb --resource-id "table/<table-name>"
```

2. Kembalikan hanya kebijakan penskalaan otomatis untuk GSI tertentu dengan menjalankan perintah berikut

```
aws application-autoscaling describe-scaling-policies --service-namespace dynamodb --resource-id "table/<table-name>/index/<gsi-name>"
```

Nilai-nilai yang kami minati untuk kebijakan penskalaan otomatis disorot di bawah ini. Kami ingin memastikan bahwa nilai target lebih besar dari 50% untuk menghindari penyediaan berlebih. Anda akan mendapatkan hasil yang mirip dengan berikut ini:

```
{
  "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. Masuk ke AWS Management Console dan navigasikan ke halaman CloudWatch layanan di [Memulai dengan AWS Management Console](#). Pilih AWS wilayah yang sesuai jika perlu.
2. Di bilah navigasi kiri, pilih Tabel. Di halaman Tabel, pilih Nama tabel.

3. Di halaman Detail Tabel di tab Pengaturan Tambahan, tinjau pengaturan penskalaan otomatis untuk tabel Anda.

Read/write capacity
The read/write capacity mode controls how you are charged for read and write throughput and how you manage capacity.

Capacity mode
Provisioned

Table capacity

Read capacity auto scaling On	Write capacity auto scaling On
Provisioned read capacity units 5	Provisioned write capacity units 5
Provisioned range for reads 1 - 10	Provisioned range for writes 1 - 10
Target read capacity utilization 70%	Target write capacity utilization 70%

► **Index capacity**

Untuk indeks, perluas tab Kapasitas Indeks untuk meninjau pengaturan penskalaan otomatis indeks.

Read/write capacity		
The read/write capacity mode controls how you are charged for read and write throughput and how you manage capacity.		
Capacity mode Provisioned		
Table capacity		
Read capacity auto scaling On	Write capacity auto scaling On	
Provisioned read capacity units 5	Provisioned write capacity units 5	
Provisioned range for reads 1 - 10	Provisioned range for writes 1 - 10	
Target read capacity utilization 70%	Target write capacity utilization 70%	
▼ Index capacity		
Index name	Read capacity	Write capacity
GSI1PK-GSI1SK-index	Range: 1 - 10 Auto scaling at 70% Current provisioned units: 5	Range: 1 - 10 Auto scaling at 70% Current provisioned units: 5

Jika nilai pemanfaatan target Anda kurang dari atau sama dengan 50%, Anda harus mempelajari metrik pemanfaatan tabel Anda untuk mengetahui apakah nilai tersebut [kurang tersedia atau disediakan secara berlebihan](#).

Cara mengatasi beban kerja dengan varian musiman

Pertimbangkan skenario berikut: aplikasi Anda sering kali beroperasi di bawah nilai rata-rata minimum, tetapi target pemanfaatannya rendah sehingga aplikasi Anda dapat bereaksi dengan cepat terhadap peristiwa yang terjadi pada jam-jam tertentu dalam sehari dan Anda memiliki kapasitas yang memadai serta menghindari throttling. Skenario ini umum terjadi ketika Anda memiliki aplikasi yang sangat sibuk selama jam kantor normal (9 pagi hingga 5 sore) tetapi kemudian berfungsi pada tingkat dasar setelah jam kerja. Karena beberapa pengguna akan mulai terhubung sebelum jam 9 pagi, aplikasi menggunakan ambang batas rendah ini untuk meningkatkan dengan cepat guna mencapai kapasitas yang diperlukan selama jam sibuk.

Skenario ini akan seperti berikut:

- Antara jam 5 sore dan 9 pagi ConsumedWriteCapacity unit tetap berada di antara 90 dan 100

- Pengguna mulai terhubung ke aplikasi sebelum jam 9 pagi dan unit kapasitas meningkat pesat (nilai maksimum yang Anda lihat adalah 1500 WCU)
- Rata-rata, penggunaan aplikasi Anda berkisar antara 800 hingga 1200 selama jam kerja

Jika skenario sebelumnya berlaku untuk Anda, sebaiknya gunakan [penskalaan otomatis terjadwal](#), yaitu tabel Anda masih dapat mengonfigurasi aturan penskalaan otomatis aplikasi, tetapi dengan pemanfaatan target yang kurang agresif yang hanya menyediakan kapasitas tambahan pada interval tertentu yang Anda perlukan.

Anda dapat menggunakan AWS CLI untuk menjalankan langkah-langkah berikut untuk membuat aturan penskalaan otomatis terjadwal yang akan dijalankan berdasarkan waktu hari dan hari dalam seminggu.

1. Daftarkan GSI atau tabel DynamoDB Anda sebagai target yang dapat diskalakan dengan Application Auto Scaling. Target yang dapat diskalakan adalah sumber daya yang skalanya dapat diperkecil atau diperbesar oleh Application Auto Scaling .

```
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. Siapkan tindakan terjadwal sesuai dengan kebutuhan Anda.

Kita akan membutuhkan dua aturan untuk mencakup skenario: satu untuk meningkatkan skala dan satu lagi untuk menurunkan skala. Aturan pertama untuk meningkatkan skala tindakan terjadwal:

```
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"
```

Aturan kedua untuk menurunkan skala tindakan terjadwal:

```
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. Jalankan perintah berikut untuk memvalidasi bahwa kedua aturan telah diaktifkan:

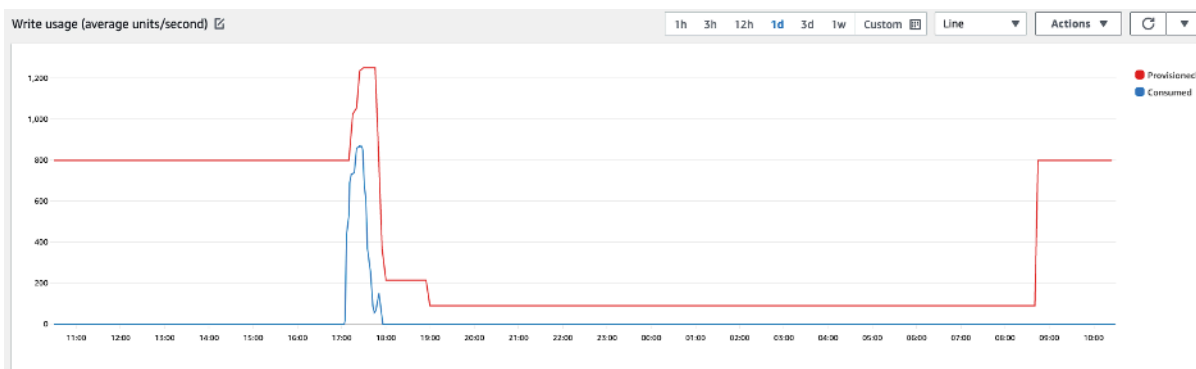
```
aws application-autoscaling describe-scheduled-actions --service-namespace dynamodb
```

Anda akan mendapatkan hasil seperti ini:

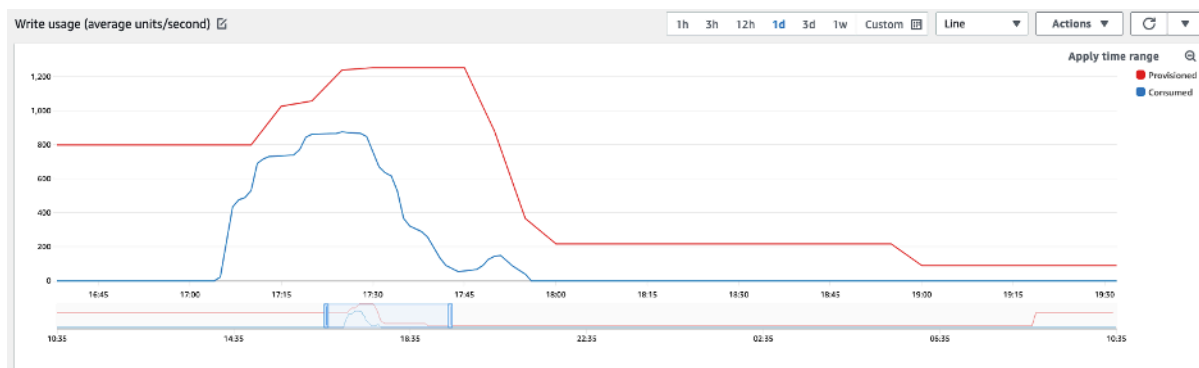
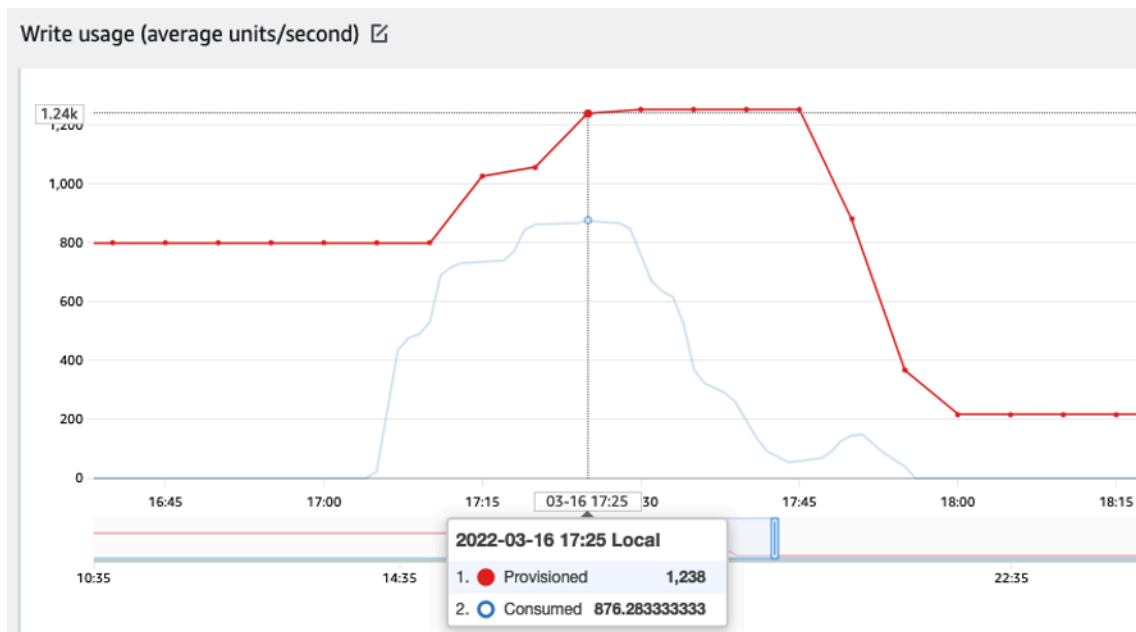
```
{  
  "ScheduledActions": [  
    {  
      "ScheduledActionName": "my-5-8-scheduled-down-action",  
      "ScheduledActionARN":  
        "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/  
table/<table-name>:scheduledActionName/my-5-8-scheduled-down-action",  
      "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"
}
]
}
```

Gambar berikut menunjukkan beban kerja sampel yang selalu mempertahankan pemanfaatan target 70%. Perhatikan bagaimana aturan penskalaan otomatis masih tetap diterapkan dan throughput tidak akan berkurang.



Jika diperbesar, kita dapat melihat adanya lonjakan dalam aplikasi yang memicu ambang penskalaan otomatis sebesar 70%, sehingga memaksa penskalaan otomatis untuk memulai dan menyediakan kapasitas tambahan yang diperlukan untuk tabel. Tindakan penskalaan otomatis terjadwal akan memengaruhi nilai maksimum dan minimum, dan Anda bertanggung jawab untuk mengaturnya.



Cara mengatasi lonjakan beban kerja dengan pola yang tidak diketahui

Dalam skenario ini, aplikasi menggunakan target pemanfaatan yang sangat rendah karena Anda belum mengetahui pola aplikasi, dan Anda ingin memastikan beban kerja Anda tidak mengalami throttling.

Sebaiknya gunakan [mode kapasitas sesuai permintaan](#). Tabel sesuai permintaan sangat cocok untuk lonjakan beban kerja yang tidak Anda ketahui pola lalu lintasnya. Dengan mode kapasitas sesuai permintaan, Anda membayar per permintaan atas pembacaan dan penulisan data yang dilakukan aplikasi Anda pada tabel Anda. Anda tidak perlu menentukan berapa banyak throughput baca dan tulis yang Anda harapkan untuk dijalankan oleh aplikasi Anda, karena DynamoDB langsung mengakomodasi beban kerja Anda saat beban kerja tersebut naik atau turun.

Cara mengatasi beban kerja dengan aplikasi tertaut

Dalam skenario ini, aplikasi bergantung pada sistem lain, seperti skenario pemrosesan batch yang dapat menghasilkan lonjakan besar dalam lalu lintas sesuai dengan peristiwa dalam logika aplikasi.

Pertimbangkan untuk mengembangkan logika penskalaan otomatis khusus yang bereaksi terhadap peristiwa tersebut saat Anda dapat meningkatkan `TargetValues` dan kapasitas tabel bergantung pada kebutuhan spesifik Anda. Anda bisa mendapatkan keuntungan dari Amazon EventBridge dan menggunakan kombinasi AWS layanan seperti Lambda dan Step Functions untuk menanggapi kebutuhan aplikasi spesifik Anda.

Evaluasi pilihan kelas tabel Anda

Bagian ini memberikan gambaran umum tentang cara memilih kelas tabel yang sesuai untuk tabel DynamoDB Anda. Dengan peluncuran kelas tabel Standard Infrequent-Access (Standard-IA), Anda sekarang memiliki kemampuan mengoptimalkan tabel untuk biaya penyimpanan yang lebih rendah atau biaya throughput yang lebih rendah.

Topik

- [Kelas tabel yang tersedia](#)
- [Kapan harus memilih kelas tabel DynamoDB Standard](#)
- [Kapan harus memilih kelas tabel DynamoDB Standard-IA](#)
- [Faktor lain yang perlu dipertimbangkan saat memilih kelas tabel](#)

Kelas tabel yang tersedia

Ketika membuat Tabel DynamoDB, Anda harus memilih DynamoDB Standard atau DynamoDB Standard-IA untuk kelas tabel. Kelas tabel dapat diubah dua kali dalam periode 30 hari, sehingga Anda selalu dapat mengubahnya di masa mendatang. Memilih salah satu kelas tabel tidak akan memengaruhi performa, ketersediaan, keandalan, atau daya tahan tabel.

Update table class


Table class

Select table class to optimize your table's cost based on your workload requirements and data access patterns.

Choose table class

DynamoDB Standard
The default general-purpose table class. Recommended for the vast majority of tables that store frequently accessed data, with throughput (reads and writes) as the dominant table cost.

DynamoDB Standard-IA
Recommended for tables that store data that is infrequently accessed, with storage as the dominant table cost.

i Table class updates is a background process. The time to update your table class depends on your table traffic, storage size, and other related variables. You can still access your table normally while it is converted. Note that no more than two table class updates on your table are allowed in a 30-day trailing period. [Learn more](#) 

Cancel

Save changes

Kelas tabel Standard

Kelas tabel Standard adalah opsi default untuk tabel baru. Opsi ini mempertahankan saldo penagihan asli DynamoDB yang menawarkan keseimbangan biaya throughput dan penyimpanan untuk tabel dengan data yang sering diakses.

Kelas tabel Standard-IA

Kelas tabel Standard-IA menawarkan biaya penyimpanan yang lebih rendah (~60% lebih rendah) untuk beban kerja yang memerlukan penyimpanan data jangka panjang dengan pembaruan atau pembacaan yang jarang. Karena kelas ini dioptimalkan untuk akses yang jarang, baca dan tulis akan dikenakan biaya yang sedikit lebih tinggi (~25% lebih tinggi) dibandingkan kelas tabel Standard.

Kapan harus memilih kelas tabel DynamoDB Standard

Kelas tabel DynamoDB Standard paling cocok untuk tabel yang biaya penyimpanannya sekitar 50% atau kurang dari keseluruhan biaya bulanan tabel. Keseimbangan biaya ini menunjukkan beban kerja yang secara berkala mengakses atau memperbarui item yang sudah disimpan dalam DynamoDB.

Kapan harus memilih kelas tabel DynamoDB Standard-IA

Kelas tabel DynamoDB Standard-IA paling cocok untuk tabel yang biaya penyimpanannya sekitar 50% atau lebih dari keseluruhan biaya bulanan tabel. Keseimbangan biaya ini menunjukkan beban

kerja yang membuat atau membaca lebih sedikit item per bulan daripada yang disimpan dalam penyimpanan.

Penggunaan umum untuk kelas tabel Standard-IA adalah memindahkan data yang jarang diakses ke tabel Standard-IA individual. Untuk informasi selengkapnya, lihat [Mengoptimalkan biaya penyimpanan beban kerja Anda dengan kelas tabel Amazon DynamoDB Standard-IA](#).

Faktor lain yang perlu dipertimbangkan saat memilih kelas tabel

Saat memutuskan antara dua kelas tabel, ada beberapa faktor lain yang layak untuk dipertimbangkan sebagai bagian dari keputusan Anda.

Kapasitas terpesan

Pembelian kapasitas terpesan untuk tabel yang menggunakan kelas tabel Standard-IA saat ini tidak didukung. Saat beralih dari tabel Standard dengan kapasitas terpesan ke tabel Standard-IA tanpa kapasitas terpesan, Anda mungkin tidak melihat manfaat biaya.

Mengidentifikasi sumber daya yang tidak terpakai

Bagian ini memberikan gambaran umum tentang cara mengevaluasi sumber daya yang Anda tidak terpakai secara berkala. Seiring berkembangnya persyaratan aplikasi Anda, Anda harus memastikan tidak ada sumber daya yang tidak terpakai dan berkontribusi terhadap biaya Amazon DynamoDB yang tidak diperlukan. Prosedur yang dijelaskan di bawah ini akan menggunakan CloudWatch metrik Amazon untuk mengidentifikasi sumber daya yang tidak digunakan dan akan membantu Anda mengidentifikasi dan mengambil tindakan terhadap sumber daya tersebut untuk mengurangi biaya.

Anda dapat memantau DynamoDB CloudWatch menggunakan, yang mengumpulkan dan memproses data mentah dari DynamoDB menjadi metrik yang dapat dibaca, mendekati real-time. Statistik ini disimpan selama jangka waktu tertentu, sehingga Anda dapat mengakses informasi historis untuk lebih memahami pemanfaatan Anda. Secara default, data metrik DynamoDB dikirim secara otomatis. CloudWatch Untuk informasi lebih lanjut, lihat [Apa itu Amazon CloudWatch?](#) dan [Retensi metrik](#) di Panduan CloudWatch Pengguna Amazon.

Topik

- [Cara mengidentifikasi sumber daya yang tidak terpakai](#)
- [Mengidentifikasi sumber daya tabel yang tidak terpakai](#)
- [Membersihkan sumber daya tabel yang tidak terpakai](#)
- [Mengidentifikasi sumber daya GSI yang tidak terpakai](#)

- [Membersihkan sumber daya GSI yang tidak terpakai](#)
- [Membersihkan tabel global yang tidak terpakai](#)
- [Membersihkan cadangan atau point-in-time pemulihan yang tidak terpakai \(PITR\)](#)

Cara mengidentifikasi sumber daya yang tidak terpakai

Untuk mengidentifikasi tabel atau indeks yang tidak digunakan, kita akan melihat CloudWatch metrik berikut selama periode 30 hari untuk memahami apakah ada pembacaan atau penulisan aktif di tabel atau pembacaan apa pun pada indeks sekunder global (GSI):

[ConsumedReadCapacityUnits](#)

Jumlah unit kapasitas baca yang terpakai selama jangka waktu tertentu, sehingga Anda dapat melacak jumlah kapasitas terpakai yang telah Anda gunakan. Anda dapat mengambil total kapasitas baca yang digunakan untuk tabel dan semua indeks sekunder globalnya, atau untuk indeks sekunder global tertentu.

[ConsumedWriteCapacityUnits](#)

Jumlah unit kapasitas tulis yang terpakai selama jangka waktu tertentu, sehingga Anda dapat melacak jumlah kapasitas terpakai yang telah Anda gunakan. Anda dapat mengambil total kapasitas tulis yang digunakan untuk tabel dan semua indeks sekunder globalnya, atau untuk indeks sekunder global tertentu.

Mengidentifikasi sumber daya tabel yang tidak terpakai

Amazon CloudWatch adalah layanan pemantauan dan observabilitas yang menyediakan metrik tabel DynamoDB yang akan Anda gunakan untuk mengidentifikasi sumber daya yang tidak digunakan. CloudWatch metrik dapat dilihat melalui AWS Management Console maupun melalui AWS Command Line Interface

AWS Command Line Interface

Untuk melihat metrik tabel Anda melalui AWS Command Line Interface, Anda dapat menggunakan perintah berikut.

1. Pertama, evaluasi pembacaan tabel Anda:

```
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>
```

Untuk menghindari kesalahan dalam mengidentifikasi tabel sebagai tidak terpakai, evaluasi metrik dalam jangka waktu yang lebih lama. Pilih rentang waktu mulai dan waktu berakhir yang sesuai, seperti 30 hari, dan periode yang sesuai, seperti 86400.

Dalam data yang dikembalikan, setiap Jumlah di atas 0 menunjukkan bahwa tabel yang Anda evaluasi menerima lalu lintas baca selama periode tersebut.

Hasil berikut menunjukkan tabel yang menerima lalu lintas baca pada periode yang dievaluasi:

```
{  
  "Timestamp": "2022-08-25T19:40:00Z",  
  "Sum": 36023355.0,  
  "Unit": "Count"  
},  
{  
  "Timestamp": "2022-08-12T19:40:00Z",  
  "Sum": 38025777.5,  
  "Unit": "Count"  
},
```

Hasil berikut menunjukkan tabel yang tidak menerima lalu lintas baca pada periode yang dievaluasi:

```
{  
  "Timestamp": "2022-08-01T19:50:00Z",  
  "Sum": 0.0,  
  "Unit": "Count"  
},  
{  
  "Timestamp": "2022-08-20T19:50:00Z",  
  "Sum": 0.0,  
  "Unit": "Count"  
},
```

2. Selanjutnya, evaluasi penulisan tabel Anda:

```
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>
```

Untuk menghindari kesalahan dalam mengidentifikasi tabel sebagai tidak terpakai, sebaiknya Anda mengevaluasi metrik dalam jangka waktu yang lebih lama. Pilih rentang waktu mulai dan waktu berakhir yang sesuai, seperti 30 hari, dan periode yang sesuai, seperti 86400.

Dalam data yang dikembalikan, setiap Jumlah di atas 0 menunjukkan bahwa tabel yang Anda evaluasi menerima lalu lintas baca selama periode tersebut.

Hasil berikut menunjukkan tabel yang menerima lalu lintas tulis pada periode yang dievaluasi:

```
{
  "Timestamp": "2022-08-19T20:15:00Z",
  "Sum": 41014457.0,
  "Unit": "Count"
},
{
  "Timestamp": "2022-08-18T20:15:00Z",
  "Sum": 40048531.0,
  "Unit": "Count"
},
```

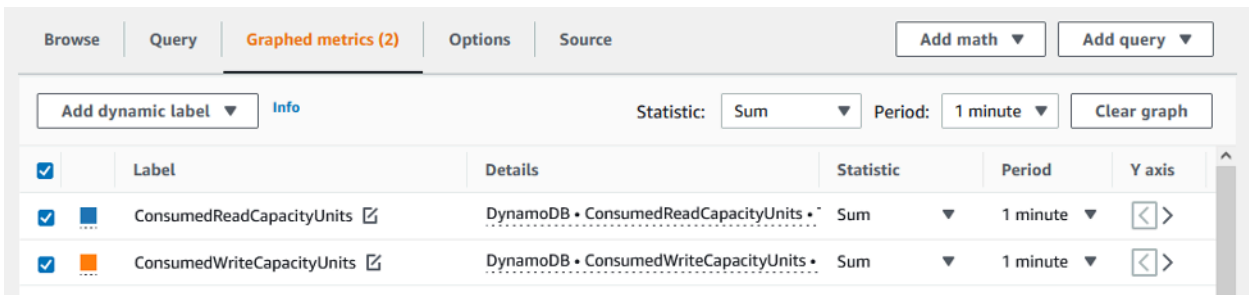
Hasil berikut menunjukkan tabel yang tidak menerima lalu lintas tulis pada periode yang dievaluasi:

```
{
  "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


Langkah-langkah berikut akan memungkinkan Anda untuk mengevaluasi penggunaan sumber daya Anda melalui AWS Management Console.


1. Masuk ke AWS konsol dan arahkan ke halaman CloudWatch layanan di <https://console.aws.amazon.com/cloudwatch/>. Pilih AWS wilayah yang sesuai di kanan atas konsol, jika perlu.
2. Di bilah navigasi kiri, cari bagian Metrik lalu pilih Semua metrik.
3. Tindakan di atas akan membuka dashboard dengan dua panel. Di panel atas, Anda akan melihat metrik bergrafik saat ini Di bagian bawah, Anda akan memilih metrik yang tersedia untuk dibuat grafik. Pilih DynamoDB di panel bawah.
4. Di panel pemilihan metrik DynamoDB, pilih kategori Metrik Tabel untuk menampilkan metrik tabel Anda di wilayah saat ini.
5. Identifikasi nama tabel Anda dengan menggulir ke bawah menu, lalu pilih metrik `ConsumedReadCapacityUnits` dan `ConsumedWriteCapacityUnits` untuk tabel Anda.
6. Pilih tab Metrik bergrafik (2) dan sesuaikan kolom Statistik menjadi Jumlah.



7. Untuk menghindari kesalahan dalam mengidentifikasi tabel sebagai tidak terpakai, sebaiknya Anda mengevaluasi metrik dalam jangka waktu yang lebih lama. Di bagian atas panel grafik, pilih kerangka waktu yang sesuai, seperti 1 bulan, untuk mengevaluasi tabel Anda. Pilih Kustom, pilih 1 Bulan di dropdown, lalu pilih Terapkan.

CloudWatch > Metrics

DynamoDB Table Usage 

1h 3h 12h 1d 3d 1w Custom (1M) 

Absolute **Relative** Local time zone ▼

Count

554,769

293,863

32,956

Minutes 1 3 5 15 30 45

Hours 1 2 3 6 8 12

Days 1 2 3 4 5 6

Weeks 1 2 4 6

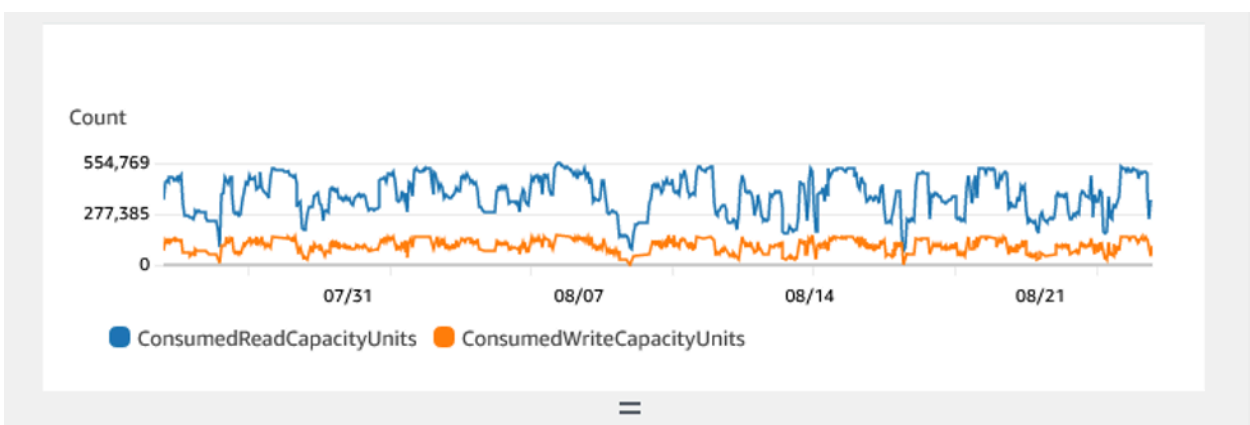
Months 3 6 12 15

1 Months ▼

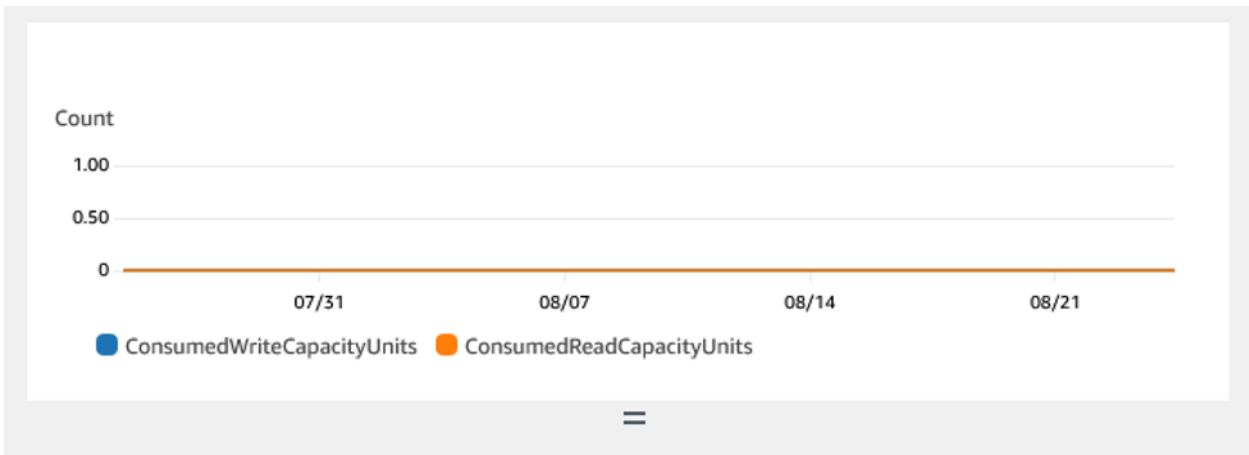
Clear Cancel Apply

8. Evaluasi metrik bergrafik untuk tabel Anda guna menentukan apakah tabel sedang digunakan. Metrik di atas 0 menunjukkan bahwa tabel telah digunakan selama jangka waktu evaluasi. Grafik datar pada 0 untuk baca dan tulis menunjukkan tabel yang tidak terpakai.

Gambar berikut menunjukkan tabel dengan lalu lintas baca:



Gambar berikut menunjukkan tabel tanpa lalu lintas baca:



Membersihkan sumber daya tabel yang tidak terpakai

Jika Anda telah mengidentifikasi sumber daya tabel yang tidak terpakai, Anda dapat mengurangi biaya berkelanjutan dengan cara berikut.

Note

Jika Anda telah mengidentifikasi tabel yang tidak terpakai tetapi masih ingin tetap tersedia jika tabel tersebut perlu diakses di masa mendatang, pertimbangkan untuk mengalihkannya ke mode sesuai permintaan. Atau, Anda dapat mempertimbangkan untuk membuat cadangan dan menghapus tabel sepenuhnya.

Mode kapasitas

DynamoDB mengenakan biaya untuk membaca, menulis, dan menyimpan data dalam tabel DynamoDB Anda.

DynamoDB memiliki [dua mode kapasitas](#), yang dilengkapi dengan opsi penagihan khusus untuk memproses baca dan tulis pada tabel Anda: sesuai permintaan dan disediakan. Mode kapasitas baca/tulis mengontrol cara Anda dikenakan biaya untuk throughput baca dan tulis serta cara Anda mengelola kapasitas.

Untuk tabel mode sesuai permintaan, Anda tidak perlu menentukan jumlah throughput baca dan tulis yang Anda harapkan untuk dijalankan oleh aplikasi Anda. DynamoDB membebankan biaya bagi Anda atas baca dan tulis yang dilakukan aplikasi Anda pada tabel Anda dalam hal unit permintaan baca dan unit permintaan tulis. Jika tidak ada aktivitas di tabel/indeks Anda, Anda tidak perlu membayar throughput tetapi Anda masih akan dikenakan biaya penyimpanan.

Kelas tabel

DynamoDB juga menawarkan [dua kelas tabel](#) yang didesain untuk membantu Anda mengoptimalkan biaya. Kelas tabel DynamoDB Standard adalah defaultnya dan direkomendasikan untuk sebagian besar beban kerja. Kelas tabel DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) dioptimalkan untuk tabel yang biayanya didominasi oleh penyimpanan.

Jika tidak ada aktivitas di tabel atau indeks Anda, penyimpanan kemungkinan akan menjadi biaya yang dominan dan mengubah kelas tabel akan menghasilkan penghematan yang signifikan.

Menghapus tabel

Jika telah menemukan tabel yang tidak terpakai dan ingin menghapusnya, sebaiknya Anda mencadangkan atau mengekspor datanya terlebih dahulu.

Pencadangan yang diambil melalui AWS Backup dapat memanfaatkan tiering cold storage, yang selanjutnya mengurangi biaya. Lihat [Menggunakan AWS Backup dengan DynamoDB](#) dokumentasi untuk informasi tentang cara mengaktifkan pencadangan melalui AWS Backup serta dokumentasi [Mengelola rencana cadangan](#) untuk informasi tentang cara menggunakan siklus hidup untuk memindahkan cadangan Anda ke penyimpanan dingin.

Atau, Anda dapat mengekspor data tabel ke S3. Untuk melakukannya, lihat dokumentasi [Mengekspor ke Amazon S3](#). Setelah data Anda diekspor, jika Anda ingin memanfaatkan S3 Glacier Instant Retrieval, S3 Glacier Flexile Retrieval, atau S3 Glacier Deep Archive untuk semakin mengurangi biaya, lihat [Mengelola siklus hidup penyimpanan Anda](#).

Setelah tabel Anda telah dicadangkan, Anda dapat menghapusnya melalui AWS Management Console atau AWS Command Line Interface.

Mengidentifikasi sumber daya GSI yang tidak terpakai

Langkah-langkah untuk mengidentifikasi sekunder global yang tidak terpakai sama dengan langkah-langkah untuk mengidentifikasi tabel yang tidak terpakai. Karena DynamoDB mereplikasi item yang ditulis ke tabel dasar Anda ke dalam GSI jika item tersebut berisi atribut yang digunakan sebagai kunci partisi GSI, GSI yang tidak digunakan kemungkinan masih memiliki `ConsumedWriteCapacityUnits` di atas 0 jika tabel dasarnya digunakan. Dengan demikian, Anda hanya akan mengevaluasi metrik `ConsumedReadCapacityUnits` untuk menentukan apakah GSI Anda tidak terpakai.

Untuk melihat metrik GSI Anda melalui AWS CLI, Anda dapat menggunakan perintah berikut untuk mengevaluasi bacaan tabel Anda:

```
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>
```

Untuk menghindari kesalahan dalam mengidentifikasi tabel sebagai tidak terpakai, sebaiknya Anda mengevaluasi metrik dalam jangka waktu yang lebih lama. Pilih rentang waktu mulai dan waktu berakhir yang sesuai, seperti 30 hari, dan periode yang sesuai, seperti 86400.

Dalam data yang dikembalikan, setiap Jumlah di atas 0 menunjukkan bahwa tabel yang Anda evaluasi menerima lalu lintas baca selama periode tersebut.

Hasil berikut menunjukkan GSI yang menerima lalu lintas baca pada periode yang dievaluasi:

```
{
  "Timestamp": "2022-08-17T21:20:00Z",
  "Sum": 36319167.0,
  "Unit": "Count"
},
{
  "Timestamp": "2022-08-11T21:20:00Z",
  "Sum": 1869136.0,
  "Unit": "Count"
},
```

Hasil berikut menunjukkan GSI yang menerima lalu lintas baca minimal pada periode yang dievaluasi:

```
{
  "Timestamp": "2022-08-28T21:20:00Z",
  "Sum": 0.0,
  "Unit": "Count"
},
{
  "Timestamp": "2022-08-15T21:20:00Z",
  "Sum": 2.0,
  "Unit": "Count"
},
```

Hasil berikut menunjukkan GSI yang tidak menerima lalu lintas baca pada periode yang dievaluasi:

```
{
  "Timestamp": "2022-08-17T21:20:00Z",
  "Sum": 0.0,
  "Unit": "Count"
},
{
  "Timestamp": "2022-08-11T21:20:00Z",
  "Sum": 0.0,
  "Unit": "Count"
},
```

Membersihkan sumber daya GSI yang tidak terpakai

Jika telah mengidentifikasi GSI yang tidak terpakai, Anda dapat menghapusnya. Karena semua data yang ada di GSI juga ada di tabel dasar, cadangan tambahan tidak diperlukan sebelum menghapus GSI. Jika GSI diperlukan lagi di masa mendatang, GSI dapat ditambahkan kembali ke tabel.

Jika telah mengidentifikasi GSI yang jarang digunakan, Anda harus mempertimbangkan perubahan desain di aplikasi Anda yang memungkinkan Anda untuk menghapusnya atau mengurangi biayanya. Misalnya, meskipun pindaian DynamoDB harus digunakan dengan hemat karena pindaian dapat mengonsumsi sejumlah besar sumber daya sistem, pindaian tersebut mungkin lebih hemat biaya daripada GSI jika pola akses yang didukungnya sangat jarang digunakan.

Selain itu, jika GSI diperlukan untuk mendukung pola akses yang jarang, pertimbangkan untuk memproyeksikan serangkaian atribut yang lebih terbatas. Meskipun mungkin memerlukan kueri berikutnya terhadap tabel dasar untuk mendukung pola akses yang jarang, hal ini berpotensi memberikan pengurangan yang signifikan dalam biaya penyimpanan dan penulisan.

Membersihkan tabel global yang tidak terpakai

Tabel global Amazon DynamoDB menyediakan solusi terkelola penuh untuk men-deploy basis data multi-Wilayah dan multi-aktif, tanpa harus membangun dan memelihara solusi replikasi Anda sendiri.

Tabel global ideal untuk menyediakan akses latensi rendah ke data yang dekat dengan pengguna dan juga wilayah sekunder untuk pemulihan bencana.

Jika opsi tabel global diaktifkan untuk sumber daya dalam upaya menyediakan akses latensi rendah ke data tetapi bukan bagian dari strategi pemulihan bencana Anda, validasi bahwa kedua replika secara aktif melayani lalu lintas baca dengan mengevaluasi metriknya. CloudWatch Jika satu replika tidak melayani lalu lintas baca, replika tersebut mungkin merupakan sumber daya yang tidak terpakai.

Jika tabel global adalah bagian dari strategi pemulihan bencana Anda, satu replika yang tidak menerima lalu lintas baca mungkin terjadi dalam pola aktif/siaga.

Membersihkan cadangan atau point-in-time pemulihan yang tidak terpakai (PITR)

DynamoDB menawarkan dua gaya pencadangan. oint-in-time Pemulihan P menyediakan pencadangan berkelanjutan selama 35 hari untuk membantu Anda melindungi dari penulisan atau penghapusan yang tidak disengaja sementara cadangan sesuai permintaan memungkinkan pembuatan snapshot yang dapat disimpan dalam jangka panjang. Kedua jenis pencadangan ini memiliki biaya yang terkait dengannya.

Lihat dokumentasi untuk [Menggunakan cadangan Sesuai Permintaan dan DynamoDB Permintaan](#) dan [oint-in-time Pemulihan P untuk DynamoDB](#) untuk menentukan apakah tabel Anda mengaktifkan cadangan yang mungkin tidak diperlukan lagi.

Evaluasi pola penggunaan tabel Anda

Bagian ini memberikan gambaran umum tentang cara mengevaluasi apakah Anda menggunakan tabel DynamoDB secara efisien. Ada pola penggunaan tertentu yang tidak optimal untuk DynamoDB, dan pola tersebut memberikan ruang untuk pengoptimalan baik dari sudut pandang performa maupun biaya.

Topik

- [Lakukan lebih sedikit operasi bacaan sangat konsisten](#)
- [Lakukan lebih sedikit transaksi untuk operasi baca](#)
- [Lakukan lebih sedikit pemindaian](#)
- [Gunakan nama atribut yang lebih pendek](#)
- [Aktifkan Waktu untuk Tayang \(TTL\)](#)
- [Ganti tabel global dengan cadangan lintas Wilayah](#)

Lakukan lebih sedikit operasi bacaan sangat konsisten

DynamoDB memungkinkan Anda mengonfigurasi [konsistensi baca](#) berdasarkan permintaan. Permintaan baca pada akhirnya konsisten secara default. Bacaan akhir konsisten dikenakan 0,5 RCU untuk data hingga 4 KB.

Sebagian besar beban kerja terdistribusi bersifat fleksibel dan dapat menoleransi konsistensi akhir. Namun, mungkin ada pola akses yang membutuhkan bacaan sangat konsisten. Bacaan sangat

konsisten dikenakan 1 RCU untuk data hingga 4 KB, yang pada dasarnya menggandakan biaya baca Anda. DynamoDB memberi Anda fleksibilitas untuk menggunakan kedua model konsistensi pada tabel yang sama.

Anda dapat mengevaluasi beban kerja dan kode aplikasi untuk mengonfirmasi apakah bacaan sangat konsisten hanya digunakan jika diperlukan.

Lakukan lebih sedikit transaksi untuk operasi baca

DynamoDB memungkinkan Anda untuk mengelompokkan tindakan tertentu dengan cara tertentu, all-or-nothing yang berarti Anda memiliki kemampuan untuk mengeksekusi transaksi ACID dengan DynamoDB. Namun, seperti halnya basis data relasional, pendekatan ini tidak perlu diikuti untuk setiap tindakan.

[Operasi baca transaksional](#) hingga 4 KB menggunakan 2 RCU dibandingkan dengan 0,5 RCU default untuk membaca jumlah data yang sama yang pada akhirnya konsisten. Biaya ini menjadi dua kali lipat dalam operasi tulis, yang berarti tulis transaksional hingga 1 KB setara dengan 2 WCU.

Untuk menentukan apakah semua operasi pada tabel Anda adalah transaksi, CloudWatch metrik untuk tabel dapat disaring ke API transaksi. Jika API transaksi adalah satu-satunya grafik yang tersedia pada metrik `SuccessfulRequestLatency` untuk tabel, hal ini akan mengonfirmasi bahwa setiap operasi adalah transaksi untuk tabel ini. Atau, jika tren penggunaan kapasitas secara keseluruhan cocok dengan tren API transaksi, pertimbangkan untuk mempertahankan desain aplikasi karena tampaknya didominasi oleh API transaksional.

Lakukan lebih sedikit pemindaian

Penggunaan operasi Scan yang ekstensif umumnya berasal dari kebutuhan untuk menjalankan kueri analitik pada data DynamoDB. Menjalankan operasi Scan yang sering pada tabel besar bisa menjadi tidak efisien dan mahal.

Alternatif yang lebih baik adalah dengan menggunakan fitur [Ekspor ke S3](#) dan memilih titik waktu untuk mengekspor status tabel ke S3. AWS menawarkan layanan seperti Athena yang kemudian dapat digunakan untuk menjalankan kueri analitis pada data tanpa menghabiskan kapasitas apa pun dari tabel.

Frekuensi operasi Scan dapat ditentukan menggunakan statistik `SampleCount` di bagian metrik `SuccessfulRequestLatency` untuk Scan. Jika operasi Scan memang sangat sering, pola akses dan model data harus dievaluasi ulang.

Gunakan nama atribut yang lebih pendek

Ukuran total item di DynamoDB adalah jumlah dari panjang dan nilai nama atributnya. Memiliki nama atribut yang panjang tidak hanya berkontribusi terhadap biaya penyimpanan, tetapi juga dapat menyebabkan konsumsi RCU/WCU yang lebih tinggi. Sebaiknya Anda memilih nama atribut yang lebih pendek. Memiliki nama atribut yang lebih pendek dapat membantu membatasi ukuran item dalam batas 4KB/1KB berikutnya setelah itu Anda akan menggunakan RCU/WCU tambahan untuk mengakses data.

Aktifkan Waktu untuk Tayang (TTL)

[Waktu untuk Tayang \(TTL\)](#) dapat mengidentifikasi item yang lebih lama dari waktu kedaluwarsa yang telah Anda tetapkan pada suatu item dan menghapus item tersebut dari tabel. Jika data Anda bertambah seiring waktu dan data lama menjadi tidak relevan, mengaktifkan TTL pada tabel dapat membantu memangkas data Anda dan menghemat biaya penyimpanan.

Aspek lain yang berguna dari TTL adalah bahwa item yang kedaluwarsa terjadi pada aliran DynamoDB Anda, jadi daripada hanya menghapus data dari data Anda, Anda dapat menggunakan item tersebut dari aliran dan mengarsipkannya ke tingkat penyimpanan berbiaya lebih rendah. Selain itu, menghapus item melalui TTL tidak dikenakan biaya tambahan — tidak menggunakan kapasitas, dan tidak ada biaya tambahan untuk merancang aplikasi pembersihan.

Ganti tabel global dengan cadangan lintas Wilayah

[Tabel global](#) memungkinkan Anda mengelola beberapa tabel replika aktif di Wilayah berbeda — semuanya dapat menerima operasi tulis dan mereplikasi data satu sama lain. Sangat mudah untuk mengatur replika dan sinkronisasi dikelola untuk Anda. Replika menyatu ke keadaan konsisten menggunakan strategi penulis terakhir menang.

Jika Anda menggunakan tabel Global semata-mata sebagai bagian dari strategi failover atau pemulihan bencana (DR), Anda dapat menggantinya dengan salinan cadangan lintas Wilayah untuk sasaran titik pemulihan yang relatif ringan dan persyaratan sasaran waktu pemulihan. Jika Anda tidak memerlukan akses lokal yang cepat dan ketersediaan lima sembilan yang tinggi, mempertahankan replika tabel global mungkin bukan pendekatan terbaik untuk failover.

Sebagai alternatif, pertimbangkan untuk menggunakan AWS Backup untuk mengelola backup DynamoDB. Anda dapat menjadwalkan pencadangan rutin dan menyalinnya di seluruh Wilayah untuk memenuhi persyaratan DR dengan pendekatan yang lebih hemat biaya dibandingkan menggunakan tabel Global.

Evaluasi penggunaan Streams

Bagian ini memberikan gambaran umum tentang cara mengevaluasi penggunaan DynamoDB Streams. Ada pola penggunaan tertentu yang tidak optimal untuk DynamoDB, dan pola tersebut memberikan ruang untuk pengoptimalan baik dari sudut pandang performa maupun biaya.

Anda memiliki dua integrasi streaming asli untuk kasus penggunaan berdasarkan peristiwa dan streaming:

- [Amazon DynamoDB Streams](#)
- [Amazon Kinesis Data Streams](#)

Halaman ini hanya akan fokus pada strategi pengoptimalan biaya untuk opsi ini. Jika Anda ingin mengetahui cara memilih di antara kedua opsi tersebut, lihat [Opsi streaming untuk tangkapan data perubahan](#).

Topik

- [Mengoptimalkan biaya untuk DynamoDB Streams](#)
- [Mengoptimalkan biaya untuk Kinesis Data Streams](#)
- [Strategi pengoptimalan biaya untuk kedua jenis penggunaan Streams](#)

Mengoptimalkan biaya untuk DynamoDB Streams

Seperti yang disebutkan di [halaman harga](#) untuk DynamoDB Streams, terlepas dari mode kapasitas throughput tabel, DynamoDB membebankan biaya pada jumlah permintaan baca yang dibuat terhadap DynamoDB Stream tabel. Permintaan baca yang dibuat terhadap DynamoDB Stream berbeda dari permintaan baca yang dibuat terhadap tabel DynamoDB.

Setiap permintaan baca dalam aliran berbentuk panggilan API `GetRecords` yang dapat mengembalikan hingga 1000 catatan atau catatan senilai 1 MB sebagai respons, mana saja yang dicapai terlebih dahulu. Tidak ada [API DynamoDB Stream lainnya](#) yang dikenakan biaya dan DynamoDB Streams tidak dikenakan biaya karena diam. Dengan kata lain, jika tidak ada permintaan baca yang dibuat ke DynamoDB Stream, mengaktifkan DynamoDB Stream pada tabel tidak akan dikenakan biaya.

Berikut adalah beberapa aplikasi konsumen untuk DynamoDB Streams:

- AWS Lambda fungsi (s)

- Aplikasi berbasis Amazon Kinesis Data Streams
- Aplikasi konsumen pelanggan yang dibuat menggunakan AWS SDK

Permintaan baca yang dibuat oleh konsumen DynamoDB Streams AWS berbasis Lambda gratis, sedangkan panggilan yang dilakukan oleh konsumen jenis lain dikenakan biaya. Setiap bulan, 2.500.000 permintaan baca pertama yang dibuat oleh konsumen non-Lambda juga tidak dikenakan biaya. Ini berlaku untuk semua permintaan baca yang dibuat ke DynamoDB Streams apa pun di AWS Akun untuk setiap Wilayah. AWS

Memantau penggunaan DynamoDB Streams

Biaya DynamoDB Streams pada konsol penagihan dikelompokkan bersama untuk semua DynamoDB Streams di seluruh Wilayah dalam Akun. AWS Saat ini, pemberian tag DynamoDB Streams tidak didukung, sehingga tag alokasi biaya tidak dapat digunakan untuk mengidentifikasi biaya mendetail untuk DynamoDB Streams. Volume panggilan `GetRecords` dapat diperoleh di tingkat DynamoDB Stream untuk menghitung biaya per aliran. Volume diwakili oleh `SuccessfulRequestLatency` metrik DynamoDB Stream dan CloudWatch statistiknya. `SampleCount` Metrik ini juga akan mencakup `GetRecords` panggilan yang dibuat oleh tabel global untuk melakukan replikasi yang sedang berlangsung serta panggilan yang dilakukan oleh konsumen AWS Lambda, yang keduanya tidak dikenakan biaya. Untuk informasi tentang CloudWatch metrik lain yang diterbitkan oleh DynamoDB Streams, lihat [Dimensi dan Metrik DynamoDB](#)

Menggunakan AWS Lambda sebagai konsumen

Mengevaluasi jika menggunakan fungsi AWS Lambda sebagai konsumen untuk DynamoDB Streams layak karena dapat menghilangkan biaya yang terkait dengan pembacaan dari DynamoDB Stream. Di sisi lain, aplikasi konsumen berbasis SDK atau Adaptor DynamoDB Streams Kinesis akan dikenakan biaya pada jumlah panggilan `GetRecords` yang dilakukan terhadap DynamoDB Stream.

Invokasi fungsi Lambda akan dikenakan biaya berdasarkan harga Lambda standar, tetapi DynamoDB Streams tidak membebankan biaya apa pun. Lambda akan melakukan polling pada pecahan di DynamoDB Stream Anda untuk catatan dengan tingkat dasar sebanyak 4 kali per detik. Setelah catatan tersedia, Lambda menginvokasi fungsi Anda dan menunggu hasilnya. Jika pemrosesan berhasil, Lambda melanjutkan polling sampai menerima lebih banyak catatan.

Menyetel aplikasi konsumen berbasis Adaptor DynamoDB Streams Kinesis

Karena permintaan baca yang dibuat oleh konsumen berbasis non-Lambda dikenakan biaya untuk DynamoDB Streams, penting untuk menemukan keseimbangan antara persyaratan mendekati waktu nyata dan frekuensi aplikasi konsumen harus melakukan polling terhadap DynamoDB Stream.

Frekuensi polling pada DynamoDB Streams menggunakan aplikasi berbasis Adaptor DynamoDB Streams Kinesis ditentukan oleh konfigurasi nilai `idleTimeBetweenReadsInMillis`. Parameter ini menentukan waktu tunggu konsumen dalam milidetik sebelum memproses pecahan jika panggilan `GetRecords` sebelumnya yang dilakukan ke pecahan yang sama tidak menghasilkan catatan apa pun. Secara default, nilai untuk parameter ini adalah 1000 ms. Jika pemrosesan mendekati waktu nyata tidak diperlukan, parameter ini dapat ditingkatkan agar aplikasi konsumen melakukan lebih sedikit panggilan `GetRecords` dan mengoptimalkan panggilan DynamoDB Streams.

Mengoptimalkan biaya untuk Kinesis Data Streams

Jika Kinesis Data Stream ditetapkan sebagai tujuan untuk mengirimkan peristiwa pengambilan data perubahan untuk tabel DynamoDB, Kinesis Data Stream mungkin memerlukan manajemen ukuran terpisah yang akan memengaruhi biaya keseluruhan. DynamoDB mengenakan biaya dalam hal Change Data capture Unit (CDU) yang setiap unitnya terdiri dari ukuran item DynamoDB sebesar 1 KB yang dicoba oleh layanan DynamoDB ke Kinesis Data Stream tujuan.

Selain biaya oleh layanan DynamoDB, biaya Kinesis Data Stream standar akan dikenakan. Seperti yang disebutkan di [halaman harga](#), harga layanan berbeda berdasarkan mode kapasitas - disediakan dan sesuai permintaan, yang berbeda dari mode kapasitas tabel DynamoDB dan ditentukan pengguna. Pada tingkat tinggi, Kinesis Data Streams membebankan tarif per jam berdasarkan mode kapasitas, serta data yang diserap ke dalam aliran oleh layanan DynamoDB. Mungkin ada biaya tambahan seperti pengambilan data (untuk mode sesuai permintaan), retensi data yang diperpanjang (melebihi default 24 jam), dan peningkatan pengambilan konsumen fan-out tergantung pada konfigurasi pengguna untuk Kinesis Data Stream.

Memantau penggunaan Kinesis Data Streams

Kinesis Data Streams untuk DynamoDB menerbitkan metrik dari DynamoDB selain Metrik Aliran Data Kinesis standar. CloudWatch Ada kemungkinan bahwa Put upaya oleh layanan DynamoDB dibatasi oleh layanan Kinesis karena kapasitas Kinesis Data Streams tidak mencukupi, atau oleh komponen dependen seperti layanan yang dapat dikonfigurasi untuk mengenkripsi data Kinesis Data Stream saat AWS KMS istirahat.

Untuk mempelajari lebih lanjut tentang CloudWatch metrik yang diterbitkan oleh layanan DynamoDB untuk Kinesis Data Stream, lihat. [Memantau pengambilan data perubahan dengan Kinesis Data](#)

[Streams](#) Untuk menghindari biaya tambahan percobaan ulang layanan karena throttling, penting untuk menentukan ukuran Kinesis Data Stream yang tepat dalam Mode yang Disediakan.

Memilih mode kapasitas yang tepat untuk Kinesis Data Streams

Kinesis Data Streams didukung dalam dua mode kapasitas — mode disediakan dan mode sesuai permintaan.

- Jika beban kerja yang melibatkan Kinesis Data Stream memiliki lalu lintas aplikasi yang dapat diprediksi, lalu lintas yang konsisten atau meningkat secara bertahap, atau lalu lintas yang dapat diperkirakan dengan akurat, maka mode yang disediakan untuk Kinesis Data Streams akan cocok dan lebih hemat biaya
- Jika beban kerja terhitung baru, memiliki lalu lintas aplikasi yang tidak dapat diprediksi, atau Anda memilih untuk tidak mengelola kapasitas, maka mode sesuai permintaan untuk Kinesis Data Streams akan cocok dan lebih hemat biaya

Praktik terbaik untuk mengoptimalkan biaya adalah dengan mengevaluasi apakah tabel DynamoDB yang terkait dengan Kinesis Data Stream memiliki pola lalu lintas yang dapat diprediksi yang dapat memanfaatkan mode yang disediakan untuk Kinesis Data Streams. Jika beban kerja baru, Anda dapat menggunakan mode sesuai permintaan untuk Kinesis Data Streams selama beberapa minggu awal, meninjau CloudWatch metrik untuk memahami pola lalu lintas, lalu mengalihkan Stream yang sama ke mode yang disediakan berdasarkan sifat beban kerja. Dalam kasus mode yang disediakan, estimasi jumlah pecahan dapat dilakukan dengan mengikuti pertimbangan manajemen pecahan untuk Kinesis Data Streams.

Evaluasi aplikasi konsumen Anda menggunakan Kinesis Data Streams untuk DynamoDB

Karena Kinesis Data Streams tidak membebankan biaya pada jumlah panggilan GetRecords seperti DynamoDB Streams, aplikasi konsumen dapat membuat panggilan sebanyak mungkin, selama frekuensinya berada di bawah batas throttling untuk GetRecords. Terkait mode sesuai permintaan untuk Kinesis Data Streams, pembacaan data dikenakan biaya per GB. Untuk mode yang disediakan pada Kinesis Data Streams, pembacaan tidak dikenakan biaya jika data berumur kurang dari 7 hari. Dalam kasus fungsi Lambda sebagai konsumen Kinesis Data Streams, Lambda melakukan polling pada setiap pecahan di Kinesis Stream Anda untuk catatan dengan tingkat dasar sebanyak sekali per detik.

Strategi pengoptimalan biaya untuk kedua jenis penggunaan Streams

Pemfilteran acara untuk konsumen AWS Lambda

Pemfilteran peristiwa Lambda memungkinkan Anda untuk membuang peristiwa berdasarkan kriteria filter agar tidak tersedia dalam batch invokasi fungsi Lambda. Ini mengoptimalkan biaya Lambda untuk memproses atau membuang catatan aliran yang tidak diinginkan dalam logika fungsi konsumen. Untuk mempelajari selengkapnya tentang mengonfigurasi pemfilteran peristiwa dan menulis kriteria pemfilteran Anda, lihat [Pemfilteran peristiwa Lambda](#).

Tuning AWS Lambda konsumen

Biaya dapat lebih dioptimalkan dengan menyetel parameter konfigurasi Lambda seperti meningkatkan `BatchSize` untuk memproses lebih banyak per invokasi, mengaktifkan `BisectBatchOnFunctionError` untuk mencegah pemrosesan duplikat (yang menimbulkan biaya tambahan), dan mengatur `MaximumRetryAttempts` agar tidak mengalami terlalu banyak percobaan ulang. Secara default, invokasi Lambda konsumen yang gagal akan dicoba ulang tanpa batas waktu hingga catatan kedaluwarsa dari aliran, yaitu sekitar 24 jam untuk DynamoDB Streams dan dapat dikonfigurasi mulai dari 24 jam hingga 1 tahun untuk Kinesis Data Streams. Opsi konfigurasi Lambda tambahan yang tersedia termasuk yang disebutkan di atas untuk konsumen DynamoDB Stream dapat dilihat di [panduan developer AWS Lambda](#).

Evaluasi kapasitas yang disediakan untuk penyediaan ukuran yang tepat

Bagian ini memberikan gambaran umum tentang cara mengevaluasi apakah Anda memiliki ukuran penyediaan yang tepat pada tabel DynamoDB Anda. Seiring berkembangnya beban kerja Anda, Anda harus memodifikasi prosedur operasional dengan tepat, terutama ketika tabel DynamoDB Anda dikonfigurasi dalam mode yang disediakan dan Anda memiliki risiko untuk menyediakan tabel secara berlebihan atau kurang.

Prosedur yang dijelaskan di bawah memerlukan informasi statistik yang harus diambil dari tabel DynamoDB yang mendukung aplikasi produksi Anda. Untuk memahami perilaku aplikasi Anda, Anda harus menentukan periode waktu yang cukup signifikan untuk mengambil data musiman dari aplikasi Anda. Misalnya, jika aplikasi Anda menunjukkan pola mingguan, menggunakan periode tiga minggu akan memberi Anda cukup ruang untuk menganalisis kebutuhan throughput aplikasi.

Jika Anda tidak tahu harus mulai dari mana, gunakan penggunaan data setidaknya selama satu bulan untuk penghitungan di bawah ini.

Saat mengevaluasi kapasitas, tabel DynamoDB dapat mengonfigurasi Unit Kapasitas Baca (RCU) dan Unit Kapasitas Tulis (WCU) secara terpisah. Jika tabel Anda memiliki Indeks Sekunder Global (GSI) yang dikonfigurasi, Anda perlu menentukan throughput yang akan digunakannya, yang juga tidak bergantung pada RCU dan WCU dari tabel dasar.

Note

Indeks Sekunder Lokal (LSI) menggunakan kapasitas dari tabel dasar.

Topik

- [Cara mengambil metrik penggunaan pada tabel DynamoDB Anda](#)
- [Cara mengidentifikasi tabel DynamoDB yang kurang tersedia](#)
- [Cara mengidentifikasi tabel DynamoDB yang disediakan secara berlebihan](#)

Cara mengambil metrik penggunaan pada tabel DynamoDB Anda

Untuk mengevaluasi tabel dan kapasitas GSI, pantau CloudWatch metrik berikut dan pilih dimensi yang sesuai untuk mengambil informasi tabel atau GSI:

Unit Kapasitas Baca	Unit Kapasitas Tulis
ConsumedReadCapacityUnits	ConsumedWriteCapacityUnits
ProvisionedReadCapacityUnits	ProvisionedWriteCapacityUnits
ReadThrottleEvents	WriteThrottleEvents

Anda dapat melakukan ini baik melalui AWS CLI atau AWS Management Console.

AWS CLI

Sebelum kita mengambil metrik konsumsi tabel, kita harus mulai dengan menangkap beberapa titik data historis menggunakan API. CloudWatch

Mulailah dengan membuat dua file: `write-calc.json` dan `read-calc.json`. File-file ini akan mewakili penghitungan untuk tabel atau GSI. Anda harus memperbarui beberapa bidang, seperti yang ditunjukkan pada tabel di bawah ini, agar sesuai dengan lingkungan Anda.

Nama Bidang	Definisi	Contoh
<table-name>	Nama tabel yang akan dianalisis	SampleTable
<period>	Jangka waktu yang akan Anda gunakan untuk mengevaluasi target pemanfaatan, berdasarkan detik	Untuk periode 1 jam, Anda harus menentukan: 3600
<start-time>	Awal interval evaluasi Anda, ditentukan dalam format ISO8601	2022-02-21T23:00:00
<end-time>	Akhir interval evaluasi Anda, ditentukan dalam format ISO8601	2022-02-22T06:00:00

File penghitungan tulis akan mengambil jumlah WCU yang disediakan dan terpakai dalam rentang tanggal tertentu. Ini juga akan menghasilkan persentase penggunaan yang akan digunakan untuk analisis. Konten lengkap file `write-calc.json` akan seperti berikut:

```
{
  "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
}
```

File penghitungan baca menggunakan file serupa. File ini akan mengambil berapa banyak RCU yang disediakan dan terpakai selama jangka waktu untuk rentang tanggal tertentu. Konten file `read-calc.json` akan seperti berikut:

```
{
  "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
}
```

Setelah membuat file, Anda dapat mulai mengambil data penggunaan.

1. Untuk mengambil data penggunaan tulis, keluarkan perintah berikut:

```
aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
```

2. Untuk mengambil data penggunaan baca, keluarkan perintah berikut:

```
aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
```

Hasil untuk kedua kueri akan menjadi serangkaian titik data dalam format JSON yang akan digunakan untuk analisis. Hasil Anda akan tergantung pada jumlah titik data yang Anda tentukan, periode, dan data beban kerja spesifik Anda sendiri. Hasilnya akan seperti berikut:

```
{
  "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

Jika menentukan jangka waktu pendek dan rentang waktu yang lama, Anda mungkin perlu memodifikasi `MaxDatapoints` yang secara default ditetapkan ke 24 dalam skrip. Ini mewakili satu titik data per jam dan 24 per hari.

AWS Management Console

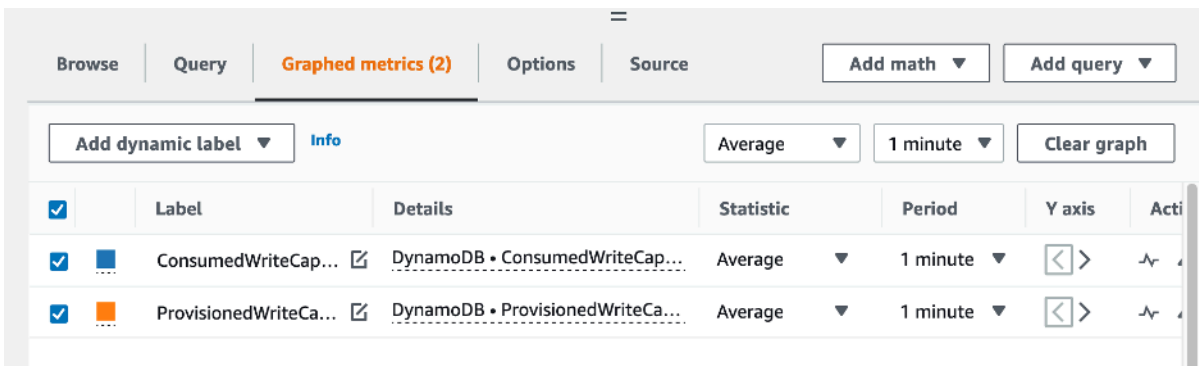
1. Masuk ke AWS Management Console dan navigasikan ke halaman CloudWatch layanan di [Memulai dengan AWS Management Console](#). Pilih AWS wilayah yang sesuai jika perlu.
2. Temukan bagian Metrik di bilah navigasi kiri, lalu pilih Semua metrik.
3. Ini akan membuka dasbor dengan dua panel. Panel atas akan menampilkan grafiknya, dan panel bawah akan menampilkan metrik yang ingin Anda buat grafiknya. Pilih panel DynamoDB.
4. Pilih kategori Metrik Tabel dari sub panel. Ini akan menampilkan tabel di Wilayah Anda saat ini.

- Identifikasi nama tabel Anda dengan menggulir menu ke bawah, lalu memilih metrik operasi tulis: `ConsumedWriteCapacityUnits` dan `ProvisionedWriteCapacityUnits`

Note

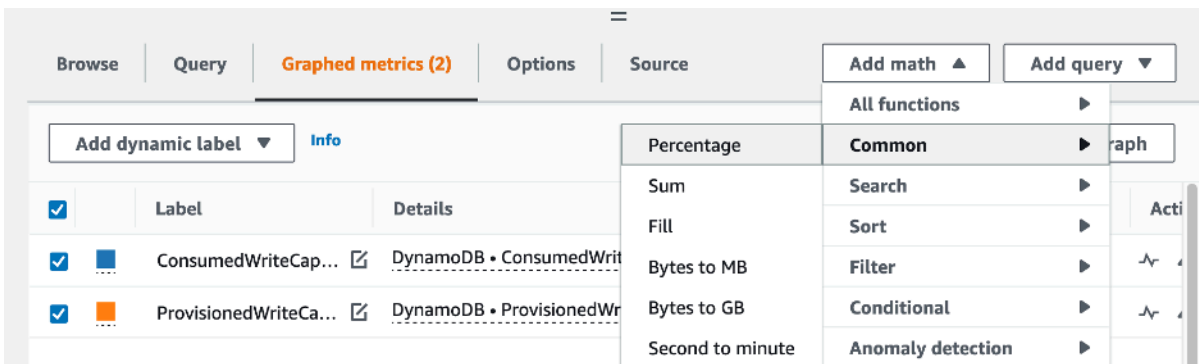
Contoh ini membahas metrik operasi tulis, tetapi Anda juga dapat menggunakan langkah-langkah ini untuk membuat grafik metrik operasi baca.

- Pilih tab Metrik bergrafik (2) untuk memodifikasi rumus. Secara default CloudWatch akan memilih fungsi statistik Rata-rata untuk grafik.



- Saat memilih kedua metrik bergrafik (kotak centang di sebelah kiri) pilih menu Tambahkan perhitungan, diikuti oleh Umum, lalu pilih fungsi Persentase. Ulangi prosedur ini dua kali.

Pertama kali memilih fungsi Persentase:



Kedua kali memilih fungsi Persentase:

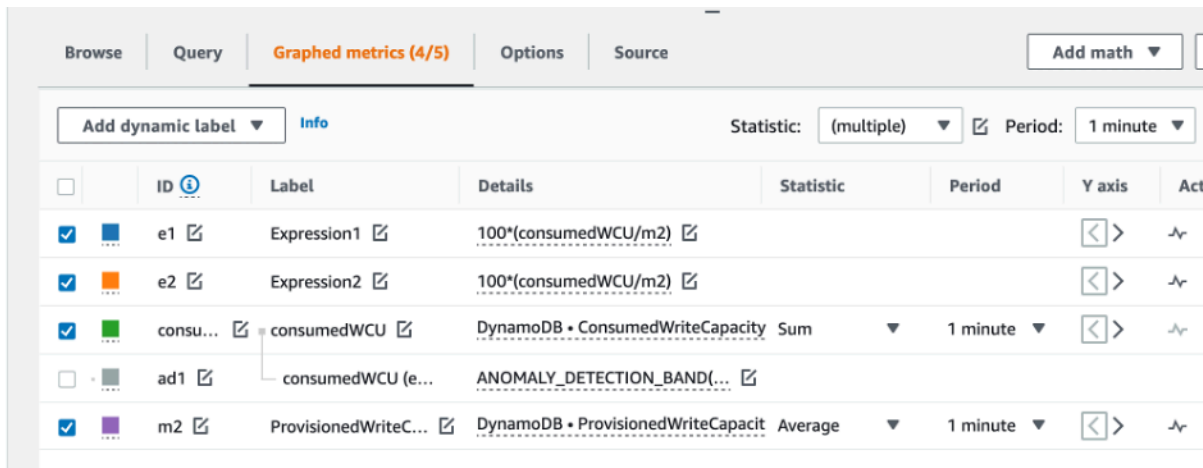
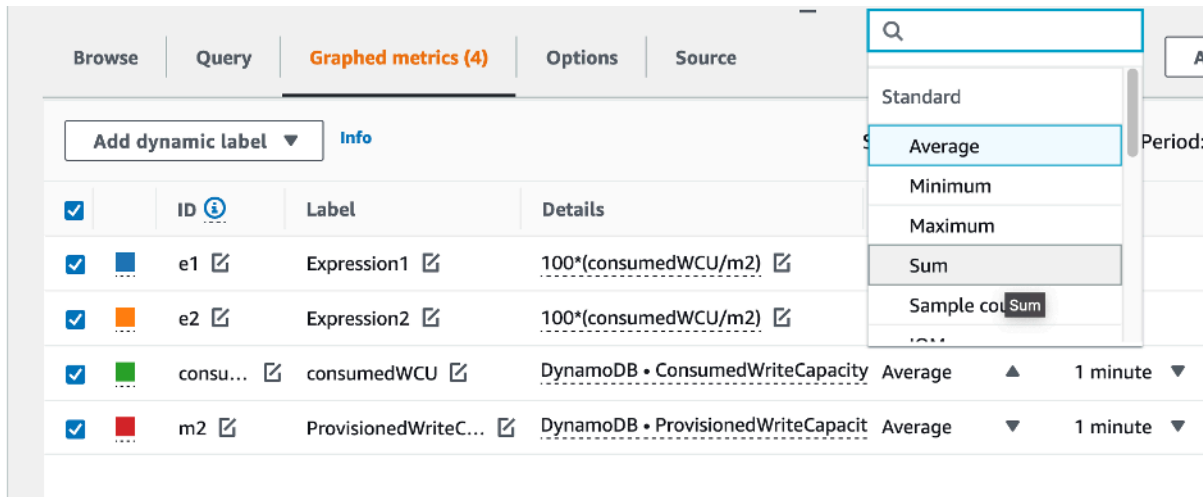
ID	Label	Details
e1	Expression1	$100*(m1/m2)$
m1	ConsumedWri...	DynamoDB • Consume
m2	ProvisionedWri...	DynamoDB • Provision

8. Pada titik ini, Anda memiliki empat metrik di menu bawah. Mari kita kerjakan penghitungan ConsumedWriteCapacityUnits. Agar konsisten, kita perlu mencocokkan nama untuk yang kita gunakan di AWS CLI bagian ini. Klik m1 ID dan ubah nilai ini menjadi consumedWCU.

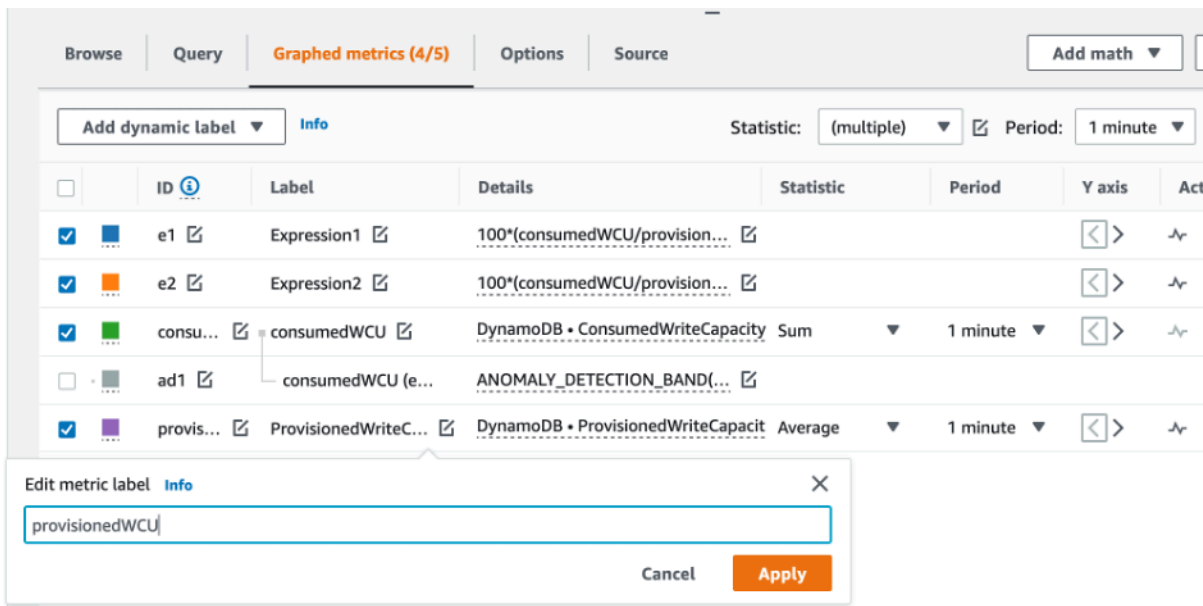
ID	Label	Details	Statistic	Period
e1	Expression1	$100*(m1/m2)$		
e2	Expression2	$100*(m1/m2)$		
m1	ConsumedWriteC...	DynamoDB • ConsumedWriteCapacity	Average	1 minute
m2	ProvisionedWri...	DynamoDB • ProvisionedWriteCapacity	Average	1 minute

ID	Label	Details	Statistic	Period
e1	Expression1	$100*(consumedWCU/m2)$		
e2	Expression2	$100*(consumedWCU/m2)$		
consumedWCU	ConsumedWriteC...	DynamoDB • ConsumedWriteCapacity	Sum	1 minute

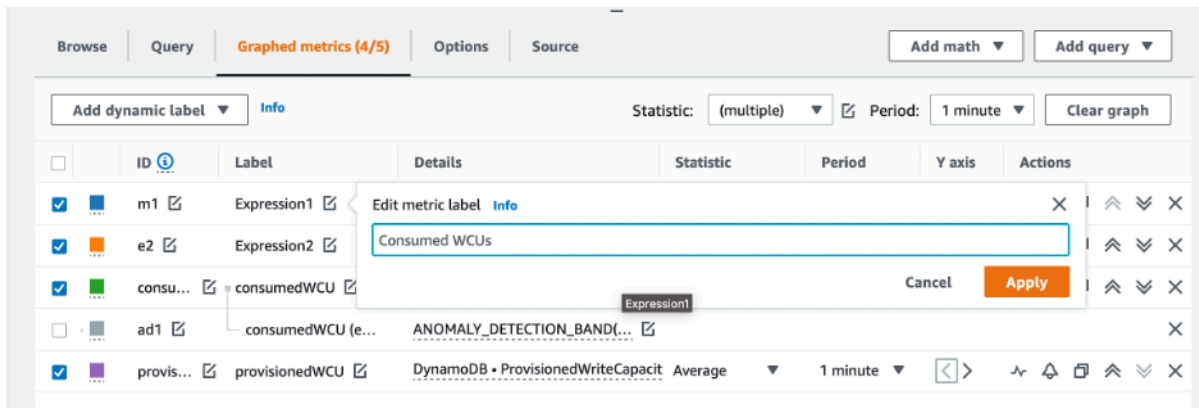
9. Ubah statistik dari Average menjadi Sum. Tindakan ini secara otomatis akan membuat metrik lain yang disebut ANOMALY_DETECTION_BAND. Untuk cakupan prosedur ini, kita dapat mengabaikannya dengan menghapus centang pada ad1 metrik yang baru dibuat.



- Ulangi langkah 8 untuk mengganti nama m2 ID menjadi ProvisionedWCU. Biarkan statistik diatur ke Average.

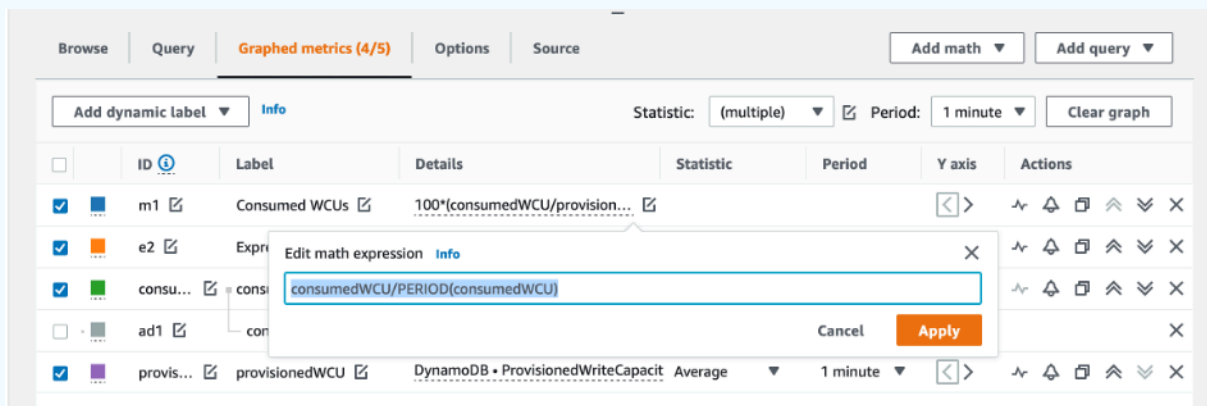


- Pilih label Expression1 lalu perbarui nilainya menjadi m1 dan labelnya menjadi Consumed WCU.

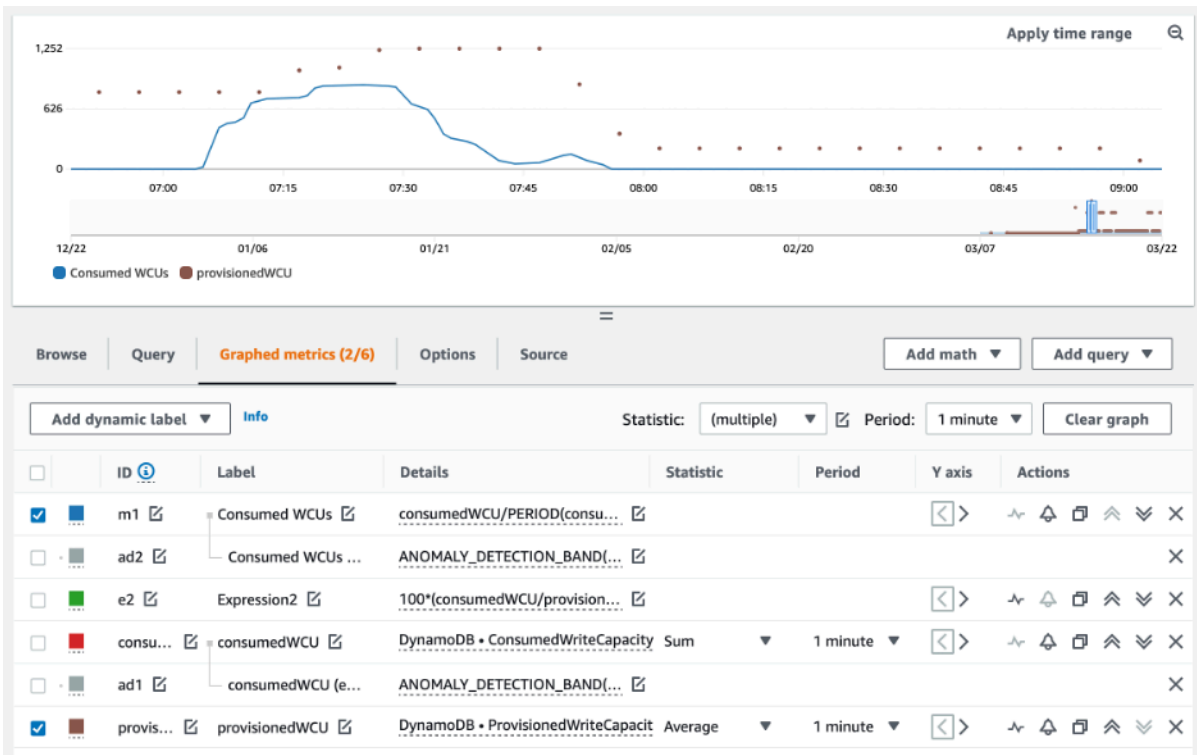


Note

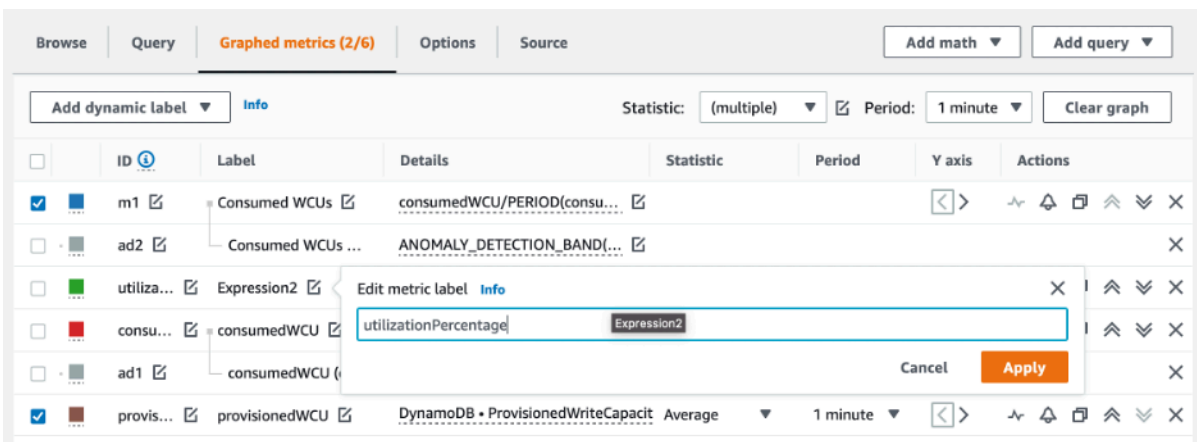
Pastikan Anda hanya memilih m1 (kotak centang di sebelah kiri) dan ProvisionedWCU untuk memvisualisasikan data dengan benar. Perbarui rumus dengan mengklik Detail dan mengubah rumus menjadi `consumedWCU/PERIOD(consumedWCU)`. Langkah ini mungkin juga menghasilkan metrik ANOMALY_DETECTION_BAND, tetapi kita dapat mengabaikannya untuk cakupan prosedur ini.



- Anda sekarang memiliki dua grafik: satu yang menunjukkan WCU yang disediakan pada tabel dan satu lagi yang menunjukkan WCU yang digunakan. Bentuk grafiknya mungkin berbeda dengan gambar di bawah ini, tetapi Anda dapat menggunakannya sebagai referensi:



- Perbarui rumus persentase dengan memilih grafik Expression2 (e2). Ganti nama label dan ID menjadi utilizationPercentage. Ganti nama rumus agar sesuai dengan $100 * (m1 / \text{provisionedWCU})$.

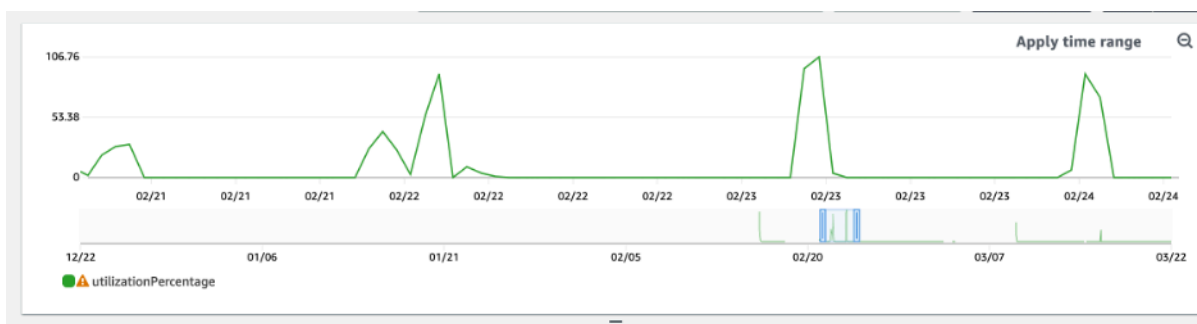


The screenshot shows the Amazon CloudWatch console interface. At the top, there are tabs for 'Browse', 'Query', 'Graphed metrics (2/6)', 'Options', and 'Source'. Below these, there are controls for 'Add dynamic label', 'Statistic: (multiple)', 'Period: 1 minute', and 'Clear graph'. A table lists several metrics with columns for 'ID', 'Label', 'Details', 'Statistic', 'Period', 'Y axis', and 'Actions'. An 'Edit math expression' dialog box is overlaid on the table, showing the formula $100 * (m1 / \text{provisionedWCU})$. The dialog has 'Cancel' and 'Apply' buttons.

- Hapus centang dari semua metrik kecuali utilizationPercentage untuk memvisualisasikan pola penggunaan Anda. Interval default diatur ke 1 menit, tetapi Anda dapat mengubahnya sesuai kebutuhan.



Berikut adalah tampilan periode waktu yang lebih lama serta periode yang lebih dari 1 jam. Anda dapat melihat ada beberapa interval yang penggunaannya lebih tinggi dari 100%, tetapi beban kerja khusus ini memiliki interval yang lebih panjang dengan penggunaan nol.



Pada titik ini, Anda mungkin mendapatkan hasil yang berbeda dari gambar dalam contoh ini. Itu semua tergantung pada data dari beban kerja Anda. Interval dengan penggunaan lebih dari 100% rentan terhadap peristiwa throttling. DynamoDB menawarkan [kapasitas lonjakan](#), tetapi segera setelah kapasitas lonjakan dilakukan, apa pun di atas 100% akan dibatasi.

Cara mengidentifikasi tabel DynamoDB yang kurang tersedia

Untuk sebagian besar beban kerja, tabel dianggap kurang tersedia jika terus-menerus menggunakan lebih dari 80% kapasitas yang disediakan.

[Kapasitas lonjakan](#) adalah fitur DynamoDB yang memungkinkan pelanggan menggunakan lebih banyak RCU/WCU daripada yang disediakan sebelumnya (lebih dari throughput yang disediakan per detik yang ditentukan dalam tabel) untuk sementara waktu. Kapasitas lonjakan diciptakan untuk menyerap peningkatan lalu lintas tiba-tiba karena peristiwa khusus atau lonjakan penggunaan. Kapasitas lonjakan ini tidak bertahan selamanya. Segera setelah RCU dan WCU yang tidak terpakai habis, Anda akan mengalami throttling jika mencoba menggunakan lebih banyak kapasitas daripada yang disediakan. Ketika lalu lintas aplikasi Anda mendekati tingkat penggunaan 80%, risiko throttling Anda jauh lebih tinggi.

Aturan tingkat penggunaan 80% bervariasi berdasarkan musim data dan pertumbuhan lalu lintas Anda. Pertimbangkan skenario berikut:

- Jika lalu lintas Anda stabil pada tingkat penggunaan ~90% selama 12 bulan terakhir, tabel Anda memiliki kapasitas yang tepat
- Jika lalu lintas aplikasi Anda tumbuh sebesar 8% setiap bulan dalam waktu kurang dari 3 bulan, Anda akan mencapai 100%
- Jika lalu lintas aplikasi Anda tumbuh sebesar 5% dalam waktu lebih dari 4 bulan, Anda masih akan mencapai 100%

Hasil dari kueri di atas memberikan gambaran tingkat penggunaan Anda. Gunakan hasil tersebut sebagai panduan untuk mengevaluasi lebih lanjut metrik lain yang dapat membantu Anda meningkatkan kapasitas tabel sesuai kebutuhan (misalnya: tingkat pertumbuhan bulanan atau mingguan). Bekerjalah dengan tim operasi Anda untuk menentukan persentase yang baik untuk beban kerja dan tabel Anda.

Ada skenario khusus, datanya miring ketika dianalisis setiap hari atau setiap minggu. Misalnya, dengan aplikasi musiman yang memiliki lonjakan penggunaan selama jam kerja (tetapi kemudian turun menjadi hampir nol di luar jam kerja), Anda bisa mendapatkan keuntungan dengan [menjadwalkan penskalaan otomatis](#) di mana Anda menentukan jam dalam sehari (dan hari dalam seminggu) untuk meningkatkan kapasitas yang disediakan dan kapan harus mengurangnya. Alih-alih bertujuan untuk kapasitas yang lebih tinggi sehingga Anda dapat menutupi jam sibuk, Anda juga dapat memanfaatkan konfigurasi penskalaan [otomatis tabel DynamoDB](#) jika musim Anda kurang terasa.

Note

Saat membuat konfigurasi penskalaan otomatis DynamoDB untuk tabel dasar Anda, jangan lupa menyertakan konfigurasi lain untuk GSI yang terkait dengan tabel tersebut.

Cara mengidentifikasi tabel DynamoDB yang disediakan secara berlebihan

Hasil kueri yang diperoleh dari skrip di atas memberikan titik data yang diperlukan untuk melakukan beberapa analisis awal. Jika set data Anda menyajikan nilai penggunaan yang lebih rendah dari 20% untuk beberapa interval, tabel Anda mungkin disediakan secara berlebihan. Untuk menentukan lebih lanjut apakah Anda perlu mengurangi jumlah WCU dan RCU, Anda harus meninjau kembali pembacaan lain dalam interval tersebut.

Ketika tabel Anda berisi beberapa interval penggunaan rendah, Anda benar-benar dapat mengambil manfaat dari menggunakan kebijakan penskalaan otomatis, baik dengan menjadwalkan penskalaan otomatis atau hanya mengonfigurasi kebijakan penskalaan otomatis default untuk tabel yang didasarkan pada pemanfaatan.

Jika Anda memiliki beban kerja dengan pemanfaatan rendah terhadap rasio throttle tinggi (Max (ThrottleEvents) /Min () dalam intervalThrottleEvents), ini bisa terjadi ketika Anda memiliki beban kerja yang sangat runcing di mana lalu lintas meningkat banyak selama beberapa hari (atau jam), tetapi secara umum lalu lintas secara konsisten rendah. Dalam skenario ini, mungkin bermanfaat untuk menggunakan [penskalaan otomatis terjadwal](#).

AWS [Well-Architected](#) Framework membantu arsitek cloud membangun infrastruktur yang aman, berkinerja tinggi, tangguh, dan efisien untuk berbagai aplikasi dan beban kerja. Dibangun di sekitar enam pilar—keunggulan operasional, keamanan, keandalan, efisiensi kinerja, optimalisasi biaya, dan keberlanjutan—AWS Well-Architected memberikan pendekatan yang konsisten bagi pelanggan dan mitra untuk mengevaluasi arsitektur dan menerapkan desain yang dapat diskalakan.

AWS [Well-Architected](#) Lenses memperluas panduan yang ditawarkan oleh AWS Well-Architected untuk domain industri dan teknologi tertentu. Amazon DynamoDB Well-Architected Lens berfokus pada beban kerja DynamoDB. Ini memberikan praktik terbaik, prinsip desain, dan pertanyaan untuk menilai dan meninjau beban kerja DynamoDB. Menyelesaikan tinjauan Amazon DynamoDB Well-Architected Lens akan memberi Anda edukasi dan panduan seputar prinsip-prinsip desain yang direkomendasikan yang berkaitan dengan masing-masing pilar AWS Well-Architected. Panduan ini didasarkan pada pengalaman kami bekerja dengan pelanggan di berbagai industri, segmen, ukuran, dan geografi.

Sebagai hasil langsung dari tinjauan Well-Architected Lens, Anda akan menerima ringkasan rekomendasi yang dapat ditindaklanjuti untuk mengoptimalkan dan meningkatkan beban kerja DynamoDB Anda.

Melakukan tinjauan Amazon DynamoDB Well-Architected Lens

Ulasan DynamoDB Well-Architected Lens biasanya dilakukan AWS oleh Arsitek Solusi bersama dengan pelanggan, tetapi juga dapat dilakukan oleh pelanggan sebagai layanan mandiri. Meskipun kami merekomendasikan untuk meninjau keenam Pilar yang Dirancang dengan Baik sebagai bagian dari Amazon DynamoDB Well-Architected Lens, Anda juga dapat memutuskan untuk memprioritaskan fokus Anda pada satu atau beberapa pilar terlebih dahulu.

[Informasi dan instruksi tambahan untuk melakukan tinjauan Amazon DynamoDB Well-Architected Lens tersedia di video ini dan halaman DynamoDB Well-Architected Lens. GitHub](#)

Pilar Amazon DynamoDB Well-Architected Lens

Amazon DynamoDB Well-Architected Lens dibangun dengan enam pilar:

Pilar efisiensi performa

Pilar efisiensi performa mencakup kemampuan untuk menggunakan sumber daya komputasi secara efisien untuk memenuhi persyaratan sistem, dan untuk mempertahankan efisiensi tersebut seiring perubahan permintaan dan perkembangan teknologi.

Prinsip desain DynamoDB yang utama untuk pilar ini berkisar pada [pemodelan data](#), [pemilihan kunci partisi](#) dan [kunci urutan](#), serta [penentuan indeks sekunder](#) berdasarkan pola akses aplikasi. Pertimbangan tambahan termasuk memilih mode throughput optimal untuk beban kerja, penyetelan AWS SDK dan, bila sesuai, menggunakan strategi caching yang optimal. Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton [video mendalam](#) tentang pilar efisiensi performa DynamoDB Well-Architected Lens.

Pilar optimasi biaya

Pilar optimasi biaya berfokus menghindari biaya yang tidak perlu.

Topik utama yang dibahas meliputi pemahaman dan pengendalian pembelanjaan uang, pemilihan jenis sumber daya yang paling sesuai dan tepat, analisis pengeluaran dari waktu ke waktu, merancang model data untuk mengoptimalkan biaya atas pola akses khusus aplikasi, dan melakukan penskalaan untuk memenuhi kebutuhan bisnis tanpa pengeluaran berlebihan.

Prinsip desain pengoptimalan biaya utama untuk DynamoDB berkisar pada pemilihan mode kapasitas dan kelas tabel yang paling sesuai untuk tabel dan menghindari penyediaan kapasitas yang berlebihan dengan menggunakan mode kapasitas sesuai permintaan, atau mode kapasitas yang disediakan dengan penskalaan otomatis. Pertimbangan tambahan termasuk pemodelan data yang efisien dan kueri untuk mengurangi jumlah kapasitas yang dikonsumsi, memesan porsi kapasitas yang dikonsumsi dengan harga diskon, meminimalkan ukuran barang, mengidentifikasi dan menghapus sumber daya yang tidak digunakan dan menggunakan [TTL](#) untuk secara otomatis menghapus data yang sudah tua tanpa biaya. Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton [video mendalam](#) tentang pilar pengoptimalan biaya DynamoDB Well-Architected Lens.

Lihat [Optimasi biaya](#) untuk informasi tambahan tentang praktik terbaik pengoptimalan biaya untuk DynamoDB.

Pilar keunggulan operasional

Pilar keunggulan operasional berfokus menjalankan dan memantau sistem guna memberikan nilai bisnis, dan terus meningkatkan proses dan prosedur. Topik utama yang dibahas meliputi mengotomatisasi perubahan, merespons peristiwa, dan mendefinisikan standar untuk mengelola operasi harian.

Prinsip desain keunggulan operasional utama untuk DynamoDB meliputi pemantauan metrik DynamoDB melalui CloudWatch AWS Config Amazon dan secara otomatis memperingatkan dan memulihkan ketika ambang batas yang telah ditentukan dilanggar, atau aturan yang tidak sesuai terdeteksi. Pertimbangan tambahan adalah mendefinisikan sumber daya DynamoDB melalui infrastruktur sebagai kode dan memanfaatkan tag untuk pengorganisasian, identifikasi, dan penghitungan biaya sumber daya DynamoDB yang lebih baik. Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton [video mendalam](#) tentang pilar keunggulan operasional DynamoDB Well-Architected Lens.

Pilar keandalan

Pilar keandalan berfokus memastikan beban kerja menjalankan fungsinya dengan benar dan konsisten sesuai harapan. Beban kerja yang tangguh dapat pulih dengan cepat dari kegagalan memenuhi permintaan bisnis dan pelanggan. Topik utama yang dibahas meliputi desain sistem terdistribusi, perencanaan pemulihan, dan cara menangani perubahan.

Prinsip desain keandalan penting untuk DynamoDB berkisar pada pemilihan strategi pencadangan dan retensi berdasarkan persyaratan RPO dan RTO Anda, menggunakan tabel global DynamoDB

untuk beban kerja multi-regional, atau skenario pemulihan bencana lintas wilayah dengan RTO rendah, menerapkan logika coba lagi dengan dukungan eksponensial dalam aplikasi dengan mengonfigurasi dan menggunakan kemampuan ini di SDK, dan memantau metrik DynamoDB melalui Amazon dan secara otomatis memperingatkan dan memulihkan ketika ambang batas yang telah ditentukan dilanggar. AWS CloudWatch Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton [video mendalam](#) tentang pilar keandalan DynamoDB Well-Architected Lens.

Pilar keamanan

Pilar keamanan berfokus pada perlindungan informasi dan sistem. Topik utama yang dibahas meliputi kerahasiaan dan integritas data, mengidentifikasi dan mengelola “siapa yang dapat melakukan apa” dengan manajemen hak istimewa, melindungi sistem, dan menetapkan kontrol untuk mendeteksi peristiwa keamanan.

Prinsip desain keamanan utama untuk DynamoDB adalah mengenkripsi data bergerak dengan HTTPS, memilih jenis kunci untuk enkripsi data diam, serta menentukan kebijakan dan peran IAM untuk mengautentikasi, mengotorisasi, dan memberikan akses mendetail ke sumber daya DynamoDB. Pertimbangan tambahan termasuk mengaudit bidang kontrol DynamoDB dan operasi pesawat data melalui AWS CloudTrail Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton [video mendalam](#) tentang pilar keamanan DynamoDB Well-Architected Lens.

Lihat [Keamanan](#) untuk informasi tambahan tentang keamanan untuk DynamoDB.

Pilar keberlanjutan

Pilar keberlanjutan berfokus meminimalkan dampak lingkungan dari menjalankan beban kerja cloud. Topik utama yang dibahas mencakup model tanggung jawab bersama untuk keberlanjutan, pemahaman terhadap dampak, dan memaksimalkan penggunaan untuk meminimalkan sumber daya yang dibutuhkan dan mengurangi dampak hilir.

Prinsip desain keberlanjutan utama untuk DynamoDB mencakup mengidentifikasi dan menghapus sumber daya DynamoDB yang tidak terpakai, menghindari penyediaan berlebihan melalui penggunaan mode kapasitas sesuai permintaan atau mode kapasitas yang disediakan dengan penskalaan otomatis, kueri yang efisien untuk mengurangi jumlah kapasitas yang digunakan dan pengurangan jejak penyimpanan dengan mengompresi data dan menghapus data lama melalui penggunaan TTL. Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton [video mendalam](#) tentang pilar keberlanjutan DynamoDB Well-Architected Lens.

Praktik Terbaik untuk merancang dan menggunakan kunci partisi secara efektif

Kunci primer yang secara unik mengidentifikasi setiap item dalam tabel Amazon DynamoDB bisa sederhana (hanya kunci partisi) atau komposit (kunci partisi dikombinasikan dengan kunci urutan).

Secara umum, Anda harus merancang aplikasi Anda untuk aktivitas seragam di semua kunci partisi logis dalam tabel dan indeks sekundernya. Anda dapat menentukan pola akses yang diperlukan aplikasi Anda, serta unit baca dan tulis yang diperlukan setiap tabel dan indeks sekunder.

Secara default, setiap partisi dalam tabel akan berusaha memberikan kapasitas penuh 3.000 RCU dan 1.000 WCU. Total throughput di seluruh partisi dalam tabel mungkin dibatasi oleh throughput yang ditentukan dalam mode yang disediakan, atau oleh batas throughput tingkat tabel dalam mode sesuai permintaan. Lihat [Kuota Layanan](#) untuk informasi selengkapnya.

Topik

- [Menggunakan kapasitas lonjakan secara efektif](#)
- [Memahami kapasitas adaptif DynamoDB](#)
- [Merancang kunci partisi untuk mendistribusikan beban kerja Anda](#)
- [Menggunakan pembagian tulis untuk mendistribusikan beban kerja secara merata](#)
- [Mendistribusikan aktivitas tulis secara efisien selama pengunggahan data](#)

Menggunakan kapasitas lonjakan secara efektif

DynamoDB memberikan beberapa fleksibilitas untuk penyediaan throughput Anda dengan kapasitas lonjakan. Setiap kali Anda tidak sepenuhnya menggunakan throughput yang tersedia, DynamoDB mencadangkan sebagian dari kapasitas yang tidak terpakai tersebut untuk lonjakan throughput berikutnya guna menangani lonjakan penggunaan.

DynamoDB saat ini mempertahankan hingga 5 menit (300 detik) kapasitas baca dan tulis yang tidak terpakai. Selama aktivitas baca atau tulis yang terjadi sesekali, unit kapasitas tambahan ini dapat digunakan dengan cepat—bahkan lebih cepat daripada kapasitas throughput yang disediakan per detik yang telah Anda tetapkan untuk tabel Anda.

DynamoDB juga dapat menggunakan kapasitas lonjakan untuk pemeliharaan di latar belakang dan tugas lain tanpa pemberitahuan sebelumnya.

Perhatikan bahwa detail kapasitas lonjakan mungkin berubah di masa mendatang.

Memahami kapasitas adaptif DynamoDB

Kapasitas adaptif adalah fitur yang memungkinkan DynamoDB untuk menjalankan beban kerja yang tidak seimbang tanpa batas waktu. Fitur ini meminimalkan throttling karena pengecualian throughput. Fitur ini juga membantu mengurangi biaya dengan memungkinkan Anda untuk hanya menyediakan kapasitas throughput yang Anda butuhkan.

Kapasitas adaptif diaktifkan secara otomatis untuk setiap tabel DynamoDB, tanpa biaya tambahan. Anda tidak perlu mengaktifkan atau menonaktifkannya secara eksplisit.

Topik

- [Meningkatkan kapasitas throughput ke partisi lalu lintas tinggi](#)
- [Mengisolasi item yang sering diakses](#)

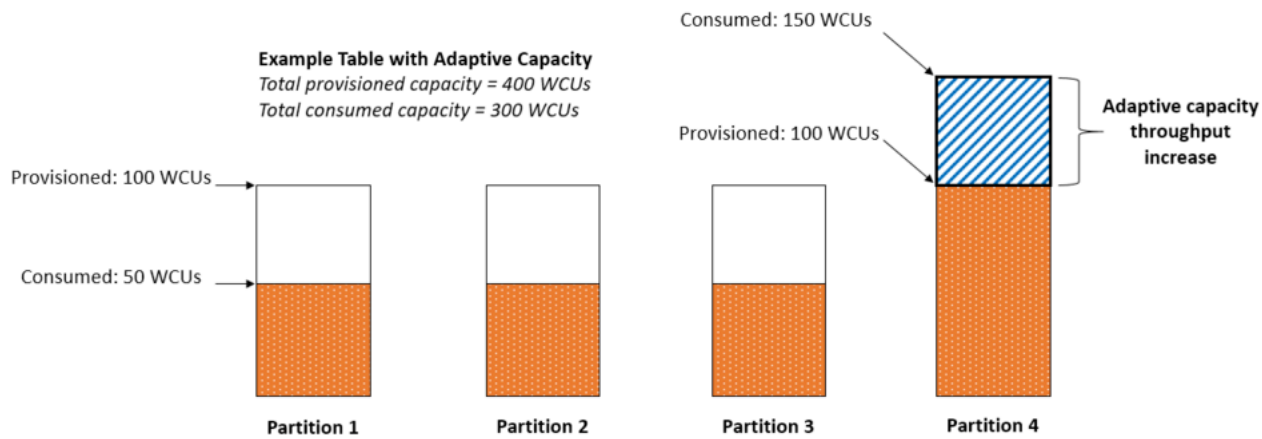
Meningkatkan kapasitas throughput ke partisi lalu lintas tinggi

Mendistribusikan aktivitas baca dan tulis secara merata tidak selalu dapat dilakukan. Ketika akses data tidak seimbang, partisi "panas" dapat menerima volume lalu lintas baca dan tulis yang lebih tinggi daripada partisi lain. Karena operasi baca dan tulis pada suatu partisi dikelola secara terpisah, throttling akan terjadi jika satu partisi menerima lebih dari 3000 operasi baca atau lebih dari 1000 operasi tulis.

Untuk lebih mengakomodasi pola akses yang tidak merata, kapasitas adaptif DynamoDB memungkinkan aplikasi Anda untuk terus membaca dan menulis ke partisi panas tanpa mengalami throttling, selama lalu lintas tidak melebihi total kapasitas yang disediakan tabel atau kapasitas maksimum partisi. Kapasitas adaptif meningkatkan kapasitas throughput secara otomatis dan langsung untuk partisi yang menerima lebih banyak lalu lintas.

Diagram berikut menggambarkan cara kerja kapasitas adaptif. Tabel contoh disediakan dengan 400 WCU yang dibagikan secara merata ke empat partisi, sehingga setiap partisi dapat mempertahankan hingga 100 WCU per detik. Partisi 1, 2, dan 3 masing-masing menerima lalu lintas tulis 50 WCU/detik. Partisi 4 menerima 150 WCU/detik. Partisi panas ini dapat menerima lalu lintas tulis meskipun masih memiliki kapasitas lonjakan yang tidak terpakai, tetapi akhirnya partisi ini melambatkan lalu lintas yang melebihi 100 WCU/detik.

Kapasitas adaptif DynamoDB merespons dengan meningkatkan kapasitas partisi 4 sehingga dapat mempertahankan beban kerja yang lebih tinggi sebesar 150 WCU/detik tanpa mengalami throttling.



Mengisolasi item yang sering diakses

Jika aplikasi Anda mendorong lalu lintas tinggi secara tidak proporsional ke satu atau beberapa item, kapasitas adaptif akan menyeimbangkan kembali partisi Anda sehingga item yang sering diakses tidak berada di partisi yang sama. Isolasi item yang sering diakses ini mengurangi kemungkinan throttling permintaan karena beban kerja Anda melebihi kuota throughput pada satu partisi. Anda juga dapat memecah koleksi item menjadi beberapa segmen berdasarkan kunci urutan, selama koleksi item tersebut bukan lalu lintas yang dilacak oleh peningkatan atau penurunan kunci urutan yang monoton.

Jika aplikasi Anda terus-menerus mendorong lalu lintas yang tinggi ke satu item, kapasitas adaptif mungkin menyeimbangkan kembali data Anda sehingga partisi hanya berisi item tunggal yang sering diakses tersebut. Dalam kasus ini, DynamoDB dapat memberikan throughput hingga maksimum partisi 3.000 RCU dan 1.000 WCU ke kunci primer item tunggal tersebut. Kapasitas adaptif tidak akan membagi koleksi item di beberapa partisi tabel ketika ada [indeks sekunder lokal](#) di tabel.

Merancang kunci partisi untuk mendistribusikan beban kerja Anda

Bagian kunci partisi pada kunci primer tabel menentukan partisi logis tempat data tabel disimpan. Pada gilirannya, hal ini akan memengaruhi partisi fisik yang mendasarinya. Desain kunci partisi yang tidak mendistribusikan permintaan I/O secara efektif dapat membuat partisi “panas” yang mengakibatkan throttling dan menggunakan kapasitas I/O yang disediakan secara tidak efisien.

Penggunaan optimal throughput yang disediakan pada tabel tidak hanya tergantung pada pola beban kerja masing-masing item, tetapi juga pada desain kunci partisi. Ini tidak berarti bahwa Anda harus mengakses semua nilai kunci partisi untuk mencapai tingkat throughput yang efisien, atau bahkan persentase nilai kunci partisi yang diakses harus tinggi. Artinya, semakin banyak nilai kunci partisi

berbeda yang diakses oleh beban kerja Anda, semakin banyak permintaan tersebut akan tersebar di seluruh ruang yang dipartisi. Secara umum, Anda akan menggunakan throughput yang disediakan secara lebih efisien seiring dengan meningkatnya rasio nilai kunci partisi yang diakses terhadap jumlah total nilai kunci partisi.

Berikut ini adalah perbandingan efisiensi throughput yang disediakan dari beberapa skema kunci partisi umum.

Nilai kunci partisi	Keseragaman
User ID, aplikasi memiliki banyak pengguna.	Baik
Kode status, hanya ada beberapa kemungkinan kode status.	Buruk
Tanggal pembuatan item, dibulatkan ke periode waktu terdekat (misalnya, hari, jam, atau menit).	Buruk
ID Perangkat, setiap perangkat mengakses data pada interval yang relatif sama.	Baik
ID Perangkat, meskipun ada banyak perangkat yang dilacak, salah satunya jauh lebih populer daripada yang lainnya.	Buruk

Jika tabel tunggal hanya memiliki sejumlah kecil nilai kunci partisi, pertimbangkan untuk mendistribusikan operasi tulis Anda di nilai kunci partisi yang lebih berbeda. Dengan kata lain, susun elemen kunci primer untuk menghindari satu nilai kunci partisi "panas" (banyak diminta) yang memperlambat performa secara keseluruhan.

Misalnya, pertimbangkan tabel dengan kunci primer komposit. Kunci partisi mewakili tanggal pembuatan item, dibulatkan ke hari terdekat. Kunci urutan adalah pengidentifikasi item. Pada hari tertentu, katakanlah 2014-07-09, semua item baru ditulis ke nilai kunci partisi tunggal (dan partisi fisik yang sesuai).

Jika tabel seluruhnya masuk ke dalam satu partisi (dengan mempertimbangkan pertumbuhan data Anda dari waktu ke waktu), dan jika persyaratan throughput baca dan tulis aplikasi Anda tidak

melebihi kemampuan baca dan tulis dari satu partisi, aplikasi Anda tidak akan mengalami throttling yang tidak terduga sebagai akibat dari pembuatan partisi.

Untuk menggunakan NoSQL Workbench untuk DynamoDB guna membantu memvisualisasikan desain kunci partisi Anda, lihat [Membangun model data dengan NoSQL Workbench](#).

Menggunakan pembagian tulis untuk mendistribusikan beban kerja secara merata

Salah satu cara untuk mendistribusikan penulisan dengan lebih baik di seluruh ruang kunci partisi di Amazon DynamoDB adalah dengan memperluas ruang tersebut. Hal ini dapat dilakukan dengan berbagai cara. Anda dapat menambahkan angka acak pada nilai kunci partisi untuk mendistribusikan item di antara partisi. Atau, Anda dapat menggunakan angka yang dihitung berdasarkan sesuatu yang Anda kueri.

Pembagian menggunakan akhiran acak

Salah satu strategi untuk mendistribusikan beban lebih merata di seluruh ruang kunci partisi adalah dengan menambahkan angka acak ke akhir nilai kunci partisi. Kemudian Anda mengacak penulisan di ruang yang lebih besar.

Misalnya, untuk kunci partisi yang menunjukkan tanggal hari ini, Anda dapat memilih nomor acak antara 1 dan 200 dan menggabungkannya sebagai akhiran untuk tanggal. Ini menghasilkan nilai kunci partisi seperti 2014-07-09.1, 2014-07-09.2, dan sebagainya, melalui 2014-07-09.200. Karena Anda mengacak kunci partisi, penulisan ke tabel pada setiap hari tersebar merata di sejumlah partisi. Hal ini menghasilkan paralelisme yang lebih baik dan throughput keseluruhan yang lebih tinggi.

Namun, untuk membaca semua item pada hari tertentu, Anda harus mengkueri item untuk semua akhiran lalu menggabungkan hasilnya. Misalnya, Anda akan terlebih dahulu mengeluarkan permintaan Query untuk nilai kunci partisi 2014-07-09.1. Kemudian mengeluarkan Query lainnya untuk 2014-07-09.2, dan seterusnya, melalui 2014-07-09.200. Terakhir, aplikasi Anda harus menggabungkan hasil dari semua permintaan Query tersebut.

Pembagian menggunakan akhiran terhitung

Strategi pengacakan dapat meningkatkan throughput tulis secara signifikan. Namun, sulit untuk membaca item tertentu karena Anda tidak mengetahui nilai akhiran mana yang digunakan saat menulis item tersebut. Untuk mempermudah membaca item satu per satu, Anda dapat menggunakan

strategi yang berbeda. Alih-alih menggunakan angka acak untuk mendistribusikan item di antara partisi, gunakan angka yang dapat Anda hitung berdasarkan sesuatu yang ingin Anda kueri.

Pertimbangkan contoh sebelumnya, yaitu tabel menggunakan tanggal hari ini dalam kunci partisi. Sekarang anggaplah setiap item memiliki atribut `OrderId` yang dapat diakses, dan Anda sering kali perlu mencari item berdasarkan ID urutan selain tanggal. Sebelum aplikasi Anda menulis item ke tabel, aplikasi Anda dapat menghitung akhiran hash berdasarkan ID urutan dan menambahkannya ke tanggal kunci partisi. Penghitungannya dapat menghasilkan angka antara 1 dan 200 yang terdistribusi cukup merata, mirip dengan yang dihasilkan strategi acak.

Perhitungan sederhana mungkin sudah cukup, seperti produk nilai titik kode UTF-8 untuk karakter dalam ID urutan, modulo 200, + 1. Nilai kunci partisi kemudian akan menjadi tanggal yang digabungkan dengan hasil penghitungan.

Dengan strategi ini, penulisan tersebar merata di seluruh nilai kunci partisi, serta di partisi fisik. Anda dapat dengan mudah melakukan operasi `GetItem` untuk item dan tanggal tertentu karena Anda dapat menghitung nilai kunci partisi untuk nilai `OrderId` tertentu.

Untuk membaca semua item pada hari tertentu, Anda masih harus `Query` setiap kunci `2014-07-09.N` (dengan `N` adalah 1–200), dan aplikasi Anda kemudian harus menggabungkan semua hasilnya. Manfaatnya adalah Anda menghindari satu nilai kunci partisi "panas" yang mengambil semua beban kerja.

Note

Untuk strategi yang lebih efisien yang dirancang khusus untuk menangani data deret waktu volume tinggi, lihat [Data deret waktu](#).

Mendistribusikan aktivitas tulis secara efisien selama pengunggahan data

Biasanya, ketika Anda memuat data dari sumber data lain, Amazon DynamoDB mempartisi data tabel Anda pada beberapa server. Anda akan mendapatkan performa yang lebih baik jika mengunggah data ke semua server yang dialokasikan secara bersamaan.

Contoh, misalkan Anda ingin mengunggah pesan pengguna ke tabel DynamoDB yang menggunakan kunci primer komposit dengan `UserID` sebagai kunci partisi dan `MessageID` sebagai kunci urutan.

Saat mengunggah data, salah satu pendekatan yang dapat Anda lakukan adalah mengunggah semua item pesan untuk setiap pengguna, satu per satu:

UserID	MessageID
U1	1
U1	2
U1	...
U1	... hingga 100
U2	1
U2	2
U2	...
U2	... hingga 200

Masalahnya dalam kasus ini adalah Anda tidak mendistribusikan permintaan tulis ke DynamoDB di seluruh nilai kunci partisi Anda. Anda mengambil satu nilai kunci partisi pada satu waktu dan mengunggah semua itemnya sebelum melanjutkan ke nilai kunci partisi berikutnya dan melakukan hal yang sama.

Di balik layar, DynamoDB mempartisi data di tabel Anda di beberapa server. Untuk sepenuhnya menggunakan semua kapasitas throughput yang disediakan untuk tabel, Anda harus mendistribusikan beban kerja ke nilai kunci partisi Anda. Dengan mengarahkan pekerjaan unggahan dalam jumlah yang tidak merata ke item yang semuanya memiliki nilai kunci partisi yang sama, Anda tidak sepenuhnya menggunakan semua sumber daya yang telah disediakan DynamoDB untuk tabel Anda.

Anda dapat mendistribusikan pekerjaan unggahan Anda dengan menggunakan kunci urutan untuk memuat satu item dari setiap nilai kunci partisi, kemudian item lain dari setiap nilai kunci partisi, dan seterusnya:

UserID	MessageID
U1	1
U2	1

UserID	MessageID
U3	1
...	...
U1	2
U2	2
U3	2
...	...

Setiap unggahan dalam urutan ini menggunakan nilai kunci partisi yang berbeda, sehingga lebih banyak server DynamoDB sibuk secara bersamaan dan meningkatkan performa throughput Anda.

Praktik terbaik untuk menggunakan kunci urutan untuk mengatur data

Dalam tabel Amazon DynamoDB, kunci utama yang secara unik mengidentifikasi setiap item dalam tabel dapat terdiri dari kunci partisi dan kunci sortir.

Kunci urutan yang dirancang dengan baik memiliki dua manfaat utama:

- Kunci-kunci tersebut mengumpulkan informasi terkait di satu tempat sehingga dapat dikueri secara efisien. Desain kunci urutan yang cermat memungkinkan Anda mengambil grup item terkait yang biasanya dibutuhkan menggunakan kueri rentang dengan operator seperti `begins_with`, `between`, `>`, `<`, dan sebagainya.
- Kunci sortir komposit memungkinkan Anda menentukan hubungan hierarkis (one-to-many) dalam data yang dapat Anda kueri di setiap tingkat hierarki.

Misalnya, dalam tabel yang mencantumkan lokasi geografis, Anda dapat menyusun kunci urutan sebagai berikut.

```
[country]#[region]#[state]#[county]#[city]#[neighborhood]
```

Ini akan memungkinkan Anda membuat kueri rentang yang efisien untuk daftar lokasi di salah satu tingkat agregasi ini, dari `country`, hingga `neighborhood`, dan segala sesuatu di antaranya.

Menggunakan kunci urutan untuk kontrol version

Banyak aplikasi yang perlu menyimpan riwayat revisi tingkat item untuk tujuan audit atau kepatuhan dan agar dapat mengambil versi terbaru dengan mudah. Ada pola desain efektif yang dapat mencapai tujuan ini menggunakan awalan kunci urutan:

- Untuk setiap item baru, buat dua salinan item: Satu berisi awalan nomor versi nol (seperti `v0_`) di awal kunci urutan, dan satu lagi berisi awalan nomor versi satu (seperti `v1_`).
- Setiap kali item diperbarui, gunakan awalan versi berikutnya yang lebih tinggi di kunci urutan versi yang diperbarui, dan salin konten yang diperbarui ke item dengan awalan versi nol. Artinya, versi terbaru item apa pun dapat ditemukan dengan mudah menggunakan awalan nol.

Misalnya, produsen suku cadang dapat menggunakan skema seperti yang diilustrasikan di bawah ini.

Primary Key		Data-Item Attributes...				
Partition Key	Sort Key	Attribute 1	Attribute 2	Attribute 3	Attribute 4	...
<i>Equipment_ID</i>	<i>(varies)</i>	Name: Biphasic Cardiometer <i>(equipment name)</i>	Factory_ID: S14_Tukwilla <i>(factory where manufactured)</i>	Line_ID: R_7 <i>(assembly-line ID)</i>		
Equipment_1	Details					
	v0_Audit	Auditor: Padma <i>(name of the auditor)</i>	Latest: 3 <i>(most recent audit version)</i>	Time: 2018-04-15T11:00 <i>(audit date and time)</i>	Result: Passed <i>(audit result)</i>	...etc.
	v1_Audit	Auditor: Rick <i>(name of the auditor)</i>	Time: 2018-03-14T11:00 <i>(audit date and time)</i>	Result: Open <i>(audit result)</i>	Report: 0943922EKG14 <i>(detailed problem report in S3)</i>	...etc.
	v2_Audit	Auditor: George <i>(name of the auditor)</i>	Time: 2018-03-18T11:00 <i>(audit date and time)</i>	Result: Open <i>(audit result)</i>	Report: 0943923EKG15 <i>(detailed problem report in S3)</i>	...etc.
	v3_Audit	Auditor: Padma <i>(name of the auditor)</i>	Time: 2018-04-15T11:00 <i>(audit date and time)</i>	Result: Passed <i>(audit result)</i>	Report: x792 <i>(pass confirmation report)</i>	...etc.

Item `Equipment_1` melewati serangkaian audit oleh berbagai auditor. Hasil dari setiap audit baru dicatat dalam item baru di tabel, dimulai dengan versi nomor satu, lalu menambahkan nomor untuk setiap revisi berturut-turut.

Ketika setiap revisi baru ditambahkan, lapisan aplikasi mengganti konten item versi nol (memiliki kunci urutan yang sama dengan `v0_Audit`) dengan konten revisi yang baru.

Setiap kali aplikasi perlu mengambil status audit terbaru, aplikasi dapat mengkueri awalan kunci urutan `v0_`.

Jika aplikasi perlu mengambil seluruh riwayat revisi, aplikasi dapat mengkueri semua item pada kunci partisi item dan memfilter item `v0_`.

Desain ini juga berfungsi untuk audit di beberapa bagian peralatan, jika Anda menyertakan masing-masing ID bagian dalam kunci urutan setelah awalan kunci pengurutan.

Praktik terbaik untuk menggunakan indeks sekunder di DynamoDB

Indeks sekunder sering kali penting untuk mendukung pola kueri yang diperlukan aplikasi Anda. Pada saat yang sama, menggunakan indeks sekunder secara berlebihan atau secara tidak efisien dapat menambah biaya dan menurunkan performa yang tidak perlu.

Daftar Isi

- [Pedoman umum tentang indeks sekunder di DynamoDB](#)
 - [Gunakan indeks secara efisien](#)
 - [Pilih proyeksi dengan hati-hati](#)
 - [Mooptimalkan kueri yang sering dilakukan untuk menghindari pengambilan](#)
 - [Memahami batas ukuran kumpulan item ketika membuat indeks sekunder lokal](#)
- [Memanfaatkan indeks jarang](#)
 - [Contoh indeks jarang di DynamoDB](#)
- [Menggunakan Indeks Sekunder Global untuk kueri agregasi yang terwujud](#)
- [Muatan berlebih Indeks Sekunder Global](#)
- [Menggunakan pembagian tulis Indeks Sekunder Global untuk kueri tabel selektif](#)
- [Menggunakan Indeks Sekunder Global untuk membuat replika yang akhirnya konsisten](#)

Pedoman umum tentang indeks sekunder di DynamoDB

Amazon DynamoDB mendukung dua jenis indeks sekunder:

- Indeks sekunder global (GSI)— Indeks dengan kunci partisi dan kunci urutan yang mungkin berbeda dari yang ada di tabel dasar. Indeks sekunder global dianggap "global" karena kueri pada indeks dapat menjangkau semua data di tabel dasar, di semua partisi. Indeks sekunder global tidak memiliki batasan ukuran dan memiliki pengaturan throughput tersendiri untuk aktivitas baca dan tulis yang terpisah dari yang ada di tabel.

- Indeks sekunder lokal (LSI)—Indeks yang memiliki kunci partisi yang sama dengan tabel dasar, tetapi kunci urutan yang berbeda. Indeks sekunder lokal bersifat "lokal" dalam arti bahwa setiap partisi indeks sekunder lokal dicakup ke partisi tabel dasar yang memiliki nilai kunci partisi yang sama. Dengan demikian, ukuran total item yang diindeks untuk setiap satu nilai kunci partisi tidak dapat melampaui 10 GB. Selain itu, indeks sekunder lokal memiliki pengaturan throughput yang disediakan yang sama untuk aktivitas baca dan tulis dengan tabel yang diindeks.

Setiap tabel di DynamoDB dapat memiliki hingga 20 indeks sekunder global (kuota default) dan 5 indeks sekunder lokal.

Indeks sekunder global sering kali lebih berguna daripada indeks sekunder lokal. Menentukan jenis indeks yang akan digunakan juga akan bergantung pada kebutuhan aplikasi Anda. Untuk perbandingan indeks sekunder global dan indeks sekunder lokal, dan informasi lebih lanjut tentang cara memilih di antara mereka, lihat [the section called “Bekerja dengan indeks”](#)

Berikut ini adalah beberapa prinsip umum dan pola desain yang perlu diingat saat membuat indeks di DynamoDB:

Topik

- [Gunakan indeks secara efisien](#)
- [Pilih proyeksi dengan hati-hati](#)
- [Mengoptimalkan kueri yang sering dilakukan untuk menghindari pengambilan](#)
- [Memahami batas ukuran kumpulan item ketika membuat indeks sekunder lokal](#)

Gunakan indeks secara efisien

Pertahankan jumlah indeks seminimal mungkin. Jangan membuat indeks sekunder pada atribut yang tidak sering Anda kueri. Indeks yang jarang digunakan berkontribusi pada peningkatan penyimpanan dan biaya I/O tanpa peningkatan performa aplikasi.

Pilih proyeksi dengan hati-hati

Karena indeks sekunder menggunakan penyimpanan dan throughput yang disediakan, Anda harus menjaga ukuran indeks sekecil mungkin. Selain itu, semakin kecil indeks, semakin besar keuntungan performa dibandingkan dengan kueri untuk tabel penuh. Jika kueri Anda biasanya hanya menampilkan sebagian kecil atribut, dan ukuran total atribut tersebut jauh lebih kecil dibandingkan keseluruhan item, hanya proyeksikan atribut yang sering Anda minta.

Jika Anda menginginkan lebih banyak aktivitas tulis pada tabel daripada aktivitas baca, ikuti praktik terbaik berikut:

- Pertimbangkan untuk memproyeksikan lebih sedikit atribut untuk meminimalkan ukuran item yang ditulis ke indeks. Namun, ini hanya berlaku jika ukuran atribut yang diproyeksikan lebih besar dari satu unit kapasitas tulis (1 KB). Sebagai contoh, jika ukuran entri indeks hanya 200 byte, DynamoDB membulatkannya hingga 1 KB. Dengan kata lain, selama item indeksnya kecil, Anda dapat memproyeksikan lebih banyak atribut tanpa biaya tambahan.
- Hindari memproyeksikan atribut yang Anda tahu akan jarang diperlukan dalam kueri. Setiap kali memperbarui atribut yang diproyeksikan dalam indeks, Anda juga dikenakan biaya tambahan untuk memperbarui indeks. Anda masih dapat mengambil atribut yang tidak diproyeksikan di Query dengan biaya yang lebih tinggi untuk throughput yang disediakan, tetapi biaya kueri mungkin jauh lebih rendah daripada biaya memperbarui indeks berulang kali.
- Tentukan ALL hanya jika Anda ingin kueri mengembalikan seluruh item tabel yang diurutkan berdasarkan kunci urutan yang berbeda. Memproyeksikan semua atribut akan menghilangkan kebutuhan untuk mengambil tabel, tetapi dalam banyak kasus, akan menggandakan biaya untuk penyimpanan dan aktivitas tulis.

Seimbangkan kebutuhan untuk menjaga indeks Anda sekecil mungkin terhadap kebutuhan untuk menjaga pengambilan tetap rendah, seperti yang dijelaskan di bagian berikutnya.

Mengoptimalkan kueri yang sering dilakukan untuk menghindari pengambilan

Untuk mendapatkan kueri tercepat dengan latensi serendah mungkin, proyeksikan semua atribut yang Anda harapkan akan dihasilkan oleh kueri tersebut. Khususnya, jika Anda mengkueri indeks sekunder lokal untuk atribut yang tidak diproyeksikan, DynamoDB otomatis mengambil atribut tersebut dari tabel, yang mengharuskan pembacaan seluruh item dari tabel. Hal ini menimbulkan latensi dan operasi I/O tambahan yang dapat Anda hindari.

Ingatlah bahwa kueri "sesekali" sering kali dapat berubah menjadi kueri "penting". Jika ada atribut yang tidak ingin Anda proyeksikan karena Anda hanya ingin mengkuerinya sesekali, pertimbangkan apakah keadaan mungkin berubah dan Anda pada akhirnya mungkin menyesal tidak memproyeksikan atribut tersebut.

Untuk informasi selengkapnya tentang pengambilan tabel, lihat [Pertimbangan throughput yang disediakan untuk Indeks Sekunder Lokal](#).

Memahami batas ukuran kumpulan item ketika membuat indeks sekunder lokal

Koleksi item adalah semua item dalam tabel dan indeks sekunder lokalnya yang memiliki kunci partisi yang sama. Koleksi item tidak dapat melampaui 10 GB, sehingga kehabisan ruang dapat terjadi untuk nilai kunci partisi tertentu.

Ketika Anda menambahkan atau memperbarui item tabel, DynamoDB memperbarui semua indeks sekunder lokal yang terpengaruh. Jika atribut yang diindeks ditetapkan dalam tabel, indeks sekunder lokal juga bertambah.

Saat Anda membuat indeks sekunder lokal, perhatikan jumlah data yang akan ditulis ke dalamnya, dan jumlah item data tersebut yang memiliki nilai kunci partisi yang sama. Jika Anda memperkirakan bahwa jumlah item tabel dan indeks untuk nilai kunci partisi tertentu mungkin melebihi 10 GB, pertimbangkan apakah Anda harus menghindari pembuatan indeks.

Jika tidak dapat menghindari pembuatan indeks sekunder lokal, Anda harus mengantisipasi batas ukuran koleksi item dan mengambil tindakan sebelum batas tersebut terlampaui. Sebagai praktik terbaik, Anda harus memanfaatkan [ReturnItemCollectionMetrics](#) parameter saat menulis item untuk memantau dan memperingatkan ukuran koleksi item yang mendekati batas ukuran 10GB. Melebihi ukuran koleksi item maksimum akan mengakibatkan upaya penulisan yang gagal. Anda dapat mengurangi masalah ukuran koleksi item dengan memantau dan memberi tahu ukuran koleksi item sebelum berdampak pada aplikasi Anda.

Note

Setelah dibuat, Anda tidak dapat menghapus indeks sekunder lokal.

Untuk strategi bekerja dalam batas dan pengambilan tindakan korektif, lihat [Batas ukuran koleksi barang](#).

Memanfaatkan indeks jarang

Untuk setiap item dalam tabel, DynamoDB menulis entri indeks yang sesuai hanya jika nilai kunci urutan indeks berada dalam item tersebut. Jika kunci urutan tidak muncul di setiap item tabel, atau jika kunci partisi indeks tidak ada dalam item, indeks dianggap jarang.

Indeks renggang berguna untuk kueri pada subbagian kecil tabel. Misalnya, anggaplah Anda memiliki tabel untuk menyimpan semua pesanan pelanggan Anda, dengan atribut kunci berikut ini:

- Kunci partisi: `CustomerId`
- Kunci urutan: `OrderId`

Untuk melacak pesanan terbuka, Anda dapat memasukkan atribut bernama `isOpen` dalam item pesanan yang belum dikirimkan. Kemudian, setelah pesanan dikirimkan, Anda dapat menghapus atribut tersebut. Jika Anda kemudian membuat indeks pada `CustomerId` (kunci partisi) dan `isOpen` (kunci urutan), hanya pesanan dengan `isOpen` yang ditetapkan yang akan muncul di dalamnya. Apabila Anda memiliki ribuan pesanan dan hanya sejumlah kecil yang terbuka, mengkueri indeks pesanan terbuka tersebut akan lebih cepat dan lebih murah daripada memindai seluruh tabel.

Daripada menggunakan jenis atribut seperti `isOpen`, Anda dapat menggunakan atribut dengan nilai yang menghasilkan urutan pesanan yang berguna dalam indeks. Misalnya, Anda dapat menggunakan atribut `OrderOpenDate` yang diatur ke tanggal ketika setiap pesanan dilakukan, lalu menghapusnya setelah pesanan dipenuhi. Dengan begitu, saat Anda mengkueri indeks jarang, item dikembalikan dan diurutkan berdasarkan tanggal ketika setiap pesanan dilakukan.

Contoh indeks jarang di DynamoDB

Indeks sekunder global bersifat jarang secara default. Ketika membuat indeks sekunder global, Anda menentukan kunci partisi dan, secara opsional, kunci urutan. Hanya item di tabel dasar yang berisi atribut tersebut yang muncul dalam indeks.

Dengan merancang indeks sekunder global menjadi jarang, Anda dapat menyediakannya dengan throughput tulis yang lebih rendah dibandingkan tabel dasar, tetapi tetap mencapai performa yang sangat baik.

Misalnya, aplikasi game dapat melacak semua skor setiap pengguna, tetapi umumnya hanya perlu mengkueri beberapa skor tinggi. Desain berikut menangani skenario ini secara efisien:

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	Date:	2017-11-14
			<i>(game score)</i>		<i>(date of game)</i>
	Game_2	Score:	69,450	Date:	2017-12-31
			<i>(game score)</i>		<i>(date of game)</i>
Padma	Game_3	Score:	135,900	Date:	2018-01-19
			<i>(game score)</i>		<i>(date of game)</i>
				Award:	Champ
					<i>(type of award)</i>
Padma	Game_4	Score:	25,350	Date:	2018-01-27
			<i>(game score)</i>		<i>(date of game)</i>
	Game_5	Score:	69,450	Date:	2028-01-19
			<i>(game score)</i>		<i>(date of game)</i>
Padma	Game_6	Score:	147,300	Date:	2018-02-02
			<i>(game score)</i>		<i>(date of game)</i>
Padma	Game_7	Score:	169,100	Date:	2018-03-10
			<i>(game score)</i>		<i>(date of game)</i>
				Award:	Champ
					<i>(type of award)</i>

Di sini, Rick telah memainkan tiga permainan dan mencapai status Champ di salah satunya. Padma telah memainkan empat permainan dan mencapai status Champ di dua di antaranya. Perhatikan bahwa atribut Award hanya ada pada item tempat pengguna mendapat penghargaan. Indeks sekunder global terkait akan seperti berikut ini:

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	

Indeks sekunder global hanya berisi skor tinggi yang sering dikueri, yang merupakan sebagian kecil item di tabel dasar.

Menggunakan Indeks Sekunder Global untuk kueri agregasi yang terwujud

Mempertahankan metrik kunci dan agregasi yang mendekati waktu nyata selain data yang berubah dengan cepat menjadi semakin berharga bagi bisnis untuk mengambil keputusan dengan cepat.

Misalnya, pustaka musik mungkin ingin menampilkan lagu-lagu yang paling banyak diunduh mendekati waktu nyata.

Pertimbangkan tata letak tabel pustaka musik berikut:

Music Library Table

Primary Key		Data-Item Attributes...					
Partition Key	Sort Key	Attribute 1		Attribute 2		Attribute 3	
Song-129 <i>(song ID)</i>	Details	Title: Wild Love <i>(song title)</i>	Artist: Argyboots <i>(artist or band name)</i>	Downloads: 15,314,822 <i>(lifetime total downloads)</i>	...etc.		
	Month-2018-01	GSI Primary Key		GSI Secondary Key			
		Month: 2018-01 <i>(download month)</i>	MonthTotal: 1,746,992 <i>(month total downloads)</i>				
	DId-9349823681	Time: 2018-01-01T00:00:07 <i>(download timestamp)</i>					
	DId-9349823682	Time: 2018-01-01T00:00:07 <i>(download timestamp)</i>					
DId-9349823683	Time: 2018-01-01T00:00:07 <i>(download timestamp)</i>						

Tabel dalam contoh ini menyimpan lagu dengan songID sebagai kunci partisi. Anda dapat mengaktifkan Amazon DynamoDB Streams pada tabel ini dan melampirkan fungsi Lambda ke alirannya sehingga saat setiap lagu diunduh, entri ditambahkan ke tabel dengan Partition-Key=SongID dan Sort-Key=DownloadID. Saat dilakukan, pembaruan ini memicu fungsi Lambda di DynamoDB Streams. Fungsi Lambda dapat mengagregasi dan mengelompokkan unduhan menurut songID dan memperbarui item tingkat atas, Partition-Key=songID, dan Sort-Key=Month. Perlu diingat bahwa jika eksekusi Lambda gagal tepat setelah menulis nilai agregat baru, eksekusi tersebut mungkin akan dicoba ulang dan mengagregasi nilai lebih dari sekali, sehingga Anda mendapatkan nilai perkiraan.

Untuk membaca pembaruan mendekati waktu nyata, dengan latensi milidetik satu digit, gunakan indeks sekunder global dengan ketentuan kueri Month=2018-01, ScanIndexForward=False, Limit=1.

Optimasi kunci lain yang digunakan di sini adalah bahwa indeks sekunder global merupakan indeks jarang dan tersedia hanya pada item yang perlu dikueri untuk mengambil data secara waktu nyata. Indeks sekunder global dapat melayani alur kerja tambahan yang memerlukan informasi tentang 10 lagu teratas yang populer, atau lagu apa pun yang diunduh pada bulan itu.

Muatan berlebih Indeks Sekunder Global

Meskipun Amazon DynamoDB memiliki kuota default 20 sekunder indeks global per tabel, dalam praktiknya, Anda dapat mengindeks di lebih dari 20 bidang data. Berbeda dari tabel dalam sistem manajemen basis data relasional (RDBMS), yang skemanya seragam, tabel di DynamoDB dapat menyimpan berbagai jenis item data sekaligus. Selain itu, atribut yang sama pada item yang berbeda dapat berisi jenis informasi yang berbeda sepenuhnya.

Perhatikan contoh tata letak tabel DynamoDB berikut yang menyimpan berbagai jenis data.

Primary Key		Data-Item Attributes...		
Partition Key	Sort Key	Attribute 1	Attribute 2	...
HR-974 <i>(employee ID)</i>	Employee_Name	Data: <input type="text" value="Murphy, John"/> <i>(employee name)</i>	Start: <input type="text" value="2008-11-08"/> <i>(start date)</i>	...etc.
	YYYY-Q1	Data: <input type="text" value="\$5,477"/> <i>(order totals in USD)</i>	Name: <input type="text" value="Murphy, John"/> <i>(employee name)</i>	
	HR_confidential	Data: <input type="text" value="2008-11-08"/> <i>(hire date)</i>	Name: <input type="text" value="Murphy, John"/> <i>(employee name)</i>	...etc.
	Warehouse_01	Data: <input type="text" value="Murphy, John"/> <i>(employee name)</i>		
	v0_Job_title	Data: <input type="text" value="Operator-1"/> <i>(job title)</i>	Start: <input type="text" value="2008-11-08"/> <i>(start date)</i>	...etc.
	v1_Job_title	Data: <input type="text" value="Operator-2"/> <i>(job title)</i>	Start: <input type="text" value="2016-11-04"/> <i>(start date)</i>	...etc.
	v2_Job_title	Data: <input type="text" value="Supervisor-1"/> <i>(job title)</i>	Start: <input type="text" value="2017-11-01"/> <i>(start date)</i>	...etc.

Atribut Data, yang lazim untuk semua item, memiliki konten yang berbeda tergantung pada item induknya. Jika Anda membuat indeks sekunder global untuk tabel yang menggunakan kunci urutan tabel sebagai kunci partisinya dan atribut Data sebagai kunci urutannya, Anda dapat membuat berbagai kueri menggunakan indeks sekunder global tunggal tersebut. Kueri tersebut dapat mencakup hal-hal berikut:

- Cari karyawan berdasarkan nama di indeks sekunder global, menggunakan `Employee_Name` sebagai nilai kunci partisi dan nama karyawan (misalnya `Murphy, John`) sebagai nilai kunci urutan.
- Gunakan indeks sekunder global untuk menemukan semua karyawan yang bekerja di gudang tertentu dengan mencari ID gudang (seperti `Warehouse_01`).
- Dapatkan daftar karyawan baru-baru ini, dengan mengkueri indeks sekunder global di `HR_confidential` sebagai nilai kunci partisi dan menggunakan rentang tanggal di nilai kunci urutan.

Menggunakan pembagian tulis Indeks Sekunder Global untuk kueri tabel selektif

Aplikasi sering kali perlu mengidentifikasi sebagian kecil item dalam tabel Amazon DynamoDB yang memenuhi syarat tertentu. Ketika item ini didistribusikan secara acak ke seluruh kunci partisi tabel, Anda dapat memanfaatkan pemindaian tabel untuk mengambil item ini. Opsi ini mungkin mahal, tetapi berfungsi dengan baik jika sejumlah besar item di tabel memenuhi ketentuan pencarian. Namun, ketika ruang kuncinya besar dan ketentuan pencariannya sangat selektif, strategi ini dapat menyebabkan banyak pemrosesan yang tidak perlu.

Solusi yang lebih baik adalah dengan mengkueri data. Untuk mengaktifkan kueri selektif di seluruh ruang kunci, Anda dapat menggunakan pembagian tulis dengan menambahkan atribut yang berisi nilai ($0-N$) ke setiap item yang akan Anda gunakan untuk kunci partisi indeks sekunder global.

Berikut adalah contoh skema yang menggunakan pembagian ini dalam alur kerja Peristiwa-Kritis:

Table	Primary Key	Data Attributes...				
	Partition Key					
Event_ID	Attribute 1	Attribute 2	Attribute 3	Attribute 4	...	
EID_12345	Time: 2018-02-07T08:42:40 <i>(event timestamp)</i>	State: INFO <i>(event state)</i>	GSI PK: (random: 0-N) <i>(random GSI-PK value)</i>	GSI SK: INFO#2018-02-07T08:42:40 <i>(composite state-time)</i>	...etc.	
EID_12346	Time: 2018-02-07T08:32:40 <i>(event timestamp)</i>	State: CRITICAL <i>(event state)</i>	GSI PK: (random: 0-N) <i>(random GSI-PK value)</i>	GSI SK: CRITICAL#2018-02-07T08:32:40 <i>(composite state-time)</i>	...etc.	
EID_12347	Time: 2018-02-07T08:22:40 <i>(event timestamp)</i>	State: WARN <i>(event state)</i>	GSI PK: (random: 0-N) <i>(random GSI-PK value)</i>	GSI SK: WARN#2018-02-07T08:22:40 <i>(composite state-time)</i>	...etc.	
EID_12348	Time: 2018-02-07T08:12:40 <i>(event timestamp)</i>	State: INFO <i>(event state)</i>	GSI PK: (random: 0-N) <i>(random GSI-PK value)</i>	GSI SK: INFO#2018-02-07T08:12:40 <i>(composite state-time)</i>	...etc.	

GSI	Primary Key		Data Attributes...
	Partition Key	Sort Key	
GSI PK	GSI SK		...
[0-N]	INFO#2018-02-07T08:42:40 <i>(composite state-time)</i>etc.
[0-N]	CRITICAL#2018-02-07T08:32:40 <i>(composite state-time)</i>etc.
[0-N]	WARN#2018-02-07T08:22:40 <i>(composite state-time)</i>etc.
[0-N]	INFO#2018-02-07T08:12:40 <i>(composite state-time)</i>etc.

Dengan menggunakan desain skema ini, item peristiwa didistribusikan ke seluruh partisi 0-N pada GSI, yang memungkinkan penyebaran baca menggunakan syarat pengurutan pada kunci komposit untuk mengambil semua item dengan status tertentu selama jangka waktu tertentu.

Pola skema ini memberikan hasil yang sangat selektif yang ditetapkan dengan biaya minimal, tanpa memerlukan pemindaian tabel.

Menggunakan Indeks Sekunder Global untuk membuat replika yang akhirnya konsisten

Anda dapat menggunakan indeks sekunder global untuk membuat replika tabel yang akhirnya konsisten. Membuat replika dapat memungkinkan Anda melakukan hal berikut:

- Atur berbagai kapasitas baca yang disediakan untuk berbagai pembaca. Sebagai contoh, misalkan Anda memiliki dua aplikasi: Satu menangani kueri prioritas tinggi dan membutuhkan performa baca

tingkat tertinggi, sedangkan satunya menangani kueri prioritas rendah yang dapat menoleransi throttling aktivitas baca.

Jika kedua aplikasi ini membaca dari tabel yang sama, beban baca berat dari aplikasi prioritas rendah dapat menggunakan semua kapasitas baca yang tersedia untuk tabel. Hal ini dapat mengakibatkan throttling pada aktivitas baca aplikasi prioritas tinggi.

Sebagai gantinya, Anda dapat membuat replika melalui indeks sekunder global yang kapasitas bacanya dapat diatur secara terpisah dari tabel itu sendiri. Anda kemudian dapat menggunakan aplikasi prioritas rendah untuk mengkueri replika, bukan tabel.

- Hilangkan pembacaan dari tabel seluruhnya. Misalnya, Anda mungkin memiliki aplikasi yang menangkap aktivitas clickstream dalam jumlah besar dari situs web, dan Anda tidak ingin mengambil risiko pembacaan mengganggu aktivitas tersebut. Anda dapat mengisolasi tabel ini dan mencegah pembacaan oleh aplikasi lain (lihat [Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci](#)), sekaligus membiarkan aplikasi lain membaca replika yang dibuat menggunakan indeks sekunder global.

Untuk membuat replika, siapkan indeks sekunder global yang memiliki skema kunci yang sama dengan tabel induk, dengan beberapa atau semua atribut non-kunci yang diproyeksikan ke dalamnya. Dalam aplikasi, Anda dapat mengarahkan beberapa atau semua aktivitas baca ke indeks sekunder global ini, bukan ke tabel induk. Anda kemudian dapat menyesuaikan kapasitas baca yang disediakan pada indeks sekunder global untuk menangani pembacaan tersebut tanpa mengubah kapasitas baca yang disediakan pada tabel induk.

Selalu ada penundaan penyebaran singkat antara penulisan ke tabel induk dan waktu ketika data yang ditulis muncul di indeks. Dengan kata lain, aplikasi Anda harus memperhitungkan bahwa replika indeks sekunder global pada akhirnya hanya konsisten dengan tabel induk.

Anda dapat membuat beberapa replika indeks sekunder global untuk mendukung pola pembacaan yang berbeda. Ketika Anda membuat replika, hanya proyeksikan atribut yang benar-benar dibutuhkan setiap pola baca. Suatu aplikasi kemudian dapat menggunakan lebih sedikit kapasitas baca yang disediakan untuk hanya memperoleh data yang diperlukan daripada harus membaca item dari tabel induk. Optimasi ini dapat menghasilkan penghematan biaya yang signifikan dari waktu ke waktu.

Praktik terbaik untuk menyimpan atribut dan item besar

Amazon DynamoDB membatasi ukuran setiap item yang Anda simpan dalam tabel hingga 400 KB (lihat). [Layanan, akun, dan tabel kuota di Amazon DynamoDB](#) Jika aplikasi Anda perlu menyimpan data dalam suatu item lebih dari yang diizinkan oleh batas ukuran DynamoDB, Anda dapat mencoba mengompresi satu atau lebih atribut besar atau memecah item menjadi beberapa item (diindeks secara efisien dengan kunci pengurutan). Anda juga dapat menyimpan item sebagai objek di Amazon Simple Storage Service (Amazon S3) dan menyimpan pengidentifikasi objek Amazon S3 di item DynamoDB Anda.

Sebagai praktik terbaik, Anda harus memanfaatkan [ReturnConsumedCapacity](#) parameter saat menulis item untuk memantau dan memperingatkan ukuran item yang mendekati ukuran item maksimum 400 KB. Melebihi ukuran item maksimum akan mengakibatkan upaya penulisan yang gagal. Pemantauan dan peringatan pada ukuran item akan memungkinkan Anda untuk mengurangi masalah ukuran item sebelum berdampak pada aplikasi Anda.

Mengompresi nilai atribut besar

Mengompresi nilai atribut yang besar dapat membuatnya sesuai dengan batas item di DynamoDB dan mengurangi biaya penyimpanan Anda. Algoritma kompresi seperti GZIP atau LZO menghasilkan output biner yang kemudian dapat Anda simpan dalam tipe Binary atribut di dalam item.

Sebagai contoh, pertimbangkan tabel yang menyimpan pesan yang ditulis oleh pengguna forum. Pesan semacam itu sering mengandung string teks yang panjang, yang merupakan kandidat untuk kompresi. Meskipun kompresi dapat mengurangi ukuran item, sisi negatifnya adalah nilai atribut terkompresi tidak berguna untuk pemfilteran.

Untuk kode sampel yang menunjukkan cara mengompresi pesan tersebut di DynamoDB, lihat berikut ini:

- [Contoh: Penanganan atribut jenis biner menggunakan AWS SDK for Java document API](#)
- [Contoh: Penanganan atribut jenis biner menggunakan API tingkat rendah AWS SDK for .NET](#)

Partisi vertikal

Solusi alternatif untuk menangani item besar adalah memecahnya menjadi potongan-potongan data yang lebih kecil dan mengaitkan semua item yang relevan dengan nilai kunci partisi. Anda kemudian

dapat menggunakan string kunci sortir untuk mengidentifikasi informasi terkait yang disimpan di sampingnya. Dengan melakukan ini, dan memiliki beberapa item yang dikelompokkan berdasarkan nilai kunci partisi yang sama, Anda membuat [koleksi item](#).

Untuk informasi lebih lanjut tentang pendekatan ini, lihat:

- [Gunakan partisi vertikal untuk menskalakan data secara efisien di Amazon DynamoDB](#)
- [Menerapkan partisi vertikal di Amazon DynamoDB menggunakan AWS Glue](#)

Menyimpan nilai atribut besar di Amazon S3

Seperti yang disebutkan sebelumnya, Anda juga dapat menggunakan Amazon S3 untuk menyimpan nilai atribut besar yang tidak dapat ditampung dalam item DynamoDB. Anda dapat menyimpannya sebagai objek di Amazon S3, lalu menyimpan pengidentifikasi objek di item DynamoDB Anda.

Anda juga dapat menggunakan dukungan metadata objek di Amazon S3 untuk menyediakan tautan kembali ke item induk di DynamoDB. Simpan nilai kunci primer item sebagai metadata Amazon S3 dari objek di Amazon S3. Tindakan ini sering kali membantu pemeliharaan objek Amazon S3.

Sebagai contoh, pertimbangkan tabel ProductCatalog di bagian [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#). Item dalam tabel ini menyimpan informasi tentang harga item, deskripsi, penulis buku, dan dimensi untuk produk lainnya. Jika Anda ingin menyimpan gambar produk yang terlalu besar untuk suatu item, Anda dapat menyimpan gambar tersebut di Amazon S3, bukan di DynamoDB.

Saat mengimplementasikan strategi ini, perhatikan hal berikut:

- DynamoDB tidak mendukung transaksi yang melintasi Amazon S3 dan DynamoDB. Oleh karena itu, aplikasi Anda harus menangani segala kegagalan, yang dapat mencakup pembersihan objek Amazon S3 yang tidak ada lagi.
- Amazon S3 membatasi panjang pengidentifikasi objek. Jadi, Anda harus mengatur data Anda sedemikian rupa sehingga tidak menghasilkan pengidentifikasi objek yang terlalu panjang atau melanggar batasan Amazon S3 lainnya.

Untuk informasi selengkapnya tentang cara menggunakan Amazon S3, lihat [Panduan Pengguna Amazon Simple Storage Service](#).

Praktik terbaik untuk penanganan data deret waktu di DynamoDB

Prinsip desain umum di Amazon DynamoDB merekomendasikan agar Anda meminimalkan jumlah tabel yang Anda gunakan. Untuk sebagian besar aplikasi, hanya satu tabel yang Anda butuhkan. Namun, untuk data deret waktu, Anda sering kali dapat menanganinya dengan menggunakan satu tabel per aplikasi per periode.

Pola desain untuk data deret waktu

Pertimbangkan skenario deret waktu yang umum, yaitu Anda ingin melacak peristiwa dalam jumlah besar. Pola akses tulis Anda adalah semua peristiwa yang direkam memiliki tanggal hari ini. Pola akses baca Anda mungkin membaca peristiwa hari ini paling sering, peristiwa kemarin lebih jarang, dan peristiwa lama sangat sedikit. Salah satu cara untuk mengatasinya adalah dengan memasukkan tanggal dan waktu saat ini ke dalam kunci primer.

Pola desain berikut sering kali menangani skenario semacam ini dengan efektif:

- Buat satu tabel per periode, dilengkapi dengan kapasitas baca dan tulis yang diperlukan serta indeks yang diperlukan.
- Sebelum akhir setiap periode, buat tabel untuk periode berikutnya. Setelah periode saat ini berakhir, arahkan lalu lintas peristiwa ke tabel baru. Anda dapat memberikan nama pada tabel-tabel ini yang menentukan periode yang telah dicatatnya.
- Segera setelah tabel tidak lagi digunakan untuk menulis, kurangi kapasitas tulis yang disediakan ke nilai yang lebih rendah (misalnya, 1 WCU), dan sediakan kapasitas baca berapa pun yang sesuai. Kurangi kapasitas baca yang disediakan pada tabel sebelumnya seiring bertambahnya usia. Anda dapat mengarsipkan atau menghapus tabel yang isinya jarang atau tidak pernah diperlukan.

Tujuannya adalah untuk mengalokasikan sumber daya yang diperlukan untuk periode saat ini yang akan mengalami volume lalu lintas tertinggi dan mengurangi penyediaan tabel lama yang tidak digunakan secara aktif, sehingga menghemat biaya. Bergantung pada kebutuhan bisnis Anda, Anda dapat mempertimbangkan pembagian tulis untuk mendistribusikan lalu lintas secara merata ke kunci partisi logis. Untuk informasi selengkapnya, lihat [Menggunakan pembagian tulis untuk mendistribusikan beban kerja secara merata](#).

Contoh tabel deret waktu

Berikut ini adalah contoh data deret waktu yang tabelnya saat ini disediakan dengan kapasitas baca/tulis yang lebih tinggi dan kapasitas tabel lama diturunkan karena jarang diakses.

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	...
...

Praktik terbaik untuk mengelola many-to-many hubungan

Adjacency list adalah pola desain yang berguna untuk pemodelan many-to-many hubungan di Amazon DynamoDB. Secara lebih umum, daftar ini menyediakan cara untuk merepresentasikan data grafik (simpul dan edge) di DynamoDB.

Pola desain daftar kedekatan

Ketika entitas yang berbeda dari aplikasi memiliki many-to-many hubungan di antara mereka, hubungan dapat dimodelkan sebagai daftar kedekatan. Dalam pola ini, semua entitas tingkat atas (sama dengan simpul dalam model grafik) direpresentasikan menggunakan kunci partisi. Hubungan apa pun dengan entitas lain (edge dalam grafik) direpresentasikan sebagai item dalam partisi dengan mengatur nilai kunci urutan ke ID entitas target (simpul target).

Keunggulan pola ini antara lain duplikasi data yang minimal dan pola kueri yang disederhanakan untuk menemukan semua entitas (simpul) yang terkait dengan entitas target (memiliki edge terhadap simpul target).

Contoh aktual kegunaan pola ini adalah sistem faktur dengan faktur yang berisi beberapa tagihan. Satu tagihan bisa masuk ke beberapa faktur. Kunci partisi dalam contoh ini adalah InvoiceID atau BillID. Partisi BillID memiliki semua atribut yang khusus untuk tagihan. Partisi InvoiceID memiliki item yang menyimpan atribut khusus faktur, dan item untuk masing-masing BillID yang dimasukkan ke faktur.

Skemanya akan seperti berikut.

	Primary Key		Data Attributes...	
	Partition Key	Sort Key (and GSI PK)		
Table	Invoice-92551	Inv_ID: Invoice-92551 <i>(invoice ID)</i>	Dated: 2018-02-07 <i>(date created)</i>	More attributes of this invoice...
		Bill_ID: Bill-4224663 <i>(bill ID)</i>	Dated: 2017-12-03 <i>(date created)</i>	Attributes of this bill <i>in this invoice..</i>
		Bill_ID: Bill-4224687 <i>(bill ID)</i>	Dated: 2018-01-09 <i>(date created)</i>	Attributes of this bill <i>in this invoice..</i>
	Invoice-92552	Inv_ID: Invoice-92552 <i>(invoice ID)</i>	Dated: 2018-03-04 <i>(date created)</i>	More attributes of this invoice...
		Bill_ID: Bill-4224687 <i>(bill ID)</i>	Dated: 2018-01-09 <i>(date created)</i>	Attributes of this bill <i>in this invoice..</i>
	Bill-4224663	Bill_ID: Bill-4224663 <i>(bill ID)</i>	Dated: 2017-12-03 <i>(date created)</i>	More attributes of this bill...
Bill-4224687	Bill_ID: Bill-4224687 <i>(bill ID)</i>	Dated: 2018-01-09 <i>(date created)</i>	More attributes of this bill...	

Dengan menggunakan skema sebelumnya, Anda dapat melihat bahwa semua tagihan dalam faktur dapat dikueri menggunakan kunci primer pada tabel. Untuk mencari semua faktur yang memuat bagian dari tagihan, buat indeks sekunder global pada kunci urutan tabel.

Proyeksi untuk indeks sekunder global akan seperti berikut.

	Primary Key	Projected Attributes...	
	Partition Key		
Bill-4224663	Bill_ID: <input type="text" value="Bill-4224663"/> <i>(table primary key)</i>	Attributes of this bill...	
	Inv_ID: <input type="text" value="Invoice-92551"/> <i>(table primary key)</i>	Attributes of this bill <i>in this invoice..</i>	
Bill-4224687	Bill_ID: <input type="text" value="Bill-4224687"/> <i>(table primary key)</i>	Attributes of this bill...	
	Inv_ID: <input type="text" value="Invoice-92551"/> <i>(table primary key)</i>	Attributes of this bill <i>in this invoice..</i>	
	Inv_ID: <input type="text" value="Invoice-92552"/> <i>(table primary key)</i>	Attributes of this bill <i>in this invoice..</i>	
Invoice-92551	Inv_ID: <input type="text" value="Invoice-92551"/> <i>(table primary key)</i>	Attributes of this invoice...	
Invoice-92552	Inv_ID: <input type="text" value="Invoice-92552"/> <i>(table primary key)</i>	Attributes of this invoice...	

Pola grafik terwujud

Banyak aplikasi dibangun berdasarkan pemahaman peringkat di antara sesama aplikasi, hubungan umum di antara entitas, status entitas tetangga, dan jenis alur kerja gaya grafik lainnya. Untuk jenis aplikasi ini, pertimbangkan pola desain skema berikut.

TABLE	Primary Key		Attributes		
	PK (NodeId)	SK (TypeTarget, GSI 2 SK)			
TABLE	1	DATE 2 BIRTH	Data	GSI PK	Graph Projections
			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)	
	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)		
	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)	NodeID	TypeTarget	Graph Projections	
GSI 1	O-N	1980-12-19	2	DATE 2	...
			NodeID	TypeTarget	
			1	DATE 2 BIRTH	
			NodeID		
		Guitar	5	TypeTarget	
			7	SKILL 7	
		Guitar Advanced	NodeID		
			5	TypeTarget	
		Jane Smith	5	Person 5	
			NodeID	TypeTarget	
			1	Person 5 FRIEND	
		Java Developer	NodeID	TypeTarget	
			6	SKILL 6	
		Java Developer Senior	NodeID		
			1	TypeTarget	
		John Doe	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			
		DATE 2 BIRTH	NodeID	1980-12-19		
			1			
			NodeID			
			5			
		PERSON 1	NodeID	Data		
			1			
		PERSON 1 FRIEND	NodeID	John Doe		
			5			
		PERSON 5	NodeID	Data		
			5			
		PERSON 5 FRIEND	NodeID	Jane Smith		
			1			
		PLACE 3	NodeID	Data		
			3			
		PLACE 3 BIRTH	NodeID	UK England London		
			5			
		PLACE 4	NodeID	Data		
			4			
PLACE 4 BIRTH	NodeID	USA texas Austin				
	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				

Skema sebelumnya menunjukkan struktur data grafik yang didefinisikan oleh satu set partisi data yang berisi item yang menentukan edge dan simpul grafik. Item edge berisi atribut Target dan Type. Atribut ini digunakan sebagai bagian dari nama kunci komposit TypeTarget "" untuk mengidentifikasi item dalam partisi di tabel primer atau dalam indeks sekunder global kedua.

Indeks sekunder global pertama dibangun di atas atribut Data. Atribut ini menggunakan muatan berlebih indeks sekunder global seperti yang dijelaskan sebelumnya untuk mengindeks beberapa jenis atribut, yaitu Dates, Names, Places, dan Skills. Di sini, satu indeks sekunder global mengindeks empat atribut yang berbeda secara efektif.

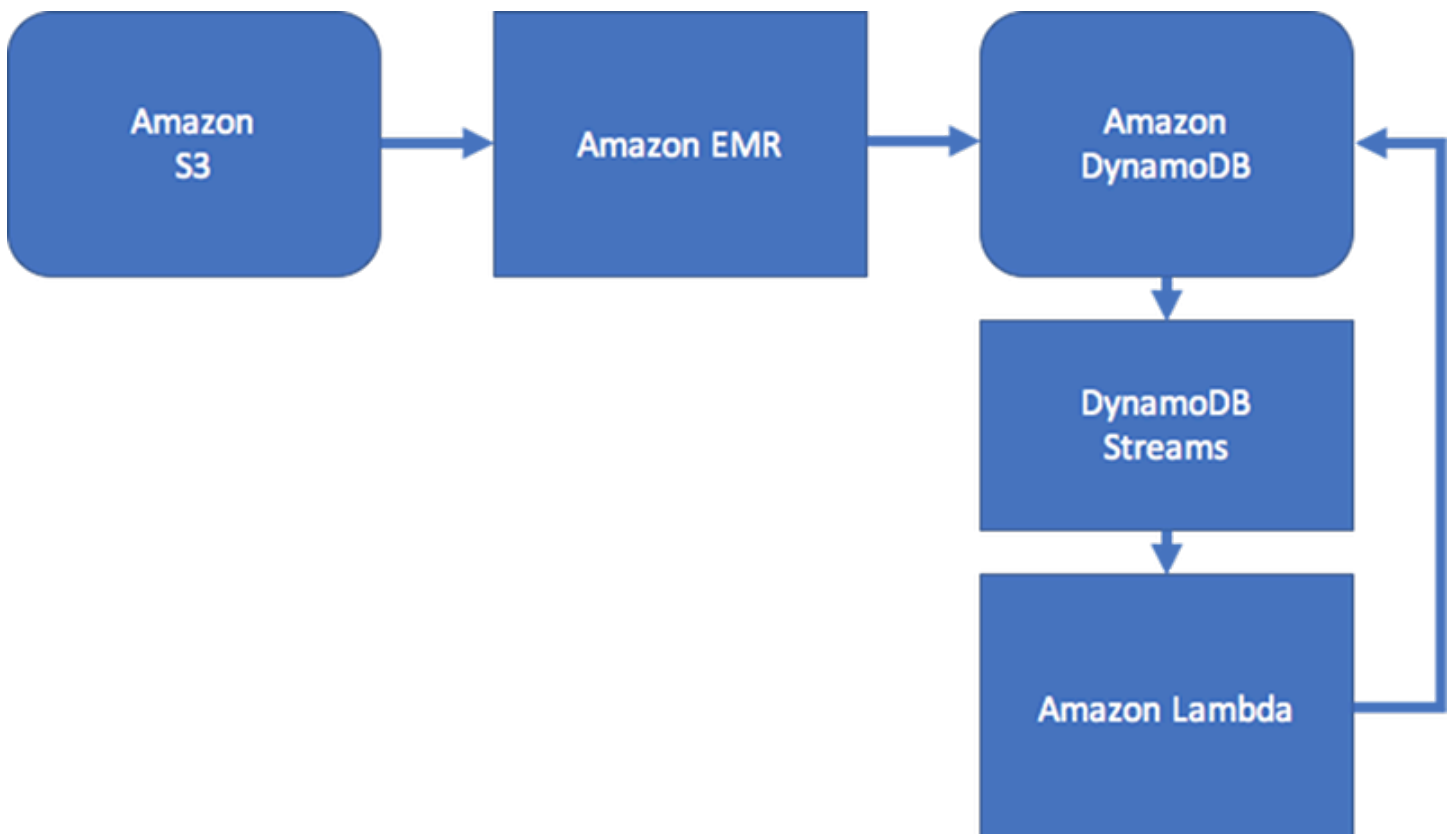
Saat memasukkan item ke dalam tabel, Anda dapat menggunakan strategi pembagian cerdas untuk mendistribusikan kumpulan item dengan agregasi besar (tanggal lahir, keterampilan) ke sebanyak mungkin partisi logis pada indeks sekunder global yang diperlukan untuk menghindari masalah baca/tulis yang panas.

Hasil dari kombinasi pola desain ini adalah penyimpanan data yang solid untuk alur kerja grafik waktu nyata yang sangat efisien. Alur kerja ini dapat memberikan status entitas tetangga performa tinggi

dan kueri agregasi edge untuk mesin rekomendasi, aplikasi jejaring sosial, peringkat simpul, agregasi subtree, dan kasus penggunaan grafik umum lainnya.

Jika kasus penggunaan Anda tidak sensitif terhadap konsistensi data waktu nyata, Anda dapat menggunakan proses Amazon EMR terjadwal untuk mengisi edge dengan agregasi ringkasan grafik yang relevan untuk alur kerja Anda. Anda dapat menggunakan proses terjadwal untuk menggabungkan hasil jika aplikasi Anda tidak perlu segera mengetahui kapan edge ditambahkan ke grafik.

Untuk mempertahankan beberapa tingkat konsistensi, desain dapat menyertakan Amazon DynamoDB Streams dan AWS Lambda untuk memproses pembaruan edge. Desain ini juga bisa menggunakan pekerjaan Amazon EMR untuk memvalidasi hasil secara berkala. Pendekatan ini diilustrasikan dengan diagram berikut. Ini biasanya digunakan dalam aplikasi jejaring sosial, jika biaya kueri waktu nyata tinggi dan kebutuhan untuk mengetahui pembaruan pengguna individu rendah.



Manajemen layanan TI (IT service-management/ITSM) dan aplikasi keamanan umumnya perlu merespons secara waktu nyata terhadap perubahan status entitas yang terdiri dari agregasi edge yang kompleks. Aplikasi semacam itu membutuhkan sistem yang dapat mendukung agregasi beberapa simpul waktu nyata dari hubungan tingkat kedua dan ketiga, atau traversal edge yang

kompleks. Jika kasus penggunaan Anda memerlukan jenis alur kerja kueri grafik waktu nyata ini, sebaiknya pertimbangkan menggunakan [Amazon Neptune](#) untuk mengelola alur kerja ini.

Note

Jika Anda perlu mengkueri set data yang sangat terhubung atau mengeksekusi kueri yang perlu melintasi beberapa simpul (juga dikenal sebagai kueri multi-hop) dengan latensi milidetik, sebaiknya gunakan [Amazon Neptune](#). Amazon Neptune adalah mesin basis data grafik berperforma tinggi yang dibuat khusus dan dioptimalkan untuk menyimpan miliaran hubungan dan mengkueri grafik dengan latensi milidetik.

Praktik terbaik untuk mengimplementasikan sistem basis data hibrida

Dalam beberapa situasi, migrasi dari satu atau beberapa sistem manajemen basis data relasional (RDBMS) ke Amazon DynamoDB mungkin tidak menguntungkan. Dalam kasus tersebut, mungkin lebih baik untuk membuat sistem hibrida.

Jika Anda tidak ingin memigrasikan semuanya ke DynamoDB

Misalnya, beberapa organisasi memiliki investasi besar dalam kode yang menghasilkan banyak laporan yang diperlukan untuk akuntansi dan operasional. Waktu yang dibutuhkan untuk menghasilkan laporan tidak penting bagi mereka. Fleksibilitas sistem relasional sangat cocok untuk tugas semacam ini, dan membuat ulang semua laporan tersebut dalam konteks NoSQL mungkin sangat sulit.

Beberapa organisasi juga mempertahankan berbagai sistem relasional lama yang telah mereka peroleh atau warisi selama beberapa dekade. Memigrasikan data dari sistem ini mungkin terlalu berisiko dan mahal untuk membenarkan upaya tersebut.

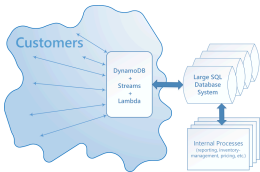
Namun, organisasi yang sama kini mungkin menyadari bahwa operasi mereka bergantung pada situs web yang berhubungan dengan pelanggan dengan lalu lintas tinggi, dan respons milidetik sangat penting. Sistem relasional tidak dapat memenuhi persyaratan ini kecuali dengan biaya yang besar (dan sering kali tidak dapat diterima).

Dalam situasi ini, jawabannya mungkin dengan membuat sistem hibrida, yaitu DynamoDB menciptakan tampilan data terwujud yang disimpan dalam satu atau beberapa sistem relasional dan

menangani permintaan lalu lintas tinggi terhadap tampilan ini. Sistem jenis ini berpotensi mengurangi biaya dengan menghilangkan perangkat keras server, pemeliharaan, dan lisensi RDBMS yang sebelumnya diperlukan untuk menangani lalu lintas yang berhubungan dengan pelanggan.

Cara penerapan sistem hibrida

DynamoDB dapat memanfaatkan DynamoDB Streams AWS Lambda dan mengintegrasikan secara mulus dengan satu atau lebih sistem database relasional yang ada:



Sebuah sistem yang mengintegrasikan DynamoDB AWS Lambda Streams dan dapat memberikan beberapa keuntungan:

- Sistem dapat beroperasi sebagai cache persisten tampilan terwujud.
- Sistem dapat diatur untuk mengisi secara bertahap dengan data saat data tersebut dikueri, dan saat data dimodifikasi dalam sistem SQL. Artinya, seluruh tampilan tidak perlu diisi sebelumnya. Sebagai hasilnya, kapasitas throughput yang disediakan lebih mungkin digunakan secara efisien.
- Sistem memiliki biaya administrasi yang rendah dan sangat tersedia serta dapat diandalkan.

Agar integrasi semacam ini dapat diimplementasikan, pada dasarnya tiga jenis interoperasi harus disediakan.



1. Isi cache DynamoDB secara bertahap. Saat sebuah item dikueri, cari terlebih dahulu di DynamoDB. Jika tidak ada, cari di sistem SQL, dan muat ke DynamoDB.
2. Tulis melalui cache DynamoDB. Saat pelanggan mengubah nilai di DynamoDB, fungsi Lambda dipicu untuk menulis kembali data baru ke sistem SQL.
3. Perbarui DynamoDB dari sistem SQL. Saat proses internal seperti manajemen inventaris atau penetapan harga mengubah nilai dalam sistem SQL, prosedur tersimpan dipicu untuk menyebarkan perubahan ke tampilan terwujud DynamoDB.

Operasi ini sangat mudah, dan tidak semuanya diperlukan untuk setiap skenario.

Solusi hibrida juga dapat berguna ketika Anda ingin mengandalkan DynamoDB, tetapi Anda juga ingin mempertahankan sistem relasional kecil untuk kueri satu kali, atau untuk operasi yang memerlukan keamanan khusus atau yang tidak kritis terhadap waktu.

Praktik terbaik untuk memodelkan data relasional di DynamoDB

Bagian ini menyediakan praktik terbaik untuk memodelkan data relasional di Amazon DynamoDB. Pertama, kami memperkenalkan konsep pemodelan data tradisional. Kemudian, kami menjelaskan keuntungan menggunakan DynamoDB dibandingkan sistem manajemen basis data relasional tradisional—bagaimana DynamoDB menghilangkan kebutuhan untuk operasi JOIN dan mengurangi overhead.

Kami kemudian menjelaskan cara menyusun tabel DynamoDB yang menskalakan secara efisien. Terakhir, kami memberikan contoh cara memodelkan data relasional di DynamoDB.

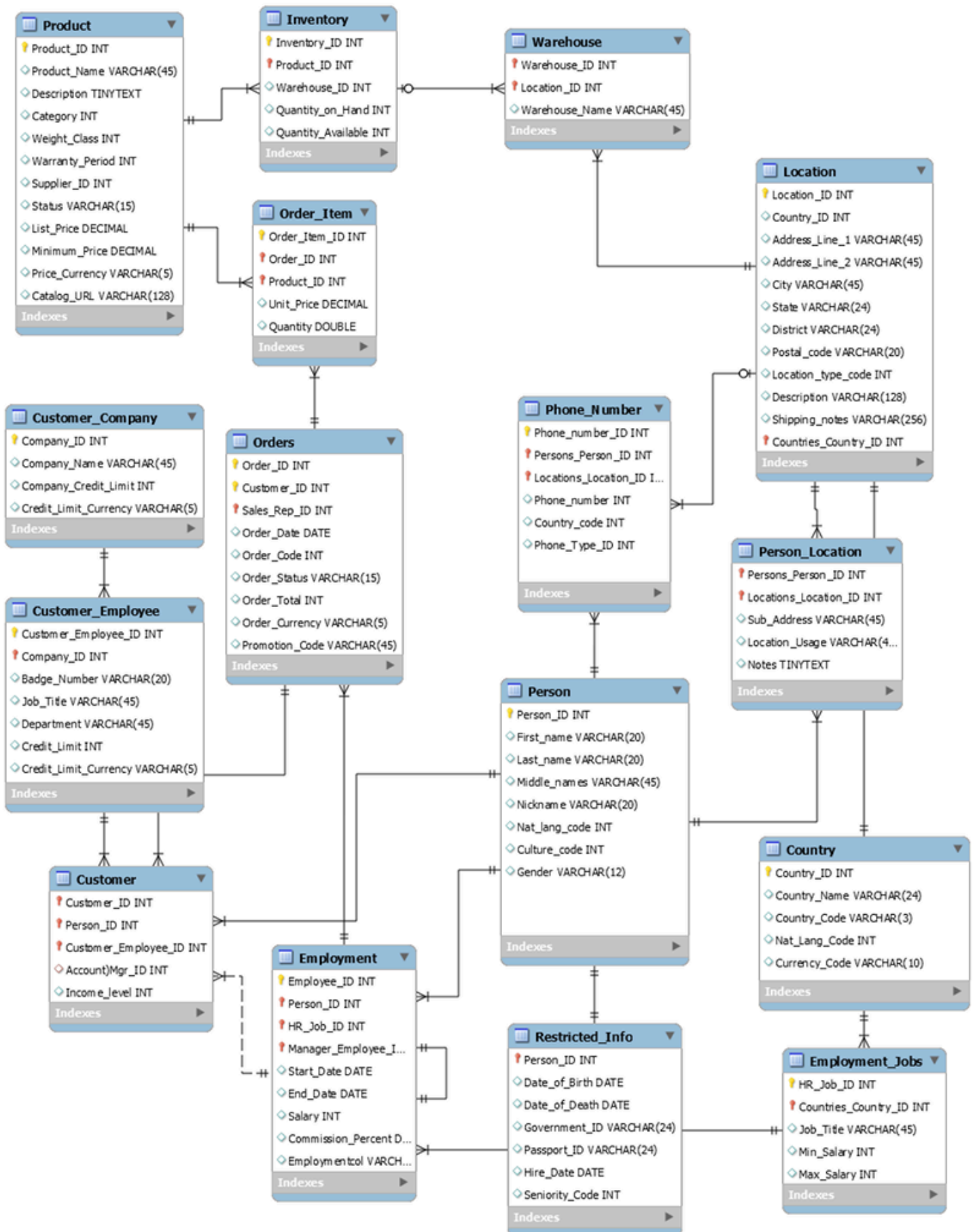
Topik

- [Model basis data relasional tradisional](#)
- [Cara DynamoDB menghilangkan kebutuhan akan operasi JOIN](#)
- [Bagaimana transaksi DynamoDB menghilangkan overhead ke proses tulis](#)
- [Langkah pertama untuk memodelkan data relasional di DynamoDB](#)
- [Contoh memodelkan data relasional di DynamoDB](#)

Model basis data relasional tradisional

Sistem manajemen basis data relasional tradisional (RDBMS) menyimpan data dalam struktur relasional yang dinormalisasi. Tujuan dari model data relasional adalah untuk mengurangi duplikasi data (melalui normalisasi) untuk mendukung integritas referensial dan mengurangi anomali data.

Skema berikut adalah contoh dari model data relasional untuk penerapan urutan-entri generik. Penerapan ini mendukung skema sumber daya manusia yang mendukung sistem pendukung operasional dan bisnis dari produsen teoretis.



Sebagai layanan basis data non-relasional, DynamoDB menawarkan banyak keunggulan dibandingkan sistem manajemen basis data relasional tradisional.

Cara DynamoDB menghilangkan kebutuhan akan operasi JOIN

Sebuah RDBMS menggunakan bahasa query terstruktur (SQL) untuk mengembalikan data ke aplikasi. Karena normalisasi model data, kueri semacam ini biasanya memerlukan penggunaan operator JOIN untuk menggabungkan data dari satu atau beberapa tabel.

Misalnya, untuk menghasilkan daftar item pesanan pembelian yang diurutkan berdasarkan jumlah stok di semua gudang yang dapat mengirimkan setiap item, Anda bisa mengeluarkan kueri SQL berikut terhadap skema sebelumnya.

```
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
```

Kueri SQL semacam ini dapat menyediakan API fleksibel untuk mengakses data, tetapi kueri tersebut membutuhkan pemrosesan dalam jumlah besar. Setiap gabungan dalam kueri meningkatkan kompleksitas runtime kueri karena data untuk setiap tabel harus ditahapkan dan kemudian dikumpulkan untuk mengembalikan set hasil.

Faktor lain yang dapat memengaruhi durasi yang dibutuhkan untuk menjalankan kueri adalah ukuran tabel dan apakah kolom yang digabungkan memiliki indeks. Kueri sebelumnya memulai kueri kompleks di beberapa tabel, lalu mengurutkan set hasilnya.

Menghilangkan kebutuhan untuk JOINS adalah inti dari pemodelan data NoSQL. Inilah sebabnya kami membangun DynamoDB untuk mendukung Amazon.com, dan mengapa DynamoDB dapat memberikan performa yang konsisten di semua skala. Mengingat kompleksitas runtime kueri SQL dan JOINS, performa RDBMS dalam skala besar tidak konstan. Hal ini menyebabkan masalah performa seiring berkembangnya aplikasi pelanggan.

Meskipun normalisasi data mengurangi jumlah data yang disimpan ke disk, sumber daya yang paling terkendala sehingga berdampak performa sering kali adalah waktu CPU dan latensi jaringan.

DynamoDB dibangun untuk meminimalkan kendala tersebut dengan menghilangkan JOINS (dan mendorong denormalisasi data) dan mengoptimalkan arsitektur basis data untuk sepenuhnya menjawab kueri aplikasi dengan satu permintaan ke item. Kualitas ini memungkinkan DynamoDB

untuk menghadirkan performa milidetik satu digit di semua skala. Ini karena kompleksitas runtime untuk operasi DynamoDB konstan, terlepas dari ukuran data, untuk pola akses umum.

Bagaimana transaksi DynamoDB menghilangkan overhead ke proses tulis

Faktor lain yang dapat memperlambat RDBMS adalah penggunaan transaksi untuk menulis ke skema yang dinormalisasi. Seperti yang ditunjukkan dalam contoh, struktur data relasional yang digunakan oleh sebagian besar aplikasi pemrosesan transaksi online (OLTP) harus diuraikan dan didistribusikan ke beberapa tabel logis jika disimpan di RDBMS.

Oleh karena itu, kerangka transaksi yang sesuai dengan ACID diperlukan untuk menghindari kondisi balapan dan masalah integritas data yang dapat terjadi jika aplikasi mencoba membaca objek yang sedang dalam proses penulisan. Kerangka transaksi semacam itu, jika digabungkan dengan skema relasional, dapat menambah overhead yang signifikan untuk proses tulis.

Implementasi transaksi di DynamoDB dapat mencegah masalah penskalaan umum yang dialami pada RDBMS. DynamoDB melakukannya dengan menerbitkan transaksi sebagai panggilan API tunggal dan membatasi jumlah item yang dapat diakses dalam transaksi tunggal tersebut. Transaksi yang berjalan lama dapat menyebabkan masalah operasional karena mengunci data dalam jangka waktu yang lama, atau terus-menerus, karena transaksi tidak pernah ditutup.

Untuk mencegah masalah demikian di DynamoDB, transaksi diimplementasikan dengan dua operasi API yang berbeda: `TransactWriteItems` dan `TransactGetItems`. Operasi API ini tidak memiliki semantik awal dan akhir yang lazim di RDBMS. Selanjutnya, DynamoDB memiliki batas akses 100 item dalam transaksi untuk mencegah transaksi yang berjalan lama. Untuk mempelajari selengkapnya tentang transaksi DynamoDB, lihat [Bekerja dengan transaksi](#).

Dengan demikian, ketika bisnis Anda memerlukan respons latensi rendah terhadap kueri dengan lalu lintas tinggi, memanfaatkan sistem NoSQL umumnya masuk akal secara teknis dan ekonomis. Amazon DynamoDB membantu memecahkan masalah yang membatasi skalabilitas sistem relasional dengan menghindari masalah tersebut.

Performa RDBMS biasanya tidak menskalakan dengan baik karena alasan berikut:

- Penggabungan yang mahal digunakan untuk mengumpulkan kembali tampilan yang diperlukan dari hasil kueri.
- Sistem tersebut menormalkan data dan menyimpannya di beberapa tabel yang memerlukan sejumlah kueri untuk menulis ke disk.
- Ini umumnya menimbulkan biaya performa sistem transaksi yang sesuai dengan ACID.

DynamoDB menskalakan dengan baik karena alasan berikut:

- Fleksibilitas skema memungkinkan DynamoDB menyimpan data hierarkis kompleks dalam satu item.
- Desain kunci komposit memungkinkan DynamoDB menyimpan item terkait secara berdekatan di tabel yang sama.
- Transaksi dilakukan dalam satu operasi. Batas jumlah item yang dapat diakses adalah 100, untuk menghindari operasi yang berjalan lama.

Kueri terhadap penyimpanan data menjadi jauh lebih sederhana, sering kali dalam bentuk berikut:

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

DynamoDB dapat memberikan data yang diminta dengan lebih efisien dibandingkan RDBMS dalam contoh sebelumnya.

Langkah pertama untuk memodelkan data relasional di DynamoDB

Important

Desain NoSQL membutuhkan pola pikir yang berbeda dari desain RDBMS. Untuk RDBMS, Anda dapat membuat model data yang dinormalisasi tanpa memikirkan pola akses. Anda kemudian dapat memperluasnya nanti ketika ada pertanyaan dan persyaratan kueri baru. Sebaliknya, di Amazon DynamoDB, Anda tidak boleh mulai merancang skema sampai Anda mengetahui pertanyaan-pertanyaan yang perlu dijawab. Memahami masalah bisnis dan kasus penggunaan aplikasi di awal sangatlah penting.

Untuk mulai merancang tabel DynamoDB yang akan menskalakan secara efisien, Anda harus melakukan beberapa langkah pertama untuk mengidentifikasi pola akses yang diperlukan oleh operasi dan sistem dukungan bisnis (OSS/BSS) yang memerlukan dukungan:

- Untuk aplikasi baru, tinjau cerita pengguna tentang aktivitas dan sasaran. Dokumentasikan berbagai kasus penggunaan yang Anda identifikasi, dan analisis pola akses yang mereka butuhkan.
- Untuk aplikasi yang sudah ada, analisis log kueri untuk mengetahui bagaimana orang-orang saat ini menggunakan sistem tersebut dan apa pola akses utamanya.

Setelah menyelesaikan proses ini, Anda akan mendapatkan daftar seperti berikut.

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

Dalam aplikasi yang sebenarnya, daftar Anda mungkin lebih panjang. Namun, kumpulan ini menunjukkan berbagai kompleksitas pola kueri yang mungkin Anda temukan di lingkungan produksi.

Pendekatan umum terhadap desain skema DynamoDB adalah mengidentifikasi entitas lapisan aplikasi dan menggunakan denormalisasi serta agregasi kunci komposit untuk mengurangi kompleksitas kueri.

Dalam DynamoDB, ini berarti menggunakan kunci urutan komposit, indeks sekunder global dengan muatan berlebih, tabel/indeks yang dipartisi, dan pola desain lainnya. Anda dapat menggunakan elemen-elemen ini untuk mengatur data sehingga aplikasi dapat mengambil apa pun yang dibutuhkannya untuk pola akses tertentu menggunakan satu kueri pada tabel atau indeks. Pola utama yang dapat Anda gunakan untuk memodelkan skema yang dinormalisasi yang ditampilkan dalam [Pemodelan relasional](#) adalah pola daftar kedekatan. Pola lain yang digunakan dalam desain ini dapat meliputi pembagian tulis indeks sekunder global, muatan berlebih indeks sekunder global, kunci komposit, dan agregasi yang terwujud.

Important

Secara umum, Anda harus mempertahankan tabel sesedikit mungkin dalam aplikasi DynamoDB. Pengecualian mencakup kasus yang melibatkan data deret waktu bervolume tinggi, atau set data yang memiliki pola akses yang sangat berbeda. Tabel tunggal dengan indeks terbalik biasanya dapat mengaktifkan kueri sederhana untuk membuat dan mengambil struktur data hierarki kompleks yang diperlukan oleh aplikasi Anda.

Untuk menggunakan NoSQL Workbench untuk DynamoDB guna membantu memvisualisasikan desain kunci partisi Anda, lihat [Membangun model data dengan NoSQL Workbench](#).

Contoh memodelkan data relasional di DynamoDB

Contoh ini menjelaskan cara memodelkan data relasional di Amazon DynamoDB. Desain tabel DynamoDB sesuai dengan skema entri urutan relasional yang ditampilkan dalam [Pemodelan relasional](#). Desain ini mengikuti [Pola desain daftar kedekatan](#), yang merupakan cara umum untuk merepresentasikan struktur data relasional di DynamoDB.

Pola desain mengharuskan Anda untuk menentukan set jenis entitas yang biasanya berkorelasi dengan berbagai tabel dalam skema relasional. Item entitas kemudian ditambahkan ke tabel menggunakan kunci primer gabungan (partisi dan urutan). Kunci partisi pada item entitas ini adalah atribut yang mengidentifikasi item secara unik dan disebut secara umum pada semua item sebagai PK. Atribut kunci urutan berisi nilai atribut yang dapat Anda gunakan untuk indeks terbalik atau indeks sekunder global. Hal ini umumnya disebut sebagai SK.

Anda menentukan entitas berikut, yang mendukung skema entri urutan relasional.

1. HR-Employee - PK: EmployeeID, SK: Nama Karyawan
2. HR-Region - PK: RegionID, SK: Nama Wilayah
3. HR-Negara - PK: CountryId, SK: Nama Negara
4. HR-Location - PK: LocationID, SK: Nama Negara
5. HR-Job - PK: JobID, SK: Jabatan
6. HR-Department - PK: DepartmentID, SK: DepartmentID
7. Pelanggan OE - PK: CustomerID, SK, ID AccountRep
8. OE-Order - PK OrderID, SK: CustomerID
9. OE-Product - PK: ProductID, SK: Nama Produk
10. OE-Warehouse - PK: WarehouseID, SK: Nama Wilayah

Setelah menambahkan item entitas ini ke tabel, Anda dapat menentukan hubungan di antara item entitas tersebut dengan menambahkan item edge ke partisi item entitas. Tabel berikut menunjukkan langkah ini.

Dalam contoh ini, partisi `Employee`, `Order`, dan `Product` Entity pada tabel memiliki item edge lain yang berisi penunjuk ke item entitas lain pada tabel. Berikutnya, tentukan beberapa indeks

sekunder global (GSI) untuk mendukung semua pola akses yang ditetapkan sebelumnya. Tidak semua item entitas menggunakan jenis nilai yang sama untuk atribut kunci primer atau kunci urutan. Yang diperlukan hanyalah atribut primer utama dan kunci urutan untuk dimasukkan ke dalam tabel.

Fakta bahwa sebagian entitas ini menggunakan nama diri dan yang lainnya menggunakan ID entitas lain sebagai nilai kunci urutan memungkinkan indeks sekunder global yang sama untuk mendukung beberapa jenis kueri. Teknik ini disebut muatan berlebih GSI. Teknik ini secara efektif menghilangkan batas default 20 indeks sekunder global untuk tabel yang berisi beberapa jenis item. Ini ditunjukkan dalam diagram berikut sebagai GSI 1.

GSI 2 dirancang untuk mendukung pola akses aplikasi yang cukup umum, yaitu untuk mengumpulkan semua item di tabel yang memiliki status tertentu. Untuk tabel besar dengan distribusi item yang tidak merata di seluruh status yang tersedia, pola akses ini dapat menghasilkan hot key, kecuali jika item didistribusikan ke lebih dari satu partisi logis yang dapat dikueri secara paralel. Pola desain ini disebut `write sharding`.

Untuk melakukannya pada GSI 2, aplikasi menambahkan atribut kunci primer GSI 2 ke setiap item Pesanan. Aplikasi mengisinya dengan nomor acak dalam kisaran 0-N, dengan N secara umum dapat dihitung menggunakan rumus berikut, kecuali ada alasan khusus untuk melakukan sebaliknya.

```
ItemsPerRCU = 4KB / AvgItemSize
```

```
PartitionMaxReadRate = 3K * ItemsPerRCU
```

```
N = MaxRequiredIO / PartitionMaxReadRate
```

Sebagai contoh, misalkan Anda mengharapkan hal berikut:

- Hingga 2 juta pesanan akan berada dalam sistem, yang berkembang hingga 3 juta dalam 5 tahun.
- Hingga 20 persen dari pesanan ini akan berada dalam status OPEN pada waktu tertentu.
- Catatan pesanan rata-rata sekitar 100 byte, dengan tiga OrderItem catatan di partisi pesanan yang masing-masing sekitar 50 byte, memberi Anda ukuran entitas pesanan rata-rata 250 byte.

Untuk tabel tersebut, penghitungan faktor N akan seperti berikut ini.

```
ItemsPerRCU = 4KB / 250B = 16
```

```
PartitionMaxReadRate = 3K * 16 = 48K
```

$$N = (0.2 * 3M) / 48K = 13$$

Dalam hal ini, Anda perlu mendistribusikan semua pesanan ke setidaknya 13 partisi logis di GSI 2 untuk memastikan bahwa pembacaan semua item `Order` dengan status `OPEN` tidak menyebabkan partisi panas pada lapisan penyimpanan fisik. Memasukkan jumlah ini untuk memungkinkan anomali dalam set data adalah praktik yang baik. Jadi, model yang menggunakan $N = 15$ mungkin baik-baik saja. Seperti yang disebutkan sebelumnya, Anda dapat melakukan ini dengan menambahkan nilai $0-N$ acak ke atribut GSI 2 PK dari setiap data `Order` dan `OrderItem` yang dimasukkan ke tabel.

Perincian ini mengasumsikan bahwa pola akses yang memerlukan pengumpulan semua faktur `OPEN` relatif jarang terjadi sehingga Anda dapat menggunakan kapasitas lonjakan untuk memenuhi permintaan. Anda dapat mengkueri indeks sekunder global berikut menggunakan syarat Kunci Urutan State dan Date Range untuk menghasilkan sebagian atau semua `Orders` dalam status tertentu sesuai kebutuhan.

Dalam contoh ini, item didistribusikan secara acak ke 15 partisi logis. Struktur ini berfungsi karena pola akses memerlukan sejumlah besar item untuk diambil. Oleh karena itu, kecil kemungkinannya bahwa salah satu dari 15 thread akan mengembalikan set hasil kosong yang berpotensi mewakili kapasitas yang terbuang. Kueri selalu menggunakan 1 unit kapasitas baca (RCU) atau 1 unit kapasitas tulis (WCU), meskipun tidak ada yang dikembalikan atau tidak ada data yang ditulis.

Jika pola akses memerlukan kueri kecepatan tinggi pada indeks sekunder global ini yang mengembalikan set hasil yang jarang, sebaiknya gunakan algoritma hash untuk mendistribusikan item daripada pola acak. Dalam hal ini, Anda dapat memilih atribut yang diketahui ketika kueri dijalankan pada saat runtime dan meng-hash atribut tersebut ke dalam ruang kunci 0–14 saat item dimasukkan. Kemudian, item tersebut dapat dibaca secara efisien dari indeks sekunder global.

Terakhir, Anda dapat meninjau kembali pola akses yang ditetapkan sebelumnya. Berikut ini adalah daftar pola akses dan ketentuan kueri yang akan Anda gunakan dengan versi DynamoDB baru pada aplikasi untuk mengakomodasinya.

	Pola akses	Ketentuan kueri
1	Cari Detail Karyawan berdasarkan ID Karyawan	Kunci Primer pada tabel, ID="HR-EMPLOYEE"
2	Kueri Detail Karyawan berdasarkan Nama Karyawan	Gunakan GSI-1, PK="Employee Name"

	Pola akses	Ketentuan kueri
3	Dapatkan detail pekerjaan karyawan saat ini saja	Kunci Primer pada tabel, PK=HR-EMPLOYEE-1, SK dimulai dengan "JH"
4	Dapatkan Pesanan pelanggan untuk rentang tanggal	Gunakan GSI-1, PK=CUSTOMER1, SK="STATUS-DATE", untuk masing-masing StatusCode
5	Tampilkan semua Pesanan dalam status OPEN untuk rentang tanggal di seluruh pelanggan	Gunakan GSI-2, PK=query secara paralel untuk rentang [0..N], SK antara OPEN-Date1 dan OPEN-Date2
6	Semua Karyawan yang baru direkrut	Gunakan GSI-1, PK="HR-CO NFIDENTIAL', SK > date1
7	Temukan semua Karyawan di Gudang tertentu	Gunakan GSI-1, PK=WAREHOUSE1
8	Dapatkan semua Orderitems untuk suatu Produk termasuk inventaris lokasi gudang	Gunakan GSI-1, PK=PRODUCT1
9	Dapatkan pelanggan dengan Perwakilan Akun	Gunakan GSI-1, PK=ACCOUNT-REP
10	Dapatkan pesanan berdasarkan Perwakilan Akun dan tanggal	Gunakan GSI-1, PK=ACCOUNT-REP, SK="STATUS-DATE", untuk masing-masing StatusCode
11	Dapatkan semua karyawan dengan Jabatan tertentu	Gunakan GSI-1, PK=JOBTITLE

	Pola akses	Ketentuan kueri
12	Dapatkan inventaris berdasarkan Produk dan Gudang	Kunci Primer pada tabel, PK=OE-PRODUCT1,SK=PRODUCT1
13	Dapatkan total inventaris produk	Kunci Primer pada tabel, PK=OE-PRODUCT1,SK=PRODUCT1
14	Dapatkan Perwakilan Akun diberi peringkat berdasarkan Total Pesanan dan Periode Penjualan	Gunakan GSI-1, PK=YYYY-Q1, =Salah scanIndexForward

Praktik terbaik untuk mengkueri dan memindai data

Bagian ini membahas beberapa praktik terbaik untuk menggunakan operasi Query dan Scan di Amazon DynamoDB.

Pertimbangan performa untuk pemindaian

Secara umum, efisiensi operasi Scan lebih rendah daripada operasi lain di DynamoDB. Operasi Scan selalu memindai seluruh tabel atau indeks sekunder. Kemudian, operasi ini memfilter nilai-nilai untuk memberikan hasil yang Anda inginkan, yang pada dasarnya menambahkan langkah tambahan untuk menghapus data dari kumpulan hasil.

Jika memungkinkan, Anda harus menghindari penggunaan operasi Scan pada indeks atau tabel besar dengan filter yang menghapus banyak hasil. Selain itu, seiring dengan berkembangnya tabel atau indeks, operasi Scan melambat. Operasi Scan memeriksa setiap item untuk nilai yang diminta dan dapat menggunakan throughput yang disediakan untuk indeks atau tabel besar dalam satu operasi. Untuk waktu respons yang lebih cepat, rancang tabel dan indeks Anda agar aplikasi Anda dapat menggunakan Query, bukan Scan. (Untuk tabel, Anda juga dapat mempertimbangkan penggunaan API `GetItem` dan `BatchGetItem`.)

Atau, Anda dapat merancang aplikasi Anda untuk menggunakan operasi Scan dengan cara yang meminimalkan dampak pada tingkat permintaan Anda. Ini dapat mencakup pemodelan

jika penggunaan indeks sekunder global mungkin lebih efisien daripada operasi Scan. Informasi selengkapnya tentang proses ini ada di video berikut.

[Memodelkan pola akses kecepatan rendah](#)

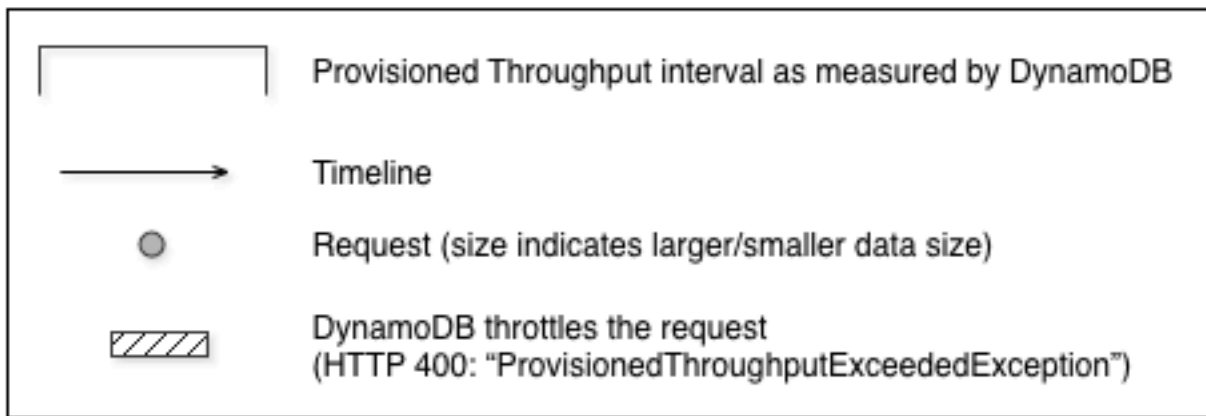
Menghindari lonjakan mendadak dalam aktivitas baca

Jika Anda membuat tabel, Anda akan mengatur persyaratan unit kapasitas baca dan tulisnya. Untuk baca, unit kapasitasnya dinyatakan sebagai jumlah permintaan baca data 4 KB yang sangat konsisten per detik. Untuk bacaan akhir konsisten, unit kapasitas baca memiliki dua permintaan baca 4 KB per detik. Operasi Scan melakukan bacaan akhir konsisten default, dan dapat mengembalikan hingga 1 MB (satu halaman) data. Oleh karena itu, satu permintaan Scan dapat menggunakan (ukuran halaman 1 MB / ukuran item 4 KB) / 2 (bacaan akhir konsisten) = 128 operasi baca. Jika Anda meminta bacaan sangat konsisten, operasi Scan akan menggunakan throughput yang disediakan dua kali lebih banyak – 256 operasi baca.

Hal ini menunjukkan lonjakan tiba-tiba pada penggunaan, dibandingkan dengan kapasitas baca yang dikonfigurasi untuk tabel. Penggunaan unit kapasitas melalui pemindaian ini mencegah permintaan lain yang berpotensi lebih penting untuk tabel yang sama menggunakan unit kapasitas yang tersedia. Akibatnya, Anda mungkin mendapatkan pengecualian `ProvisionedThroughputExceeded` untuk permintaan tersebut.

Masalahnya bukan hanya peningkatan mendadak pada unit kapasitas yang digunakan Scan. Pemindaian juga kemungkinan akan menggunakan semua unit kapasitasnya dari partisi yang sama karena pemindaian meminta item baca yang berdampingan di partisi. Artinya, permintaan tersebut mengenai partisi yang sama, sehingga menyebabkan semua unit kapasitasnya terpakai, dan membatasi permintaan lain ke partisi tersebut. Jika permintaan untuk membaca data tersebar di beberapa partisi, operasi tidak akan membatasi partisi tertentu.

Diagram berikut mengilustrasikan dampak lonjakan penggunaan unit kapasitas secara mendadak oleh operasi Query dan Scan, serta dampaknya pada permintaan Anda lainnya terhadap tabel yang sama.



1. Good: Even distribution of requests and size



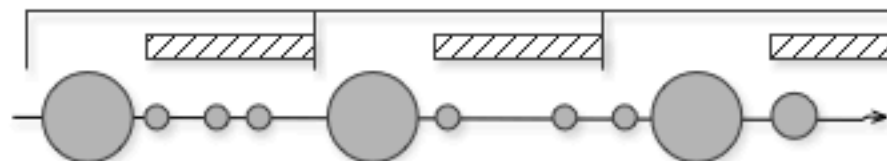
2. Not as Good: Frequent requests in bursts



3. Bad: A few random large requests



4. Bad: Large scan operations



Seperti yang diilustrasikan di sini, lonjakan penggunaan dapat memengaruhi throughput yang disediakan pada tabel dalam beberapa cara:

1. Baik: Distribusi permintaan dan ukuran yang merata

2. Tidak cukup baik: Permintaan yang sering terjadi secara beruntun
3. Buruk: Beberapa permintaan besar acak
4. Buruk: Operasi pemindaian besar

Daripada menggunakan operasi Scan besar, Anda dapat menggunakan teknik berikut untuk meminimalkan dampak pemindaian pada throughput yang disediakan di tabel.

- Mengurangi ukuran halaman

Karena operasi Pindai membaca seluruh halaman (secara default, 1 MB), Anda dapat mengurangi dampak operasi pemindaian dengan mengatur ukuran halaman yang lebih kecil. Operasi Scan menyediakan parameter Limit yang dapat Anda gunakan untuk mengatur ukuran halaman untuk permintaan Anda. Setiap permintaan Query atau Scan yang memiliki ukuran halaman yang lebih kecil menggunakan lebih sedikit operasi baca dan membuat "jeda" di antara setiap permintaan. Misalnya, katakanlah setiap item berukuran 4 KB dan Anda mengatur ukurannya menjadi 40 item. Permintaan Query kemudian hanya akan menggunakan 20 operasi bacaan akhir konsisten atau 40 operasi bacaan sangat konsisten. Sejumlah besar operasi Query atau Scan yang lebih kecil akan memungkinkan permintaan penting Anda yang lain untuk berhasil tanpa mengalami throttling.

- Mengisolasi operasi pemindaian

DynamoDB dirancang untuk skalabilitas yang mudah. Dengan demikian, aplikasi dapat membuat tabel untuk tujuan yang berbeda, bahkan mungkin menduplikasi konten di beberapa tabel. Anda ingin melakukan pemindaian pada tabel yang tidak mengambil lalu lintas "vital". Beberapa aplikasi menangani beban ini dengan memutar lalu lintas setiap jam antara dua tabel—satu untuk lalu lintas vital, dan satu lagi untuk pembukuan. Aplikasi lain dapat melakukan ini dengan melakukan setiap penulisan pada dua tabel: tabel "vital", dan tabel "bayangan".

Konfigurasi aplikasi Anda untuk mencoba kembali permintaan apa pun yang menerima kode respons yang menunjukkan bahwa Anda telah melampaui throughput yang disediakan. Atau, tingkatkan throughput yang disediakan untuk tabel Anda menggunakan operasi UpdateTable. Jika Anda mengalami lonjakan sementara dalam beban kerja yang menyebabkan throughput Anda melebihi, terkadang, melampaui tingkat yang disediakan, coba ulang permintaan dengan mundur eksponensial. Untuk informasi selengkapnya tentang cara mengimplementasikan mundur eksponensial, lihat [Percobaan ulang kesalahan dan mundur eksponensial](#).

Memanfaatkan pemindaian paralel

Banyak aplikasi mendapatkan keuntungan dari penggunaan operasi Scan paralel dibandingkan pemindaian berurutan. Misalnya, aplikasi yang memproses tabel data historis besar dapat melakukan pemindaian paralel jauh lebih cepat dibandingkan pemindaian berurutan. Beberapa thread pekerja dalam proses "sweeper" latar belakang dapat memindai tabel dengan prioritas rendah tanpa memengaruhi lalu lintas produksi. Dalam masing-masing contoh ini, Scan paralel digunakan sedemikian rupa sehingga tidak membuat aplikasi lain kekurangan sumber daya throughput yang disediakan.

Meskipun pemindaian paralel menguntungkan, pemindaian ini dapat memberikan tuntutan besar pada throughput yang disediakan. Dengan pemindaian paralel, aplikasi Anda memiliki beberapa pekerja yang semuanya menjalankan operasi Scan secara bersamaan. Operasi ini dapat dengan cepat menghabiskan seluruh kapasitas baca yang disediakan pada tabel Anda. Dalam hal ini, aplikasi lain yang perlu mengakses tabel mungkin akan mengalami throttling.

Pemindaian paralel dapat menjadi pilihan tepat jika kondisi berikut terpenuhi:

- Ukuran tabel sebesar 20 GB atau lebih.
- Throughput baca yang disediakan pada tabel tidak digunakan sepenuhnya.
- Operasi Scan yang berurutan terlalu lambat.

Memilih TotalSegments

Pengaturan terbaik untuk `TotalSegments` tergantung pada data spesifik Anda, pengaturan throughput yang disediakan pada tabel, dan kebutuhan performa Anda. Anda mungkin perlu bereksperimen untuk melakukannya dengan benar. Sebaiknya mulai dengan rasio sederhana, seperti satu segmen per 2 GB data. Misalnya, untuk tabel 30 GB, Anda dapat mengatur `TotalSegments` ke 15 (30 GB / 2 GB). Aplikasi Anda kemudian akan menggunakan 15 pekerja, masing-masing pekerja memindai segmen yang berbeda.

Anda juga dapat memilih nilai untuk `TotalSegments` yang didasarkan pada sumber daya klien. Anda dapat mengatur `TotalSegments` ke angka berapa pun dari 1 hingga 1000000, dan DynamoDB akan memungkinkan Anda memindai jumlah segmen tersebut. Misalnya, jika klien Anda membatasi jumlah thread yang dapat berjalan secara bersamaan, Anda dapat meningkatkan `TotalSegments` secara bertahap hingga mendapatkan performa Scan terbaik dengan aplikasi Anda.

Pantau pemindaian paralel Anda untuk mengoptimalkan penggunaan throughput yang disediakan, sekaligus memastikan bahwa aplikasi Anda yang lain tidak kekurangan sumber daya. Tingkatkan nilai `TotalSegments` jika Anda tidak menggunakan seluruh throughput yang disediakan tetapi masih mengalami throttling dalam permintaan Scan Anda. Kurangi nilai `TotalSegments` jika permintaan Scan menggunakan lebih banyak throughput yang disediakan daripada yang ingin Anda gunakan.

Praktik terbaik untuk desain tabel DynamoDB

Prinsip desain umum di Amazon DynamoDB merekomendasikan agar Anda meminimalkan jumlah tabel yang Anda gunakan. Dalam sebagian besar kasus, sebaiknya Anda menggunakan satu tabel. Namun, jika satu atau sejumlah kecil tabel tidak dapat digunakan, pedoman ini mungkin berguna.

- Batas per akun tidak dapat ditingkatkan di atas 10.000 tabel per akun. Jika aplikasi Anda memerlukan lebih banyak tabel, rencanakan untuk mendistribusikan tabel ke beberapa akun. Untuk informasi selengkapnya, lihat [kuota layanan, akun, dan tabel di Amazon DynamoDB](#).
- Pertimbangkan batas bidang kontrol untuk operasi bidang kontrol bersamaan yang dapat memengaruhi manajemen tabel Anda.
- Bekerja dengan arsitek AWS solusi untuk memvalidasi pola desain Anda untuk desain multi-tenant.

Praktik terbaik untuk desain tabel global DynamoDB

Tabel global dibangun berdasarkan jejak global Amazon DynamoDB untuk memberi Anda basis data yang terkelola sepenuhnya, multi-Wilayah, dan multi-aktif yang memberikan performa baca dan tulis yang cepat serta lokal, untuk aplikasi global berskala besar. Dengan tabel global, data Anda akan direplikasi secara otomatis di seluruh AWS wilayah pilihan Anda. Karena tabel global menggunakan API DynamoDB yang ada, tidak diperlukan perubahan pada aplikasi Anda. Tidak ada biaya atau komitmen di muka untuk menggunakan tabel global, dan Anda hanya membayar untuk sumber daya yang Anda gunakan.

Topik

- [Panduan preskriptif untuk desain tabel global DynamoDB](#)
- [Fakta kunci tentang desain tabel global DynamoDB](#)
- [Kasus penggunaan](#)
- [Mode tulis dengan tabel global](#)

- [Meminta perutean dengan tabel global](#)
- [Mengevakuasi Wilayah dengan tabel global](#)
- [Perencanaan kapasitas throughput untuk tabel global](#)
- [Daftar periksa persiapan untuk tabel global dan Pertanyaan yang Sering Diajukan](#)

Panduan preskriptif untuk desain tabel global DynamoDB

Penggunaan tabel global yang efisien memerlukan pertimbangan yang cermat terhadap faktor-faktor seperti mode tulis pilihan Anda, model perutean, dan proses evakuasi. Anda harus melengkapi aplikasi Anda di setiap Wilayah dan siap menyesuaikan rute atau melakukan evakuasi untuk menjaga kesehatan global. Sehingga Anda akan mendapatkan kumpulan data yang terdistribusi secara global dengan pembacaan dan penulisan latensi rendah serta perjanjian tingkat layanan 99,999%.

Fakta kunci tentang desain tabel global DynamoDB

- Ada dua versi tabel global: versi saat ini [Tabel Global versi 2019.11.21 \(Saat ini\)](#) (kadang-kadang disebut "V2"), dan [Versi tabel global 2017.11.29 \(Legacy\)](#) (kadang-kadang disebut "V1"). Panduan ini berfokus secara khusus pada versi terbaru, V2.
- Tanpa menggunakan tabel global, DynamoDB adalah layanan Regional. Ini memiliki ketersediaan tinggi dan secara intrinsik memiliki ketahanan terhadap kegagalan infrastruktur suatu Wilayah, termasuk kegagalan seluruh zona ketersediaan (AZ). Tabel DynamoDB Wilayah tunggal memiliki 99,99% ketersediaan <https://aws.amazon.com/dynamodb/sla/> Perjanjian Tingkat Layanan (SLA).
- Dengan penggunaan tabel global, DynamoDB memungkinkan tabel mereplikasi datanya di antara dua Wilayah atau lebih. Tabel DynamoDB multi-Wilayah memiliki 99,999% ketersediaan SLA. Dengan perencanaan yang tepat, tabel global dapat membantu menciptakan arsitektur yang tangguh dan menolak kegagalan Regional.
- Tabel global menggunakan model replikasi aktif-aktif. Dari perspektif DynamoDB, tabel di setiap Wilayah memiliki kedudukan yang sama untuk menerima permintaan baca dan tulis. Setelah menerima permintaan penulisan, tabel replika lokal akan mereplikasi penulisan tersebut ke Wilayah yang berpartisipasi lainnya di latar belakang.
- Item direplikasi satu per satu. Item yang diperbarui dalam satu transaksi mungkin tidak direplikasi bersama.
- Setiap partisi tabel di Wilayah sumber mereplikasi penulisannya secara paralel dengan setiap partisi lainnya. Urutan penulisan dalam Wilayah jarak jauh mungkin tidak cocok dengan urutan penulisan yang terjadi dalam Wilayah sumber. Untuk informasi selengkapnya tentang partisi tabel,

lihat postingan blog [Penskalaan DynamoDB: Dampak partisi, hot key, dan pemisahan panas terhadap performa](#).

- Item yang baru ditulis biasanya disebar ke semua tabel replika dalam hitungan detik. Wilayah terdekat cenderung menyebarkan lebih cepat.
- Amazon CloudWatch menyediakan `ReplicationLatency` metrik untuk setiap pasangan Wilayah. Ini dihitung berdasarkan item yang tiba dan membandingkan waktu kedatangannya dengan waktu penulisan awal serta menghitung rata-ratanya. Pengaturan waktu disimpan CloudWatch di dalam Wilayah sumber. Melihat pengaturan waktu rata-rata dan maksimum dapat membantu menentukan jeda replikasi rata-rata dan terburuk. Tidak ada SLA pada latensi ini.
- Jika item yang sama diperbarui pada waktu yang hampir bersamaan (dalam jendela `ReplicationLatency` ini) di dua Wilayah yang berbeda, dan penulisan kedua terjadi sebelum penulisan pertama direplikasi, ada potensi konflik penulisan. Tabel global menyelesaikan konflik demikian dengan mekanisme penulis terakhir menang, berdasarkan timestamp penulisan. Penulisan pertama “kalah” dengan penulisan kedua. Konflik ini tidak dicatat dalam CloudWatch atau AWS CloudTrail.
- Setiap item memiliki timestamp tulis terakhir yang disimpan sebagai properti sistem privat. Pendekatan penulis terakhir menang diimplementasikan dengan menggunakan penulisan bersyarat yang mengharuskan timestamp item masuk lebih besar daripada timestamp item yang ada.
- Tabel global akan mereplikasi semua item ke semua Wilayah yang berpartisipasi. Jika ingin memiliki cakupan replikasi yang berbeda, Anda dapat membuat beberapa tabel berbeda dan memasukkan Wilayah yang berpartisipasi yang berbeda ke masing-masing tabel.
- Penulisan akan diterima di Wilayah lokal meskipun Wilayah replika sedang offline atau `ReplicationLatency` berkembang. Tabel lokal terus mencoba mereplikasi item ke tabel jarak jauh hingga setiap item berhasil.
- Jika suatu Wilayah menjadi offline sepenuhnya, ketika kemudian kembali online, semua replikasi keluar dan masuk yang tertunda akan dicoba ulang. Tidak ada tindakan khusus yang diperlukan untuk mengembalikan sinkronisasi tabel. Mekanisme penulis terakhir menang memastikan data pada akhirnya akan menjadi konsisten.
- Anda dapat menambahkan Region baru ke tabel DynamoDB kapan saja. DynamoDB akan menangani sinkronisasi awal dan replikasi yang sedang berlangsung. Jika suatu Wilayah dihapus, meskipun itu adalah Wilayah asli, hanya tabel untuk Wilayah tersebut yang akan dihapus.
- DynamoDB tidak memiliki titik akhir global. Semua permintaan dibuat ke titik akhir regional, yang kemudian mengakses instans tabel global yang bersifat lokal di Wilayah tersebut.

- Panggilan ke DynamoDB tidak boleh dilakukan lintas Wilayah. Praktik terbaiknya adalah lapisan komputasi di satu Wilayah hanya mengakses titik akhir DynamoDB lokal untuk Wilayah tersebut secara langsung. Jika masalah terdeteksi dalam Wilayah, apakah masalah tersebut ada di lapisan DynamoDB atau di tumpukan sekitarnya, maka lalu lintas pengguna akhir harus diarahkan ke lapisan komputasi berbeda yang dihosting di Wilayah yang berbeda. Berkat replikasi tabel global, Wilayah yang berbeda sudah memiliki salinan lokal data yang sama untuk digunakan secara lokal. Dalam beberapa situasi, lapisan komputasi di satu Wilayah dapat meneruskan permintaan ke lapisan komputasi Wilayah lain untuk diproses, tetapi lapisan komputasi ini tidak boleh mengakses titik akhir DynamoDB jarak jauh secara langsung. Untuk informasi selengkapnya tentang kasus penggunaan khusus ini, lihat [Perutean permintaan lapisan komputasi](#).

Kasus penggunaan

Tabel global memberikan manfaat umum ini:

- Pembacaan berlatensi rendah. Anda dapat menempatkan salinan data lebih dekat ke pengguna akhir untuk mengurangi latensi jaringan selama pembacaan. Cache tetap baru seperti nilai `ReplicationLatency`.
- Penulisan berlatensi lebih rendah. Anda dapat menulis ke wilayah terdekat untuk mengurangi latensi jaringan dan waktu yang dibutuhkan untuk menghasilkan penulisan. Lalu lintas tulis harus dirutekan dengan hati-hati untuk memastikan tidak ada konflik. Teknik perutean dibahas lebih detail di [Meminta perutean dengan tabel global](#).
- Peningkatan ketahanan dan pemulihan bencana. Anda dapat mengevakuasi suatu Wilayah (memindahkan beberapa atau semua permintaan ke Wilayah tersebut) jika Wilayah tersebut mengalami penurunan performa atau penghentian total, dengan sasaran titik pemulihan (RPO) dan sasaran waktu pemulihan (RTO) diukur dalam detik. Penggunaan tabel global juga meningkatkan [SLA DynamoDB](#) dari 99,99% menjadi 99,999%.
- Migrasi Wilayah yang mulus. Anda dapat menambahkan Wilayah baru lalu menghapus Wilayah lama untuk memigrasikan deployment dari satu Wilayah ke Wilayah lainnya, tanpa adanya waktu henti pada lapisan data. Misalnya, Anda dapat menggunakan tabel global DynamoDB agar sistem manajemen pesanan mencapai pemrosesan latensi rendah yang andal dalam skala tinggi sekaligus menjaga ketahanan terhadap kegagalan AZ dan Regional.

Mode tulis dengan tabel global

Tabel global selalu aktif-aktif di tingkat tabel. Namun, Anda dapat memperlakukannya sebagai aktif-pasif dengan mengontrol cara Anda merutekan permintaan tulis. Misalnya, Anda dapat memutuskan untuk merutekan permintaan tulis ke satu Wilayah untuk menghindari potensi konflik penulisan.

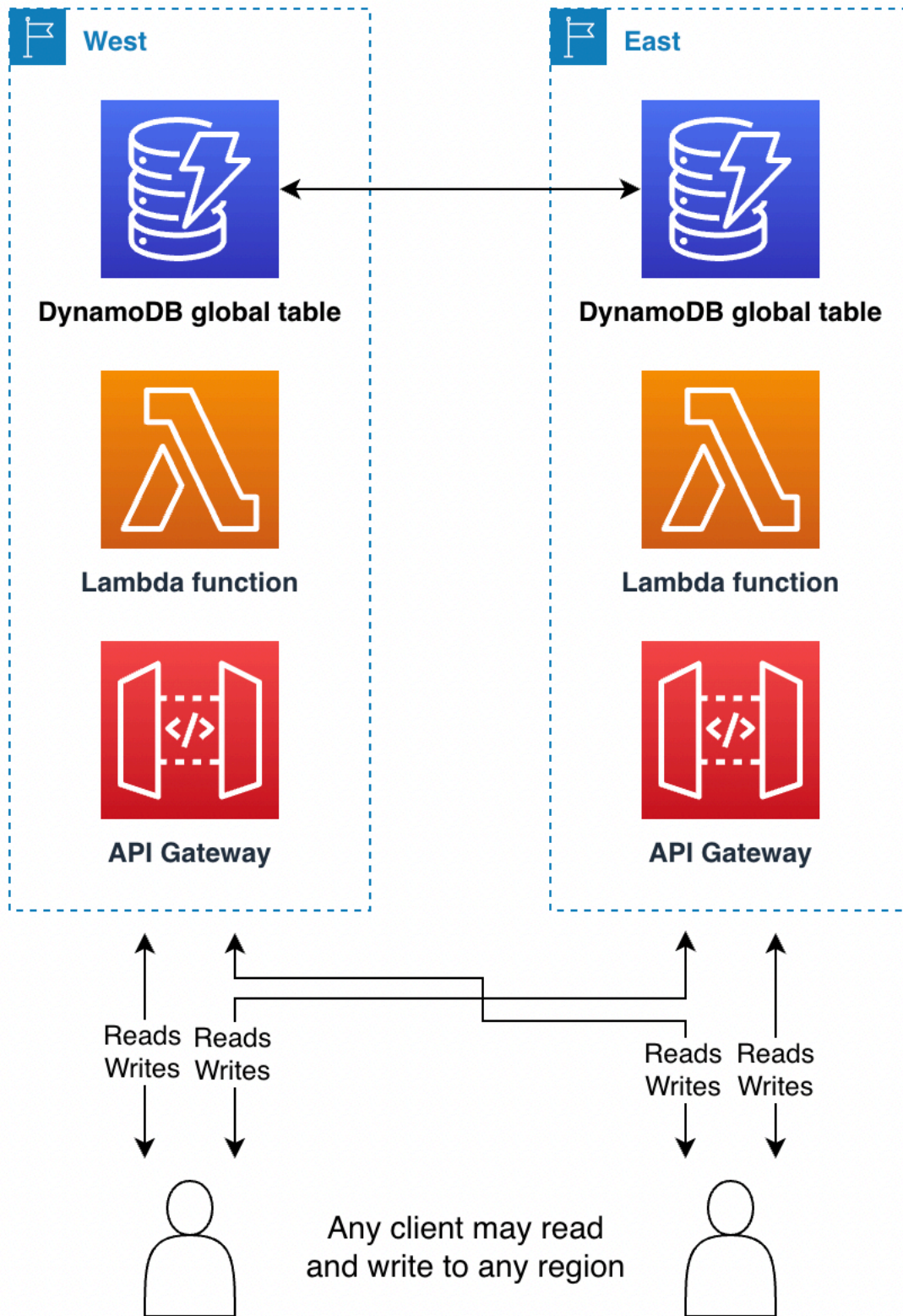
Ada tiga kategorisasi utama pola penulisan terkelola:

- Mode tulis ke Wilayah mana pun (tidak ada primer)
- Mode tulis ke satu Wilayah (primer tunggal)
- Mode penulisan ke Wilayah Anda (primer campuran)

Anda harus mempertimbangkan pola penulisan yang sesuai dengan kasus penggunaan Anda. Pilihan ini memengaruhi cara Anda merutekan permintaan, mengevakuasi suatu Wilayah, dan menangani pemulihan bencana. Praktik terbaik secara keseluruhan dapat berbeda tergantung pada mode tulis aplikasi Anda.

Mode tulis ke Wilayah mana pun (tidak ada primer)

Mode tulis ke Wilayah mana pun sepenuhnya aktif-aktif dan tidak memberlakukan batasan tempat penulisan dapat terjadi. Wilayah mana pun dapat menerima penulisan kapan saja. Ini adalah mode yang paling sederhana. Mode ini hanya dapat digunakan dengan beberapa jenis aplikasi. Ini cocok jika semua penulis idempoten, dan oleh karena itu dapat diulang dengan aman sehingga operasi tulis secara bersamaan atau berulang di seluruh Wilayah tidak menimbulkan konflik. Misalnya, saat pengguna memperbarui data kontak mereka. Mode ini juga berfungsi dengan baik untuk kasus khusus idempoten, kumpulan data khusus tambahan yang semua penulisannya merupakan sisipan unik dengan kunci primer deterministik. Terakhir, mode ini cocok jika risiko penulisan yang bertentangan dapat diterima.



Mode tulis ke Wilayah mana pun adalah arsitektur yang paling mudah untuk diimplementasikan. Perutean lebih mudah karena Wilayah mana pun dapat menjadi target penulisan kapan saja. Failover

lebih mudah, karena setiap penulisan terbaru dapat diputar ulang beberapa kali ke Wilayah sekunder mana pun. Jika memungkinkan, Anda harus merancang mode tulis ini.

Misalnya, layanan streaming video sering kali menggunakan tabel global untuk melacak bookmark, ulasan, bendera status tontonan, dan sebagainya. Deployment ini dapat menggunakan mode tulis ke Wilayah mana pun selama setiap tulis bersifat idempoten dan nilai benar berikutnya untuk suatu item tidak bergantung pada nilainya saat ini. Hal ini berlaku untuk pembaruan pengguna yang menetapkan status baru pengguna secara langsung, seperti mengatur kode waktu terbaru, menetapkan ulasan baru, atau menyetel status tontonan baru. Jika permintaan tulis pengguna dirutekan ke Wilayah lain, operasi tulis terakhir akan tetap ada dan status global akan diselesaikan sesuai dengan penetapan terakhir. Operasi baca dalam mode ini pada akhirnya akan menjadi konsisten, setelah tertunda oleh nilai `ReplicationLatency` terbaru.

Contoh lain, sebuah perusahaan jasa keuangan menggunakan tabel global sebagai bagian dari sistem untuk menyimpan penghitungan pembelian kartu debit untuk setiap pelanggan, untuk menghitung hadiah cash-back pelanggan tersebut. Transaksi baru mengalir dari seluruh dunia dan masuk ke berbagai Wilayah. Untuk desain mereka saat ini yang tidak memanfaatkan tabel global, mereka menggunakan satu `RunningBalance` item per pelanggan. Tindakan pelanggan memperbarui saldo dengan ekspresi `ADD`, yang tidak idempoten karena nilai baru yang benar bergantung pada nilai saat ini. Ini berarti saldo menjadi tidak sinkron jika ada dua operasi tulis ke saldo yang sama pada waktu yang hampir bersamaan di Wilayah berbeda.

Perusahaan yang sama ini dapat menerapkan mode tulis ke Wilayah mana pun melalui desain ulang yang cermat dengan tabel global DynamoDB. Desain baru ini dapat mengikuti model "streaming peristiwa" - yang pada dasarnya merupakan buku besar dengan alur kerja tambah-saja. Setiap tindakan pelanggan menambahkan item baru ke koleksi item yang dikelola untuk pelanggan tersebut. Koleksi item adalah kumpulan item yang memiliki kunci primer yang sama, tetapi kunci urutannya berbeda. Setiap tindakan tulis yang menambahkan tindakan pelanggan merupakan penyisipan idempoten, menggunakan ID pelanggan sebagai kunci partisi dan ID transaksi sebagai kunci urutan. Desain ini membuat penghitungan saldo menjadi lebih rumit, karena memerlukan `Query` untuk menarik item yang diikuti oleh beberapa perhitungan sisi klien. Namun, keuntungannya adalah membuat semua penulisan menjadi idempoten, sehingga memberikan penyederhanaan perutean dan failover yang signifikan. Untuk mengetahui informasi selengkapnya, lihat [Meminta perutean dengan tabel global](#).

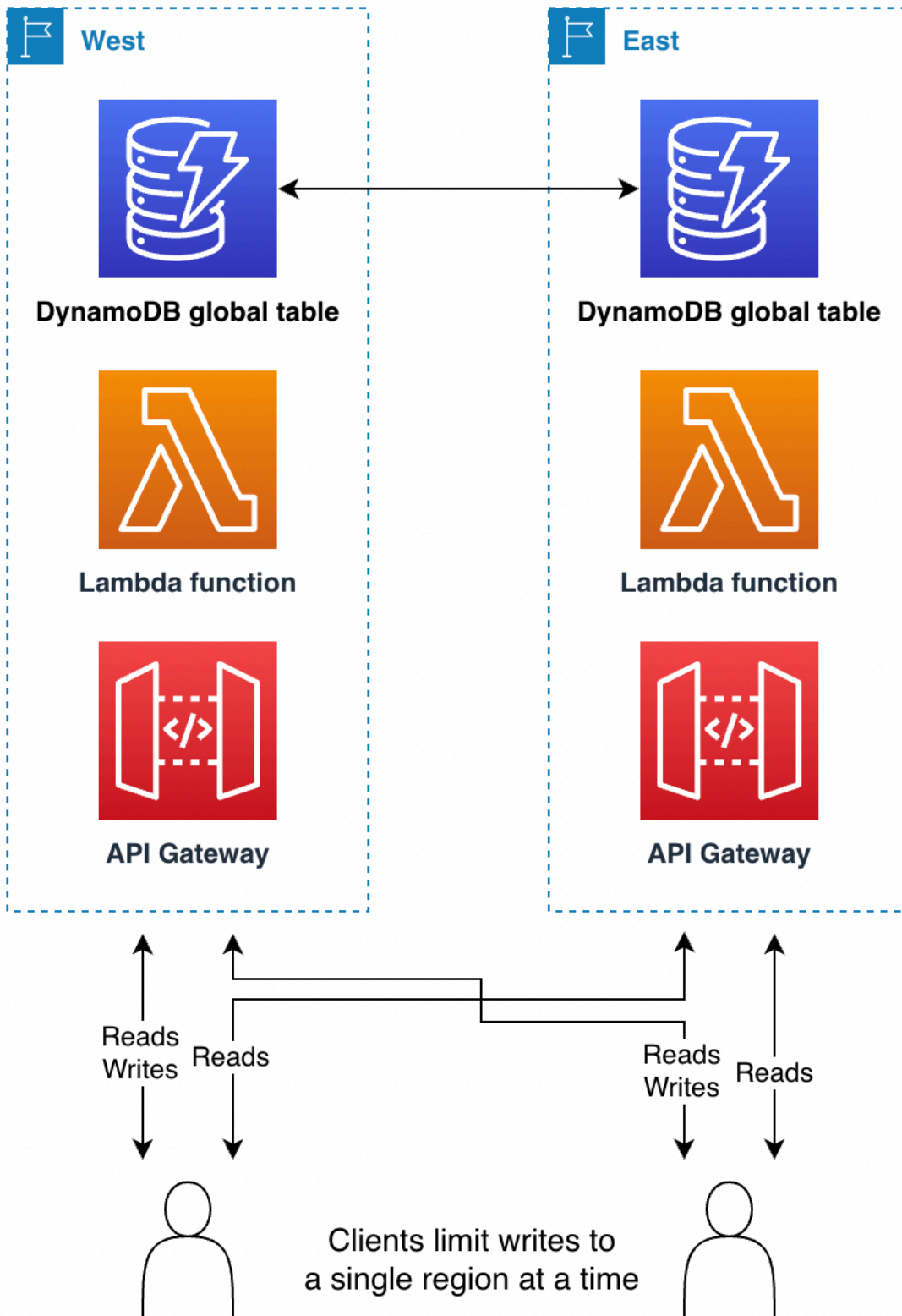
Contoh ketiga, katakanlah ada pelanggan yang menempatkan iklan online. Mereka telah memutuskan risiko rendah kehilangan data akan dapat diterima untuk menghasilkan penyederhanaan desain mode tulis Wilayah mana pun. Saat menayangkan iklan, mereka hanya

memiliki waktu beberapa milidetik untuk mengambil metadata yang cukup guna menentukan iklan yang akan ditampilkan, lalu mencatat tayangan iklan sehingga iklan yang sama tidak terulang kembali kepada pengguna tersebut. Dengan tabel global, mereka bisa mendapatkan pembacaan latensi rendah untuk pengguna akhir di seluruh dunia dan penulisan latensi rendah. Mereka dapat mencatat semua tayangan iklan untuk pengguna dalam satu item, dan menampilkannya sebagai Daftar yang terus bertambah. Mereka dapat menggunakan satu item alih-alih menambahkan ke koleksi item, karena dengan cara ini mereka dapat menghapus tayangan iklan lama yang merupakan bagian dari setiap penulisan tanpa membayar biaya penghapusan. Operasi tulis ini TIDAK idempoten, jadi jika pengguna akhir yang sama melihat iklan ditayangkan di beberapa Wilayah pada waktu yang kurang lebih bersamaan, ada kemungkinan satu penulisan tayangan iklan dapat menimpa yang lain. Untuk penempatan iklan online, risiko bahwa pengguna terkadang melihat iklan berulang-ulang sepadan dengan desain yang lebih sederhana dan efisien ini.

Primer tunggal (“Tulis ke satu Wilayah”)

Mode tulis ke satu Wilayah bersifat aktif-pasif dan merutekan semua penulisan tabel ke satu wilayah aktif. Perhatikan bahwa DynamoDB tidak memiliki gagasan tentang satu wilayah aktif; perutean aplikasi di luar DynamoDB mengatur ini. Mode tulis ke satu Wilayah menghindari konflik penulisan dengan memastikan penulisan hanya mengalir ke satu Wilayah pada satu waktu. Mode tulis ini berguna ketika Anda ingin menggunakan ekspresi atau transaksi bersyarat, karena keduanya tidak akan berfungsi kecuali Anda mengetahui bahwa Anda bertindak berdasarkan data terbaru. Jadi, menggunakan ekspresi dan transaksi bersyarat memerlukan pengiriman semua permintaan tulis ke satu Wilayah dengan data terbaru.

Bacaan akhir konsisten dapat dilakukan di Wilayah replika mana pun untuk mendapatkan latensi yang lebih rendah. Bacaan sangat konsisten harus ditujukan ke satu wilayah utama.



Terkadang Wilayah aktif perlu diubah sebagai respons terhadap kegagalan Regional, untuk membantu data. [Mengevakuasi Wilayah dengan tabel global](#) adalah salah satu contoh kasus penggunaan ini. Beberapa pelanggan akan mengubah Wilayah yang aktif saat ini dengan jadwal reguler, seperti deployment "follow-the-sun". Hal ini menempatkan Wilayah aktif di dekat geografi dengan aktivitas terbanyak, sehingga memberikan latensi baca dan tulis terendah. Hal ini juga memiliki keuntungan tambahan dengan memanggil jalur kode yang mengubah Wilayah setiap hari, memastikan jalur tersebut telah diuji dengan baik sebelum pemulihan bencana apa pun.

Wilayah pasif dapat mempertahankan serangkaian infrastruktur yang diperkecil di sekitar DynamoDB yang akan dibangun hanya jika wilayah tersebut menjadi wilayah aktif. Untuk diskusi yang lebih mendalam tentang desain pilot light dan warm standby, lihat [Disaster Recovery \(DR\) Architecture on AWS, Bagian III: Pilot Light and Warm Standby](#).

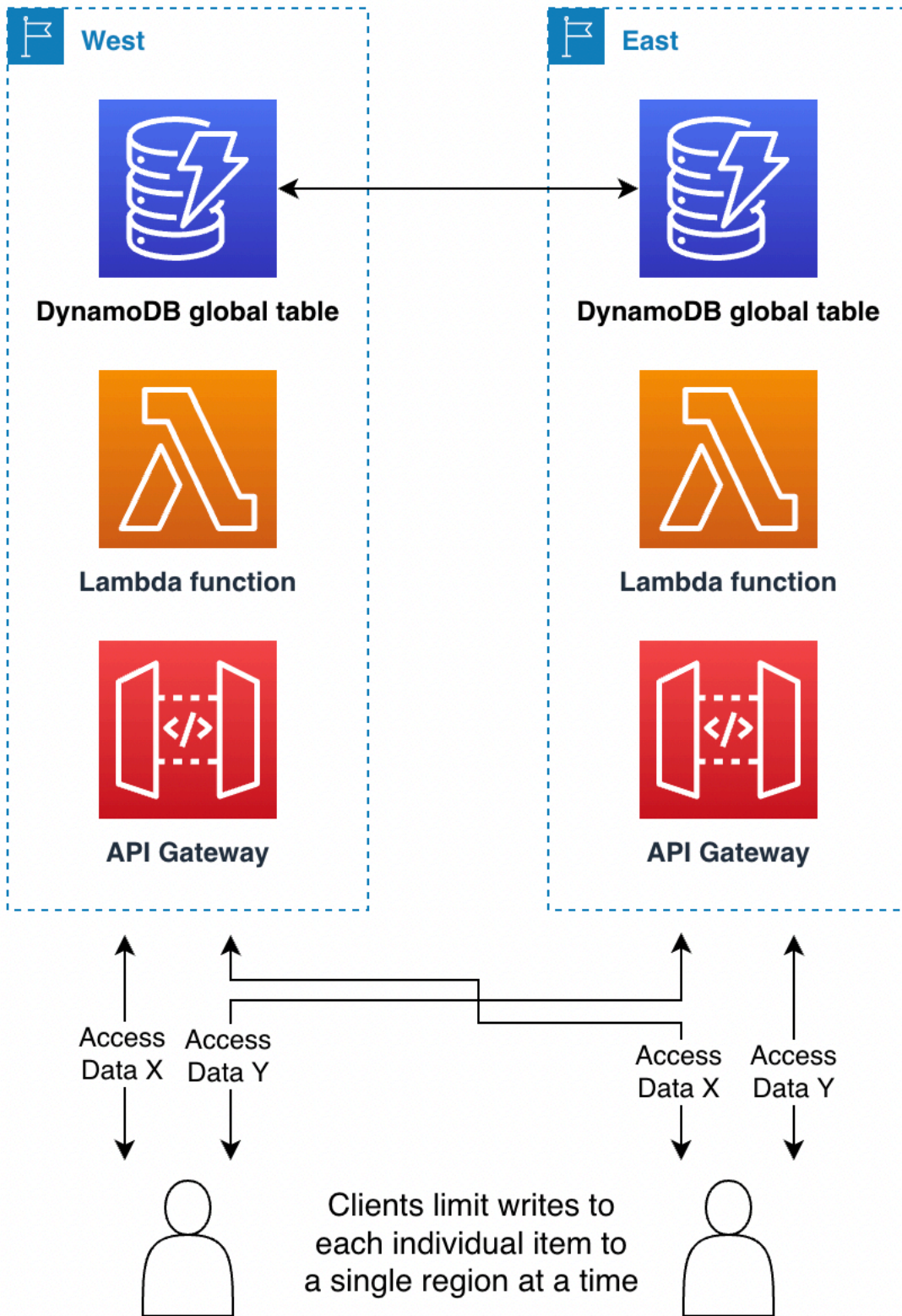
Menggunakan mode tulis ke satu Wilayah berfungsi dengan baik saat memanfaatkan tabel global untuk pembacaan terdistribusi global dengan latensi rendah. Misalnya, sebuah perusahaan media sosial besar memiliki jutaan pengguna dan miliaran postingan. Setiap pengguna ditetapkan ke suatu Wilayah pada saat pembuatan akun, yang ditempatkan dekat dengan lokasi mereka secara geografis. Semua data mereka berada di tabel non-global itu. Perusahaan menggunakan tabel global terpisah untuk menyimpan pemetaan pengguna ke wilayah asal mereka, menggunakan mode tulis ke satu Wilayah. Ini menyimpan salinan hanya-baca di seluruh dunia untuk membantu menemukan lokasi setiap data pengguna secara langsung dengan latensi tambahan minimum. Pembaruan jarang terjadi (hanya ketika memindahkan Wilayah asal pengguna dari satu Wilayah ke Wilayah lain) dan selalu melalui satu Wilayah untuk penulisan, guna menghindari kemungkinan konflik penulisan.

Contoh lainnya, perhatikan pelanggan jasa keuangan yang menerapkan penghitungan cash-back harian. Mereka menggunakan mode tulis ke Wilayah mana pun untuk menghitung saldo tetapi menggunakan mode tulis ke satu Wilayah untuk melacak pembayaran cash-back yang sebenarnya. Jika mereka ingin memberikan hadiah kepada pelanggan sebesar 1 sen untuk setiap \$10 yang dibelanjakan dalam sehari, mereka perlu melakukan Query untuk semua transaksi dari hari sebelumnya, menghitung total pembelanjaan, menulis keputusan cash-back ke tabel baru, menghapus kumpulan item yang dikueri untuk menandainya sebagai digunakan, dan menggantinya dengan item tunggal yang menyimpan jumlah sisa yang harus dimasukkan ke dalam penghitungan hari berikutnya. Pekerjaan ini memerlukan transaksi, sehingga akan lebih baik dengan mode tulis ke satu Wilayah. Suatu aplikasi dapat menggabungkan mode penulisan, bahkan pada tabel yang sama, selama beban kerja tidak memiliki peluang untuk tumpang tindih.

Primer campuran (“Tulis ke Wilayah Anda”)

Mode tulis ke Wilayah Anda menetapkan subset data yang berbeda ke Wilayah yang berbeda dan hanya mengizinkan operasi tulis pada item melalui wilayah asalnya. Mode ini aktif-pasif tetapi menetapkan Wilayah aktif berdasarkan item. Setiap Wilayah adalah yang utama untuk kumpulan datanya yang tidak tumpang tindih, dan penulisan harus dijaga untuk memastikan lokalitas yang tepat.

Mode ini mirip dengan mode tulis ke satu Wilayah, hanya saja mode ini memungkinkan penulisan dengan latensi lebih rendah, karena data yang terkait dengan setiap pengguna akhir dapat ditempatkan di jaringan yang lebih dekat dengan pengguna tersebut. Mode ini juga menyebarkan infrastruktur sekitar secara lebih merata di antara Wilayah dan memerlukan lebih sedikit pekerjaan untuk membangun infrastruktur selama skenario failover, karena sebagian infrastruktur di semua wilayah sudah aktif.



Menentukan wilayah asal untuk item dapat dilakukan dengan berbagai cara:

- **Intrinsik:** Beberapa aspek data memperjelas Wilayah tempat data tersebut berada, seperti kunci partisinya. Misalnya, pelanggan dan semua data tentang pelanggan tersebut akan ditandai dalam data pelanggan sebagai berasal dari wilayah tertentu. Teknik ini dijelaskan dalam [Menggunakan penyematan Wilayah untuk menetapkan Wilayah asal item dalam tabel global Amazon DynamoDB](#)
- **Negosiasi:** Wilayah asal setiap kumpulan data dinegosiasikan dengan cara eksternal, seperti dengan layanan global terpisah yang mempertahankan penetapan. Penetapan tersebut mungkin memiliki jangka waktu terbatas dan setelah itu harus dinegosiasikan ulang.
- **Berorientasi tabel:** Daripada hanya mereplikasi satu tabel global, buatlah tabel global sebanyak jumlah Wilayah yang direplikasi. Nama setiap tabel menunjukkan Wilayah asalnya. Dalam operasi standar, semua data ditulis ke Wilayah asal sementara Wilayah lain menyimpan salinan hanya-baca. Selama failover, Wilayah lain untuk sementara waktu akan menerapkan tugas penulisan untuk tabel tersebut.

Misalnya, anggaplah Anda bekerja di perusahaan game. Anda memerlukan pembacaan dan penulisan dengan latensi rendah untuk semua gamer di seluruh dunia. Anda dapat menempatkan setiap gamer di Wilayah terdekat mereka. Wilayah itu mengambil semua bacaan dan tulisan mereka, memastikan selalu ada read-after-write konsistensi yang kuat. Namun, jika gamer tersebut melakukan perjalanan atau Wilayah asal mereka mengalami gangguan, salinan lengkap data mereka akan tersedia di Wilayah alternatif. Jadi, gamer dapat ditetapkan ke Wilayah asal yang berbeda sesuai kebutuhan.

Contoh lainnya, katakanlah Anda bekerja di perusahaan konferensi video. Setiap metadata panggilan konferensi ditetapkan ke Wilayah tertentu. Pemanggil dapat menggunakan Wilayah yang paling dekat dengan mereka untuk mendapatkan latensi terendah. Jika terjadi gangguan Wilayah, penggunaan tabel global memungkinkan pemulihan cepat karena sistem dapat memindahkan pemrosesan panggilan ke Wilayah lain yang sudah terdapat salinan data yang direplikasi.

Meminta perutean dengan tabel global

Mungkin bagian paling rumit dari deployment tabel global adalah mengelola perutean permintaan. Permintaan harus dikirim dari pengguna akhir ke Wilayah yang dipilih terlebih dahulu, lalu dirutekan dengan cara tertentu. Permintaan menemukan beberapa tumpukan layanan di Wilayah tersebut, termasuk lapisan komputasi yang mungkin terdiri dari penyeimbang beban yang didukung oleh AWS Lambda fungsi, wadah, atau node Amazon Elastic Compute Cloud (Amazon EC2), dan mungkin layanan lain termasuk database lain. Lapisan komputasi tersebut berkomunikasi dengan DynamoDB yang dilakukan menggunakan titik akhir lokal untuk Wilayah tersebut. Data dalam tabel global

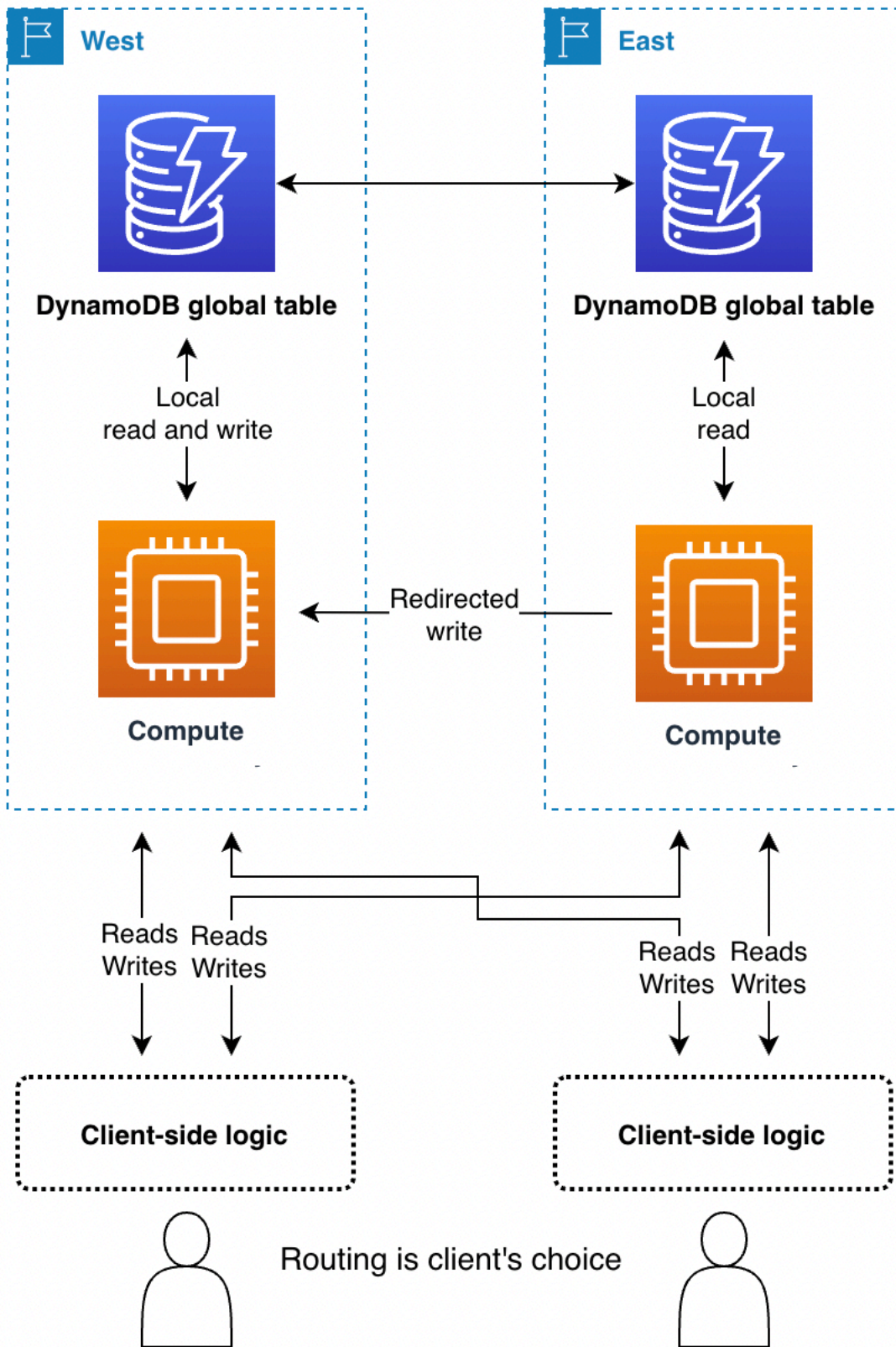
direplikasi ke semua Wilayah lain yang berpartisipasi, dan setiap Wilayah memiliki tumpukan layanan serupa di sekitar tabel DynamoDB-nya.

Tabel global menyediakan salinan lokal dari data yang sama untuk setiap tumpukan di berbagai Wilayah. Sebaiknya Anda merancang tumpukan tunggal dalam satu Wilayah dan mengantisipasi melakukan panggilan jarak jauh ke titik akhir DynamoDB Wilayah sekunder jika tabel DynamoDB lokal mengalami masalah. Ini bukan praktik terbaik. Latensi yang terkait dengan lintas Wilayah mungkin 100 kali lebih tinggi dibandingkan akses lokal. back-and-forth Serangkaian 5 permintaan mungkin membutuhkan milidetik saat dilakukan secara lokal tetapi detik saat melintasi dunia. Sebaiknya rutekan pengguna akhir ke Wilayah lain untuk diproses. Untuk memastikan ketahanan, Anda memerlukan replikasi di beberapa Wilayah, dengan replikasi lapisan komputasi serta lapisan data.

Ada banyak teknik alternatif untuk merutekan permintaan pengguna akhir ke suatu Wilayah untuk diproses. Pilihan optimal bergantung pada mode tulis dan pertimbangan failover Anda. Bagian ini membahas empat opsi: berbasis klien, lapisan komputasi, Route 53, dan Global Accelerator.

Perutean permintaan berbasis klien

Dengan routing permintaan berbasis klien, klien pengguna akhir seperti aplikasi, halaman web dengan klien JavaScript lain akan melacak titik akhir aplikasi yang valid. Hal tersebut akan menjadi titik akhir aplikasi seperti Amazon API Gateway, bukan titik akhir DynamoDB literal. Klien pengguna akhir menggunakan logika tertanamnya sendiri untuk memilih Wilayah yang akan diajak berkomunikasi. Pengguna akhir tersebut dapat memilih berdasarkan pemilihan acak, latensi terendah yang diamati, pengukuran bandwidth tertinggi yang diamati, atau pemeriksaan kesehatan yang dilakukan secara lokal.



Keuntungan perutean permintaan berbasis klien adalah dapat adaptif terhadap hal-hal seperti kondisi lalu lintas internet publik di dunia nyata untuk berpindah Wilayah jika terjadi penurunan performa. Klien harus mengetahui semua titik akhir potensial, tetapi peluncuran titik akhir Regional baru bukanlah hal yang sering terjadi.

Dengan mode tulis ke Wilayah mana pun, klien dapat memilih titik akhir pilihannya secara sepihak. Jika aksesnya ke satu Wilayah terganggu, klien dapat merutekan ke titik akhir lain.

Dengan mode tulis ke satu Wilayah, klien memerlukan mekanisme untuk merutekan penulisannya ke wilayah yang sedang aktif. Hal ini dapat berupa pengujian empiris terhadap wilayah mana yang saat ini menerima penulisan (mencatat setiap penolakan penulisan dan kembali ke alternatif) atau rumit seperti memanggil koordinator global untuk mengkueri status aplikasi saat ini (mungkin dibangun pada kontrol perutean Pengendali Pemulihan Aplikasi (ARC) Route 53 yang menyediakan sistem berbasis kuorum 5 wilayah untuk mempertahankan status global untuk kebutuhan semacam ini). Klien dapat memutuskan apakah pembacaan dapat dikirim ke Wilayah mana pun untuk konsistensi akhir atau harus dirutekan ke wilayah aktif untuk konsistensi yang kuat. Untuk informasi selengkapnya, lihat [Cara kerja Route 53](#).

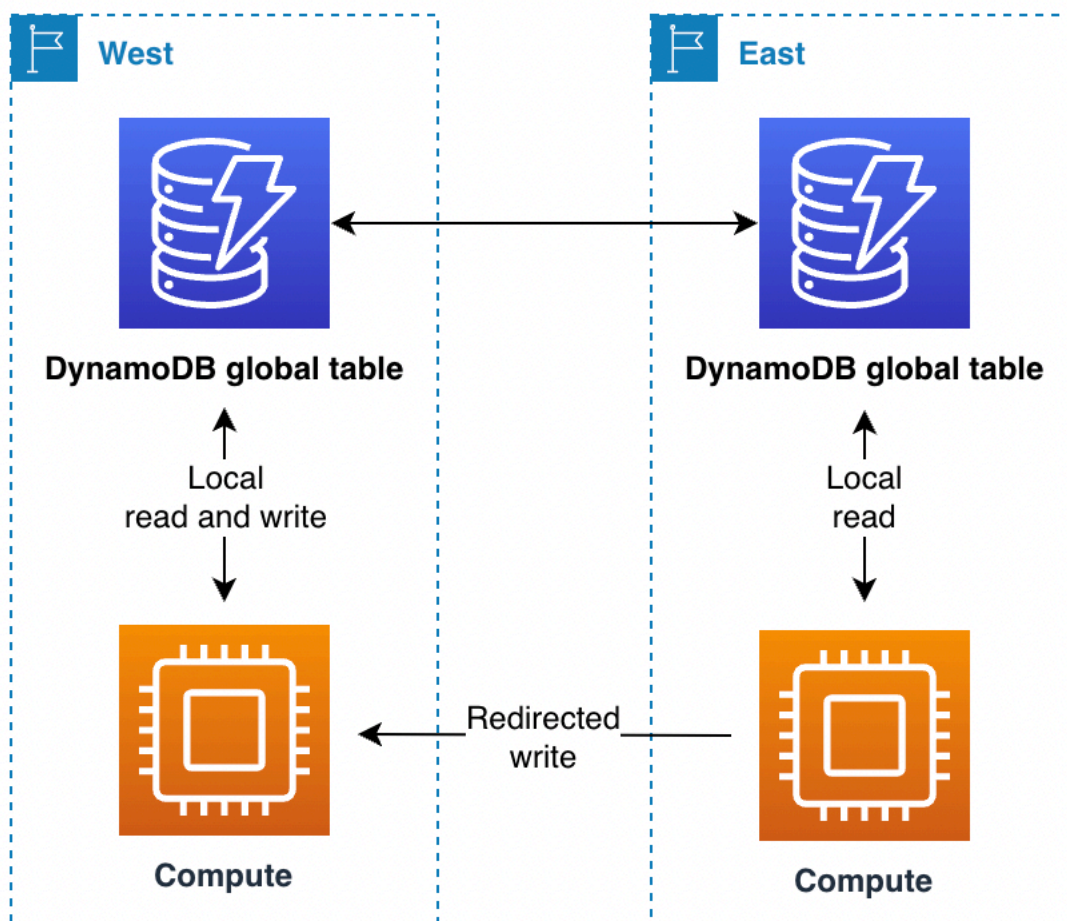
Dengan mode tulis ke Wilayah Anda, klien perlu menentukan wilayah asal kumpulan data yang digunakannya. Misalnya, jika klien berhubungan dengan akun pengguna dan setiap akun pengguna ditempatkan di suatu Wilayah, klien dapat meminta titik akhir yang sesuai dari sistem login global.

Misalnya, perusahaan jasa keuangan yang membantu pengguna mengelola keuangan bisnisnya melalui web dapat menggunakan tabel global dengan mode tulis ke Wilayah Anda. Setiap pengguna harus masuk ke layanan pusat. Layanan tersebut mengembalikan kredensial dan titik akhir untuk Wilayah tempat kredensial tersebut akan berfungsi. Kredensialnya valid untuk waktu yang singkat. Setelah itu, halaman web otomatis menegosiasikan aktivitas masuk baru, yang memberikan peluang untuk mengalihkan aktivitas pengguna ke Wilayah baru.

Perutean permintaan lapisan komputasi

Dengan perutean permintaan lapisan komputasi, kode yang berjalan di lapisan komputasi memutuskan apakah kode akan memproses permintaan secara lokal, atau meneruskannya ke salinannya sendiri yang berjalan di Wilayah lain. Saat Anda menggunakan mode tulis ke satu Wilayah, lapisan komputasi dapat mendeteksi bahwa itu bukan wilayah yang aktif dan memungkinkan operasi baca lokal sekaligus meneruskan semua operasi tulis ke Wilayah lain. Kode lapisan komputasi ini harus mengetahui topologi data dan aturan perutean, serta menerapkannya secara andal berdasarkan pengaturan terbaru yang menentukan Wilayah yang aktif untuk data tertentu. Tumpukan perangkat lunak luar di dalam Wilayah tidak harus mengetahui bagaimana permintaan

baca dan tulis dirutekan oleh layanan mikro. Dalam desain yang kuat, Wilayah penerima memvalidasi apakah Wilayah tersebut merupakan wilayah primer saat ini untuk operasi tulis. Jika tidak, maka akan muncul kesalahan yang menunjukkan bahwa status global perlu diperbaiki. Wilayah penerima mungkin juga melakukan buffering pada operasi tulis untuk sementara waktu jika Wilayah utama sedang dalam proses perubahan. Dalam semua kasus, tumpukan komputasi di suatu Wilayah hanya menulis ke titik akhir DynamoDB lokalnya, tetapi tumpukan komputasi tersebut mungkin berkomunikasi satu sama lain.

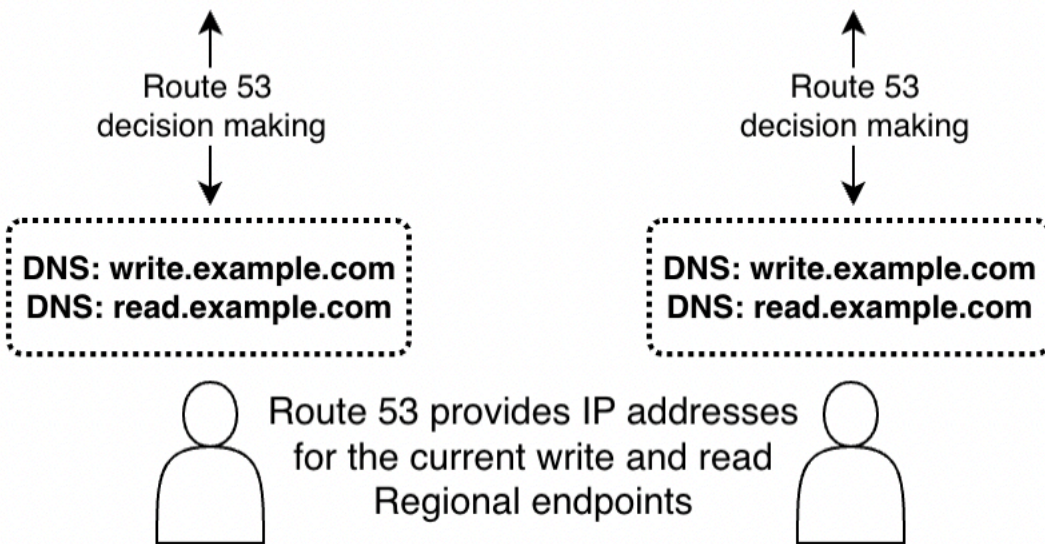
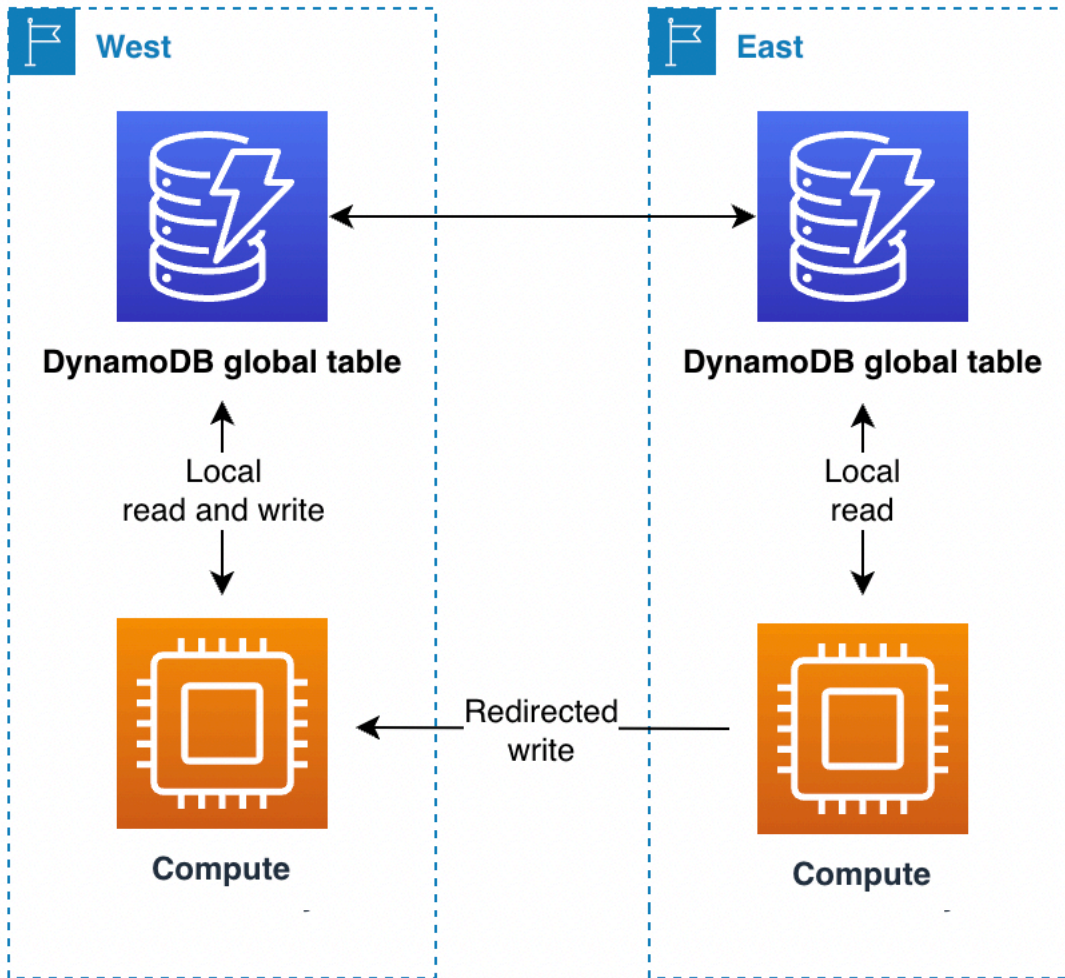


Dalam skenario ini, katakanlah perusahaan jasa keuangan menggunakan model follow-the-sun Single Primary. Mereka menggunakan sistem dan pustaka untuk proses perutean ini. Sistem mereka secara keseluruhan mempertahankan keadaan global, mirip AWS dengan Route 53 ARC routing control. Mereka menggunakan tabel global untuk melacak Wilayah yang merupakan Wilayah utama, dan kapan peralihan utama berikutnya dijadwalkan. Semua operasi baca dan tulis melalui pustaka, yang berkoordinasi dengan sistemnya. Pustaka memungkinkan operasi baca dilakukan secara lokal dengan latensi rendah. Untuk operasi tulis, aplikasi memeriksa apakah Wilayah lokal adalah Wilayah utama saat ini. Jika ya, operasi tulis selesai secara langsung. Jika tidak, librante meneruskan tugas tulis ke pustaka yang ada di Wilayah utama saat ini. Pustaka penerima tersebut mengonfirmasi

bahwa ia juga menganggap dirinya sebagai Wilayah utama dan menimbulkan kesalahan jika bukan, yang menunjukkan penundaan penyebaran dengan keadaan global. Pendekatan ini memberikan manfaat validasi dengan tidak menulis langsung ke titik akhir DynamoDB jarak jauh.

Perutean permintaan Route 53

Pengendali Pemulihan Aplikasi Amazon Route 53 adalah teknologi Domain Name Service (DNS). Dengan Route 53, klien meminta titik akhir dengan mencari nama domain DNS yang dikenal, dan Route 53 mengembalikan alamat IP yang sesuai dengan titik akhir regional yang dianggap paling tepat. Route 53 memiliki [daftar kebijakan perutean yang digunakan untuk menentukan wilayah yang sesuai](#). Route 53 juga dapat melakukan [perutean failover untuk merutekan lalu lintas dari wilayah yang gagal dalam pemeriksaan kondisi](#).



- Dengan mode tulis ke Wilayah mana pun, atau jika digabungkan dengan perutean permintaan lapisan komputasi di backend, Route 53 dapat diberi akses penuh untuk mengembalikan Wilayah berdasarkan aturan internal kompleks seperti Wilayah dalam kedekatan jaringan terdekat, atau kedekatan geografis terdekat, atau pilihan lainnya.
- Dengan mode tulis ke satu Wilayah, Route 53 dapat dikonfigurasi untuk mengembalikan wilayah yang saat ini aktif (menggunakan Route 53 ARC).

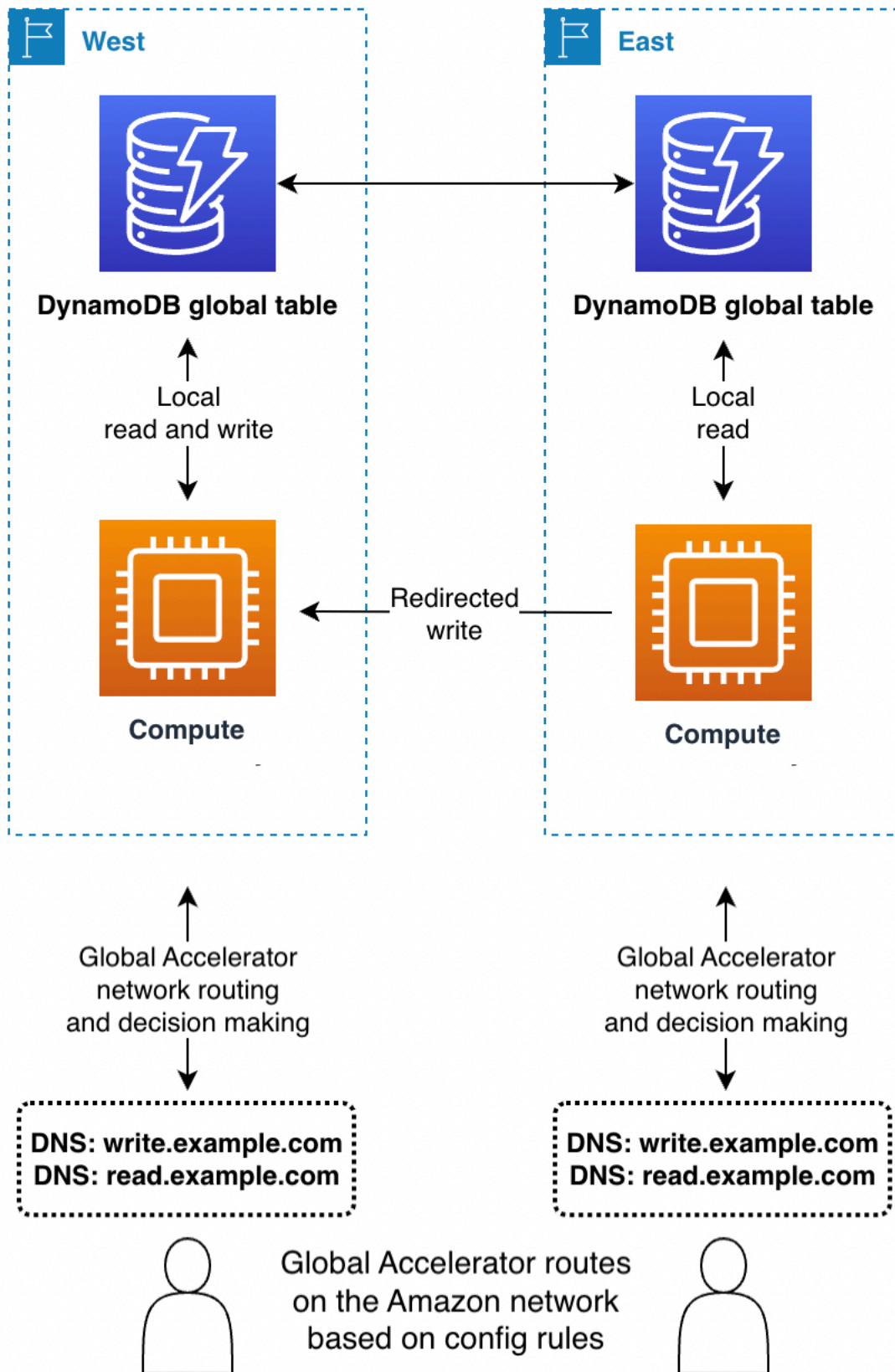
Note

Klien menyimpan alamat IP dalam respons dari Route 53 selama waktu yang ditunjukkan oleh pengaturan waktu untuk tayang (TTL) pada nama domain. TTL yang lebih panjang memperpanjang sasaran waktu pemulihan (RTO) bagi semua klien untuk mengenali titik akhir baru. Nilai 60 detik adalah tipikal untuk penggunaan failover. Tidak semua perangkat lunak mematuhi masa berlaku TTL DNS dengan sempurna.

- Dengan mode tulis ke Wilayah Anda, sebaiknya hindari Route 53 kecuali Anda juga menggunakan perutean permintaan lapisan komputasi.

Perutean permintaan Global Accelerator

Klien menggunakan [AWS Global Accelerator](#) untuk mencari nama domain yang dikenal di Route 53. Namun, alih-alih mendapatkan kembali alamat IP yang sesuai dengan titik akhir regional, klien menerima alamat IP statis anycast yang merutekan ke lokasi AWS tepi terdekat. Mulai dari lokasi tepi itu, semua lalu lintas dirutekan di AWS jaringan pribadi dan ke beberapa titik akhir (seperti penyeimbang beban atau API Gateway) di Wilayah yang dipilih oleh aturan perutean yang dipertahankan dalam Global Accelerator. Dibandingkan dengan perutean berdasarkan aturan Route 53, perutean permintaan Global Accelerator memiliki latensi yang lebih rendah karena mengurangi jumlah lalu lintas di internet publik. Selain itu, karena Global Accelerator tidak bergantung pada masa berlaku TTL DNS untuk mengubah aturan perutean, perutean dapat disesuaikan dengan lebih cepat.



- Dengan mode tulis ke Wilayah mana pun, atau jika dikombinasikan dengan perutean permintaan lapisan komputasi di back-end, Global Accelerator dapat bekerja dengan lancar. Klien terhubung ke lokasi edge terdekat dan tidak perlu khawatir dengan Wilayah mana yang menerima permintaan.
- Dengan tulis ke satu Wilayah, aturan perutean Global Accelerator harus mengirimkan permintaan ke Wilayah yang sedang aktif. Anda dapat menggunakan pemeriksaan kondisi yang secara artifisial melaporkan kegagalan pada Wilayah yang tidak diperhitungkan sebagai Wilayah aktif oleh sistem global Anda. Seperti halnya DNS, nama domain DNS alternatif dapat digunakan untuk merutekan permintaan baca jika permintaan tersebut dapat berasal dari Wilayah mana pun.
- Dengan mode tulis ke Wilayah Anda, sebaiknya hindari Global Accelerator kecuali Anda juga menggunakan perutean permintaan lapisan komputasi.

Mengevakuasi Wilayah dengan tabel global

Evakuasi suatu Wilayah adalah proses memigrasikan aktivitas baca dan tulis keluar dari Wilayah tersebut. Ini paling sering merupakan aktivitas tulis, dan terkadang aktivitas baca.

Mengevakuasi Wilayah aktif

Anda dapat memutuskan untuk mengevakuasi Wilayah aktif karena sejumlah alasan. Penghindaran dapat menjadi bagian dari aktivitas bisnis biasa seperti jika Anda menggunakan mode follow-the-sun tulis ke satu Wilayah. Evakuasi juga bisa disebabkan oleh keputusan bisnis untuk mengubah Wilayah yang sedang aktif, sebagai respons terhadap kegagalan tumpukan perangkat lunak di luar DynamoDB, atau karena Anda menghadapi masalah umum seperti latensi yang lebih tinggi dari biasanya di dalam Wilayah.

Dengan mode tulis ke Wilayah mana pun, mengevakuasi Wilayah aktif sangatlah mudah. Anda dapat merutekan lalu lintas ke Wilayah alternatif melalui sistem perutean apa pun, dan membiarkan operasi tulis yang telah terjadi di Wilayah yang dievakuasi direplikasi seperti biasa.

Dengan mode tulis ke satu Wilayah dan tulis ke Wilayah Anda, Anda harus memastikan semua penulisan ke Wilayah aktif telah direkam sepenuhnya, diproses streaming, dan disebar secara global sebelum memulai penulisan di Wilayah aktif yang baru. Hal ini diperlukan untuk memastikan bahwa penulisan di masa mendatang bertentangan dengan versi data terbaru.

Katakanlah Wilayah A aktif dan Wilayah B pasif (baik untuk tabel lengkap atau untuk item yang ditempatkan di Wilayah A). Mekanisme umum untuk melakukan evakuasi adalah dengan menjeda

operasi tulis ke A, menunggu cukup lama hingga operasi tersebut disebarakan sepenuhnya ke B, memperbarui tumpukan arsitektur untuk mengenali B sebagai aktif, lalu melanjutkan operasi tulis ke B. Tidak ada metrik yang menunjukkan kepastian mutlak bahwa Wilayah A telah sepenuhnya mereplikasi datanya ke Wilayah B. Jika Wilayah A sehat, menjeda operasi tulis ke Wilayah A dan menunggu 10 kali nilai maksimum terbaru metrik `ReplicationLatency` biasanya sudah cukup untuk menentukan bahwa replikasi tersebut selesai. Jika Wilayah A tidak sehat dan menunjukkan area lain mengalami peningkatan latensi, Anda dapat memilih kelipatan waktu tunggu yang lebih besar.

Mengevakuasi Wilayah offline

Ada kasus khusus yang perlu dipertimbangkan: bagaimana jika Wilayah A offline sepenuhnya tanpa pemberitahuan? Hal ini sangat kecil kemungkinannya, tetapi tetap perlu dipertimbangkan. Jika hal ini terjadi, operasi tulis apa pun di Wilayah A yang belum disebarakan akan ditahan dan disebarakan setelah Wilayah A kembali online. Operasi tulis tidak hilang, tetapi penyebarannya tertunda tanpa batas waktu.

Cara melanjutkan peristiwa ini merupakan keputusan aplikasi. Untuk kelangsungan bisnis, operasi tulis mungkin perlu diteruskan ke Wilayah B utama yang baru. Namun, jika item di Wilayah B menerima pembaruan sementara ada penyebaran operasi tulis yang tertunda untuk item tersebut dari Wilayah A, penyebaran tersebut akan disembunyikan di model penulis terakhir menang. Pembaruan apa pun di Wilayah B mungkin menyembunyikan permintaan tulis yang masuk.

Dengan mode tulis ke Wilayah mana pun, pembacaan dan penulisan dapat dilanjutkan di Wilayah B, dengan keyakinan bahwa item di Wilayah A pada akhirnya akan disebarakan ke Wilayah B dan mengenali potensi item yang hilang hingga Wilayah A kembali online. Jika memungkinkan, Anda harus mempertimbangkan untuk memutar ulang lalu lintas tulis terbaru (misalnya, dengan menggunakan sumber peristiwa hulu) untuk mengisi celah dari operasi tulis yang mungkin hilang dan membiarkan resolusi konflik penulis terakhir menang menekan penyebaran akhir dari operasi tulis yang masuk.

Dengan mode tulis lainnya, Anda harus mempertimbangkan sejauh mana pekerjaan dapat dilanjutkan dengan sedikit out-of-date pandangan dunia. Beberapa operasi tulis berdurasi pendek, seperti yang dilacak oleh `ReplicationLatency`, akan hilang hingga Wilayah A kembali online. Apakah bisnis bisa maju? Dalam beberapa kasus penggunaan, bisa, tetapi pada kasus lainnya mungkin tidak bisa, tanpa mekanisme mitigasi tambahan.

Misalnya, katakanlah Anda perlu mempertahankan saldo kredit yang tersedia tanpa gangguan bahkan setelah kegagalan Wilayah. Anda dapat membagi saldo menjadi dua item berbeda, satu

ditempatkan di Wilayah A dan satu lagi di Wilayah B, masing-masing dimulai dengan setengah saldo yang tersedia. Ini akan menggunakan mode tulis ke Wilayah Anda. Pembaruan transaksional yang diproses di setiap Wilayah akan dituliskan pada salinan lokal saldo. Jika Wilayah A sepenuhnya offline, pekerjaan masih dapat dilanjutkan dengan pemrosesan transaksi di Wilayah B, dan operasi tulis akan terbatas pada bagian saldo yang disimpan di Wilayah B. Memisahkan saldo seperti ini menimbulkan kerumitan ketika saldo hampir habis atau kredit harus diseimbangkan kembali, tetapi hal ini memberikan satu contoh pemulihan bisnis yang aman bahkan dengan operasi tulis tertunda yang tidak pasti.

Contoh lain, misalkan Anda mengambil data formulir web. Anda dapat menggunakan [Optimistic Concurrency Control \(OCC\)](#) untuk menetapkan versi ke item data dan menyimpan versi terbaru ke formulir web sebagai bidang tersembunyi. Pada setiap pengiriman, operasi tulis berhasil hanya jika versi dalam basis data masih cocok dengan versi yang digunakan untuk membuat formulir. Jika versinya tidak cocok, formulir web bisa disegarkan (atau digabungkan secara hati-hati) berdasarkan versi saat ini dalam basis data, dan pengguna bisa melanjutkan lagi. Model OCC biasanya melindungi terhadap penipaan klien lain dan pembuatan versi data yang baru, tetapi model ini juga dapat membantu selama failover ketika klien mungkin menemukan versi data yang lebih lama.

Misalkan Anda menggunakan timestamp sebagai versinya. Katakanlah formulir pertama kali dibuat pada Wilayah A pada pukul 12.00 tetapi (setelah failover) mencoba menulis ke Wilayah B dan mengetahui bahwa versi terbaru dalam basis data adalah 11.59. Dalam skenario ini, klien dapat menunggu versi 12.00 untuk disebar ke Wilayah B lalu menulis di atas versi tersebut, atau membangun pada 11.59 dan membuat versi 12.01 baru (yang, setelah ditulis, akan menyembunyikan versi yang masuk setelah Wilayah A pulih).

Contoh terakhir, perusahaan jasa keuangan menyimpan data tentang akun pelanggan dan transaksi keuangan mereka dalam basis data DynamoDB. Jika terjadi pemadaman total Wilayah A, mereka ingin memastikan bahwa aktivitas tulis apa pun yang terkait dengan akun mereka tersedia sepenuhnya di Wilayah B, atau ingin menggarantini akun mereka sebagai diketahui sebagian hingga Wilayah A kembali online. Alih-alih menghentikan semua bisnis, mereka memutuskan untuk menghentikan bisnis untuk sementara waktu, hanya pada sebagian kecil akun yang mereka anggap memiliki transaksi yang tidak disebar. Untuk mencapai hal ini, mereka menggunakan Wilayah ketiga, yang akan kita sebut Wilayah C. Sebelum memproses operasi tulis apa pun di Wilayah A, mereka menempatkan ringkasan singkat operasi yang tertunda tersebut (misalnya, jumlah transaksi baru untuk sebuah akun) di Wilayah C. Ringkasan ini cukup bagi Wilayah B untuk menentukan apakah pandangannya benar-benar mutakhir. Tindakan ini mengunci akun secara efektif sejak penulisan di Wilayah C hingga Wilayah A menerima operasi tulis dan Wilayah B menerimanya. Data

di Wilayah C tidak digunakan kecuali sebagai bagian dari proses failover, setelah itu Wilayah B dapat memeriksa datanya dengan Wilayah C untuk mengetahui apakah ada akun yang kedaluwarsa. Akun tersebut akan ditandai sebagai dikarantina hingga pemulihan Wilayah A menyebarkan sebagian data ke Wilayah B.

Jika Wilayah C gagal, Wilayah D baru dapat dibuka untuk digunakan. Data di Wilayah C sangat sementara, dan setelah beberapa menit Wilayah D akan memiliki up-to-date catatan yang cukup tentang operasi penulisan dalam penerbangan untuk sepenuhnya berguna. Jika Wilayah B gagal, Wilayah A dapat terus menerima permintaan tulis yang bekerja sama dengan Wilayah C. Perusahaan ini bersedia menerima penulisan dengan latensi yang lebih tinggi (ke dua Wilayah: C dan kemudian A) dan beruntung memiliki model data yang status akunnya dapat dirangkum secara ringkas.

Perencanaan kapasitas throughput untuk tabel global

Migrasi lalu lintas dari satu Wilayah ke Wilayah lain memerlukan pertimbangan cermat terhadap pengaturan tabel DynamoDB terkait kapasitas.

Beberapa pertimbangan dalam mengelola kapasitas tulis:

- Tabel global harus berada dalam mode sesuai permintaan atau disediakan dengan penskalaan otomatis yang diaktifkan.
- Jika disediakan dengan penskalaan otomatis, pengaturan tulis (pemanfaatan minimum, maksimum, dan target) direplikasi di seluruh Wilayah. Meskipun pengaturan penskalaan otomatis disinkronkan, kapasitas tulis aktual yang disediakan mungkin mengambang secara independen di antara Wilayah.
- Salah satu alasan Anda mungkin melihat perbedaan kapasitas tulis yang disediakan adalah karena fitur TTL. Saat mengaktifkan TTL di DynamoDB, Anda dapat menentukan nama atribut yang nilainya menunjukkan waktu kedaluwarsa item, dalam format waktu Unix dalam hitungan detik. Setelah itu, DynamoDB dapat menghapus item tanpa menimbulkan biaya tulis. Dengan tabel global, Anda dapat mengonfigurasi TTL di Wilayah mana pun, dan pengaturannya otomatis direplikasi ke Wilayah lain yang terkait dengan tabel global. Ketika suatu item memenuhi syarat untuk penghapusan melalui aturan TTL, pekerjaan tersebut dapat dilakukan di Wilayah mana pun. Operasi penghapusan dilakukan tanpa menggunakan unit tulis pada tabel sumber, tetapi tabel replika akan mendapatkan penulisan yang direplikasi untuk operasi penghapusan tersebut dan akan dikenakan biaya unit tulis yang direplikasi.
- Jika Anda menggunakan penskalaan otomatis, pastikan pengaturan kapasitas tulis maksimum yang disediakan cukup tinggi untuk menangani semua operasi tulis serta semua potensi operasi

penghapusan TTL. Penskalaan otomatis menyesuaikan setiap Wilayah sesuai dengan konsumsi tulisnya. Tabel sesuai permintaan tidak memiliki pengaturan kapasitas tulis maksimum yang disediakan, tetapi batas throughput tulis maksimum tingkat tabel menentukan kapasitas tulis berkelanjutan maksimum yang diperbolehkan oleh tabel sesuai permintaan. Batas defaultnya adalah 40.000, tetapi dapat disesuaikan. Sebaiknya Anda mengaturnya cukup tinggi untuk menangani semua operasi tulis (termasuk operasi tulis TTL) yang mungkin diperlukan tabel sesuai permintaan. Nilai ini harus sama di seluruh Wilayah yang berpartisipasi saat Anda menyiapkan tabel global.

Beberapa pertimbangan dalam mengelola kapasitas baca:

- Pengaturan manajemen kapasitas baca boleh berbeda di antara Wilayah karena diasumsikan bahwa Wilayah yang berbeda mungkin memiliki pola baca yang independen. Saat Anda pertama kali menambahkan replika global ke tabel, kapasitas Wilayah sumber disebar. Setelah pembuatan, Anda dapat menyesuaikan pengaturan kapasitas baca, yang tidak ditransfer ke sisi lain.
- Saat Anda menggunakan penskalaan otomatis DynamoDB, pastikan pengaturan kapasitas baca maksimum yang disediakan cukup tinggi untuk menangani semua operasi baca di seluruh Wilayah. Selama operasi standar, kapasitas baca mungkin akan disebar di seluruh Wilayah, tetapi tabel selama failover harus mampu beradaptasi secara otomatis dengan peningkatan beban kerja baca. Tabel sesuai permintaan tidak memiliki pengaturan kapasitas baca maksimum yang disediakan, tetapi batas throughput baca maksimum tingkat tabel menentukan kapasitas baca berkelanjutan maksimum yang diperbolehkan oleh tabel sesuai permintaan. Batas defaultnya adalah 40.000, tetapi dapat disesuaikan. Sebaiknya Anda mengaturnya cukup tinggi untuk menangani semua operasi baca yang mungkin diperlukan tabel jika semua operasi baca dirutekan ke Wilayah tunggal ini.
- Jika tabel di satu Wilayah biasanya tidak menerima lalu lintas baca tetapi mungkin harus menyerap sejumlah besar lalu lintas baca setelah failover, Anda dapat meningkatkan kapasitas baca tabel yang disediakan, menunggu tabel selesai diperbarui, lalu menyediakan tabelnya lagi. Anda dapat membiarkan tabel dalam mode yang disediakan atau mengalihkannya ke mode sesuai permintaan. Ini akan menghangatkan tabel untuk menerima tingkat lalu lintas baca yang lebih tinggi.

Route 53 ARC memiliki [pemeriksaan kesiapan](#) yang dapat berguna untuk mengonfirmasi bahwa wilayah DynamoDB memiliki pengaturan tabel dan kuota akun yang serupa, baik Anda menggunakan Route 53 untuk merutekan permintaan atau tidak. Pemeriksaan kesiapan ini juga dapat membantu menyesuaikan kuota tingkat akun untuk memastikannya cocok.

Daftar periksa persiapan untuk tabel global dan Pertanyaan yang Sering Diajukan

Gunakan daftar periksa berikut untuk keputusan dan tugas saat Anda men-deploy tabel global.

- Tentukan jumlah dan Wilayah yang akan berpartisipasi dalam tabel global.
- Tentukan mode tulis aplikasi Anda. Untuk informasi selengkapnya, lihat [Mode tulis dengan tabel global](#).
- Rencanakan strategi [Meminta perutean dengan tabel global](#), berdasarkan mode tulis Anda.
- tentukan

Evakuasi suatu Wilayah adalah proses memigrasikan aktivitas baca dan tulis keluar dari Wilayah tersebut. Ini paling sering merupakan aktivitas tulis, dan terkadang aktivitas baca.

Mengevakuasi Wilayah aktif

Anda dapat memutuskan untuk mengevakuasi Wilayah aktif karena sejumlah alasan.

Penghindaran dapat menjadi bagian dari aktivitas bisnis biasa seperti jika Anda menggunakan mode follow-the-sun tulis ke satu Wilayah. Evakuasi juga bisa disebabkan oleh keputusan bisnis untuk mengubah Wilayah yang sedang aktif, sebagai respons terhadap kegagalan tumpukan perangkat lunak di luar DynamoDB, atau karena Anda menghadapi masalah umum seperti latensi yang lebih tinggi dari biasanya di dalam Wilayah.

Dengan mode tulis ke Wilayah mana pun, mengevakuasi Wilayah aktif sangatlah mudah.

Anda dapat merutekan lalu lintas ke Wilayah alternatif melalui sistem perutean apa pun, dan membiarkan operasi tulis yang telah terjadi di Wilayah yang dievakuasi direplikasi seperti biasa.

Dengan mode tulis ke satu Wilayah dan tulis ke Wilayah Anda, Anda harus memastikan semua penulisan ke Wilayah aktif telah direkam sepenuhnya, diproses streaming, dan disebarkan secara global sebelum memulai penulisan di Wilayah aktif yang baru. Hal ini diperlukan untuk memastikan bahwa penulisan di masa mendatang bertentangan dengan versi data terbaru.

Katakanlah Wilayah A aktif dan Wilayah B pasif (baik untuk tabel lengkap atau untuk item yang ditempatkan di Wilayah A). Mekanisme umum untuk melakukan evakuasi adalah dengan menjeda operasi tulis ke A, menunggu cukup lama hingga operasi tersebut disebarkan sepenuhnya ke B, memperbarui tumpukan arsitektur untuk mengenali B sebagai aktif, lalu melanjutkan operasi tulis ke B. Tidak ada metrik yang menunjukkan kepastian mutlak bahwa

Wilayah A telah sepenuhnya mereplikasi datanya ke Wilayah B. Jika Wilayah A sehat, menjeda operasi tulis ke Wilayah A dan menunggu 10 kali nilai maksimum terbaru metrik `ReplicationLatency` biasanya sudah cukup untuk menentukan bahwa replikasi tersebut selesai. Jika Wilayah A tidak sehat dan menunjukkan area lain mengalami peningkatan latensi, Anda dapat memilih kelipatan waktu tunggu yang lebih besar.

Mengevakuasi Wilayah offline

Ada kasus khusus yang perlu dipertimbangkan: bagaimana jika Wilayah A offline sepenuhnya tanpa pemberitahuan? Hal ini sangat kecil kemungkinannya, tetapi tetap perlu dipertimbangkan. Jika hal ini terjadi, operasi tulis apa pun di Wilayah A yang belum disebar akan ditahan dan disebar setelah Wilayah A kembali online. Operasi tulis tidak hilang, tetapi penyebarannya tertunda tanpa batas waktu.

Cara melanjutkan peristiwa ini merupakan keputusan aplikasi. Untuk kelangsungan bisnis, operasi tulis mungkin perlu diteruskan ke Wilayah B utama yang baru. Namun, jika item di Wilayah B menerima pembaruan sementara ada penyebaran operasi tulis yang tertunda untuk item tersebut dari Wilayah A, penyebaran tersebut akan disembunyikan di model penulis terakhir menang. Pembaruan apa pun di Wilayah B mungkin menyembunyikan permintaan tulis yang masuk.

Dengan mode tulis ke Wilayah mana pun, pembacaan dan penulisan dapat dilanjutkan di Wilayah B, dengan keyakinan bahwa item di Wilayah A pada akhirnya akan disebar ke Wilayah B dan mengenali potensi item yang hilang hingga Wilayah A kembali online. Jika memungkinkan, Anda harus mempertimbangkan untuk memutar ulang lalu lintas tulis terbaru (misalnya, dengan menggunakan sumber peristiwa hulu) untuk mengisi celah dari operasi tulis yang mungkin hilang dan membiarkan resolusi konflik penulis terakhir menang menekan penyebaran akhir dari operasi tulis yang masuk.

Dengan mode tulis lainnya, Anda harus mempertimbangkan sejauh mana pekerjaan dapat dilanjutkan dengan sedikit out-of-date pandangan dunia. Beberapa operasi tulis berdurasi pendek, seperti yang dilacak oleh `ReplicationLatency`, akan hilang hingga Wilayah A kembali online. Apakah bisnis bisa maju? Dalam beberapa kasus penggunaan, bisa, tetapi pada kasus lainnya mungkin tidak bisa, tanpa mekanisme mitigasi tambahan.

Misalnya, katakanlah Anda perlu mempertahankan saldo kredit yang tersedia tanpa gangguan bahkan setelah kegagalan Wilayah. Anda dapat membagi saldo menjadi dua item berbeda, satu ditempatkan di Wilayah A dan satu lagi di Wilayah B, masing-masing dimulai dengan

setengah saldo yang tersedia. Ini akan menggunakan mode tulis ke Wilayah Anda. Pembaruan transaksional yang diproses di setiap Wilayah akan dituliskan pada salinan lokal saldo. Jika Wilayah A sepenuhnya offline, pekerjaan masih dapat dilanjutkan dengan pemrosesan transaksi di Wilayah B, dan operasi tulis akan terbatas pada bagian saldo yang disimpan di Wilayah B. Memisahkan saldo seperti ini menimbulkan kerumitan ketika saldo hampir habis atau kredit harus diseimbangkan kembali, tetapi hal ini memberikan satu contoh pemulihan bisnis yang aman bahkan dengan operasi tulis tertunda yang tidak pasti.

Contoh lain, misalkan Anda mengambil data formulir web. Anda dapat menggunakan [Optimistic Concurrency Control \(OCC\)](#) untuk menetapkan versi ke item data dan menyimpan versi terbaru ke formulir web sebagai bidang tersembunyi. Pada setiap pengiriman, operasi tulis berhasil hanya jika versi dalam basis data masih cocok dengan versi yang digunakan untuk membuat formulir. Jika versinya tidak cocok, formulir web bisa disegarkan (atau digabungkan secara hati-hati) berdasarkan versi saat ini dalam basis data, dan pengguna bisa melanjutkan lagi. Model OCC biasanya melindungi terhadap penimpaan klien lain dan pembuatan versi data yang baru, tetapi model ini juga dapat membantu selama failover ketika klien mungkin menemukan versi data yang lebih lama.

Misalkan Anda menggunakan timestamp sebagai versinya. Katakanlah formulir pertama kali dibuat pada Wilayah A pada pukul 12.00 tetapi (setelah failover) mencoba menulis ke Wilayah B dan mengetahui bahwa versi terbaru dalam basis data adalah 11.59. Dalam skenario ini, klien dapat menunggu versi 12.00 untuk disebarkan ke Wilayah B lalu menulis di atas versi tersebut, atau membangun pada 11.59 dan membuat versi 12.01 baru (yang, setelah ditulis, akan menyembunyikan versi yang masuk setelah Wilayah A pulih).

Contoh terakhir, perusahaan jasa keuangan menyimpan data tentang akun pelanggan dan transaksi keuangan mereka dalam basis data DynamoDB. Jika terjadi pemadaman total Wilayah A, mereka ingin memastikan bahwa aktivitas tulis apa pun yang terkait dengan akun mereka tersedia sepenuhnya di Wilayah B, atau ingin menggarantini akun mereka sebagai diketahui sebagian hingga Wilayah A kembali online. Alih-alih menghentikan semua bisnis, mereka memutuskan untuk menghentikan bisnis untuk sementara waktu, hanya pada sebagian kecil akun yang mereka anggap memiliki transaksi yang tidak disebarkan. Untuk mencapai hal ini, mereka menggunakan Wilayah ketiga, yang akan kita sebut Wilayah C. Sebelum memproses operasi tulis apa pun di Wilayah A, mereka menempatkan ringkasan singkat operasi yang tertunda tersebut (misalnya, jumlah transaksi baru untuk sebuah akun) di Wilayah C. Ringkasan ini cukup bagi Wilayah B untuk menentukan apakah pandangannya benar-benar mutakhir. Tindakan ini mengunci akun secara efektif sejak penulisan di Wilayah C hingga Wilayah A

menerima operasi tulis dan Wilayah B menerimanya. Data di Wilayah C tidak digunakan kecuali sebagai bagian dari proses failover, setelah itu Wilayah B dapat memeriksa datanya dengan Wilayah C untuk mengetahui apakah ada akun yang kedaluwarsa. Akun tersebut akan ditandai sebagai dikarantina hingga pemulihan Wilayah A menyebarkan sebagian data ke Wilayah B.

Jika Wilayah C gagal, Wilayah D baru dapat dibuka untuk digunakan. Data di Wilayah C sangat sementara, dan setelah beberapa menit Wilayah D akan memiliki up-to-date catatan yang cukup tentang operasi penulisan dalam penerbangan untuk sepenuhnya berguna. Jika Wilayah B gagal, Wilayah A dapat terus menerima permintaan tulis yang bekerja sama dengan Wilayah C. Perusahaan ini bersedia menerima penulisan dengan latensi yang lebih tinggi (ke dua Wilayah: C dan kemudian A) dan beruntung memiliki model data yang status akunya dapat dirangkum secara ringkas.

rencana evakuasi, berdasarkan mode tulis dan strategi perutean Anda.

- Tangkap metrik tentang kesehatan, latensi, dan kesalahan di setiap Wilayah. Untuk daftar metrik DynamoDB, lihat AWS posting blog Memantau [Amazon DynamoDB untuk Kesadaran Operasional untuk](#) daftar metrik yang harus diamati. Anda juga harus menggunakan [synthetic canary](#) (permintaan buatan yang dirancang untuk mendeteksi kegagalan, dinamai menurut nama kenari di tambang batu bara), serta pengamatan langsung terhadap lalu lintas pelanggan. Tidak semua masalah akan muncul di metrik DynamoDB.
- Atur alarm untuk peningkatan `ReplicationLatency` yang berkelanjutan. Peningkatan mungkin menunjukkan kesalahan konfigurasi yang tidak disengaja, yaitu tabel global memiliki pengaturan tulis berbeda di Wilayah berbeda, yang menyebabkan kegagalan permintaan yang direplikasi dan peningkatan latensi. Hal ini juga dapat mengindikasikan adanya gangguan regional. [Contoh yang baik](#) adalah menghasilkan peringatan jika rata-rata terkini melebihi 180.000 milidetik. Anda mungkin juga memperhatikan `ReplicationLatency` turun ke 0, yang menunjukkan replikasi terhenti.
- Tetapkan pengaturan baca dan tulis maksimum yang memadai untuk setiap tabel global.
- Identifikasi terlebih dahulu alasan untuk mengevakuasi suatu Wilayah. Jika keputusan melibatkan penilaian manusia, dokumentasikan semua pertimbangannya. Pekerjaan ini harus dilakukan dengan hati-hati sebelumnya, bukan di bawah tekanan.
- Simpan runbook untuk setiap tindakan yang harus dilakukan saat Anda mengevakuasi suatu Wilayah. Biasanya pekerjaan yang dilakukan untuk tabel global sangat sedikit, tetapi memindahkan sisa tumpukan dapat menjadi pekerjaan yang rumit.

Note

Praktik terbaiknya adalah hanya mengandalkan operasi bidang data, bukan operasi bidang kontrol karena beberapa operasi bidang kontrol mungkin terdegradasi selama kegagalan wilayah.

Untuk informasi selengkapnya, lihat posting AWS blog [Membangun aplikasi tangguh dengan tabel global Amazon DynamoDB](#): Bagian 4.

- Uji semua aspek runbook secara berkala, termasuk evakuasi Wilayah. Runbook yang belum teruji adalah runbook yang tidak dapat diandalkan.
- Pertimbangkan untuk menggunakan Resilience Hub untuk mengevaluasi ketahanan seluruh aplikasi Anda (termasuk tabel global). Resilience Hub memberikan pandangan komprehensif tentang status ketahanan portofolio aplikasi Anda secara keseluruhan melalui dasbornya.
- Sebaiknya gunakan pemeriksaan kesiapan Route 53 ARC untuk mengevaluasi konfigurasi aplikasi Anda saat ini dan melacak penyimpangan dari praktik terbaik.
- Saat menulis pemeriksaan kondisi untuk digunakan dengan Route 53 atau Global Accelerator, tidak cukup hanya dengan melakukan ping untuk mengetahui bahwa titik akhir DynamoDB sudah aktif. Ping tidak mencakup banyak mode kegagalan seperti kesalahan konfigurasi IAM, masalah deployment kode, kegagalan dalam tumpukan di luar DynamoDB, latensi baca atau tulis yang lebih tinggi dari rata-rata, dan sebagainya. Praktik terbaiknya adalah melakukan serangkaian panggilan yang menggunakan aliran basis data penuh.

Pertanyaan yang Sering Diajukan (FAQ) untuk men-deploy tabel global

Apa saja prinsip yang berguna untuk penggunaan keseluruhan tabel global DynamoDB?

Tabel global DynamoDB memiliki sedikit tombol kontrol, tetapi masih memerlukan sejumlah pertimbangan. Anda harus menentukan mode tulis, model perutean, dan proses evakuasi. Anda harus melengkapi aplikasi Anda di setiap Wilayah dan siap menyesuaikan rute atau melakukan evakuasi untuk menjaga kesehatan global. Sehingga Anda akan mendapatkan kumpulan data yang terdistribusi secara global dengan pembacaan dan penulisan latensi rendah serta perjanjian tingkat layanan 99,999%.

Berapa harga untuk tabel global?

Penulisan ke tabel DynamoDB tradisional diberi harga dalam Unit Kapasitas Tulis (WCU, untuk tabel yang disediakan) atau Unit Permintaan Tulis (WRU, untuk tabel sesuai permintaan). Jika menulis item 5 KB, Anda dikenakan biaya 5 unit. Penulisan ke tabel global diberi harga dalam Unit Kapasitas Tulis yang Direplikasi (rWCU, untuk tabel yang disediakan) atau Unit Permintaan Tulis yang Direplikasi (rWRU, untuk tabel sesuai permintaan).

rWCU dan rWRU mencakup biaya infrastruktur streaming yang diperlukan untuk mengelola replikasi. Dengan demikian, harganya 50% lebih tinggi dari WCU dan WRU. Biaya transfer data lintas Wilayah berlaku.

Biaya Unit Tulis yang Direplikasi dikenakan di setiap Wilayah tempat item tersebut ditulis langsung atau ditulis dengan replikasi.

Menulis ke Indeks Sekunder Global (GSI) dianggap sebagai penulisan lokal dan menggunakan Unit Tulis biasa.

Saat ini, tidak ada Kapasitas Terpesan yang tersedia untuk rWCU saat ini. Membeli Kapasitas Terpesan mungkin masih bermanfaat untuk tabel dengan GSI yang menggunakan unit tulis.

Bootstrap awal saat menambahkan Wilayah baru ke tabel global dikenakan biaya seperti pemulihan per GB data yang dipulihkan, ditambah biaya transfer data lintas Wilayah.

Wilayah mana yang didukung tabel global?

[Versi Tabel Global 2019.11.21 \(Saat Ini\)](#) tersedia di sebagian besar Wilayah. Anda dapat melihat daftar terbaru di daftar dropdown Wilayah di konsol DynamoDB saat menambahkan replika.

Bagaimana GSI ditangani dengan tabel global?

Di [Tabel Global versi 2019.11.21 \(Saat Ini\)](#), [saat](#) Anda membuat GSI di satu Wilayah, GSI secara otomatis dibuat di Wilayah lain yang berpartisipasi dan diisi ulang secara otomatis.

Bagaimana cara menghentikan replikasi tabel global?

Anda dapat menghapus tabel replika seperti Anda menghapus tabel lainnya. Menghapus tabel global akan menghentikan replikasi ke Wilayah tersebut dan menghapus salinan tabel yang disimpan di Wilayah tersebut. Namun, Anda tidak dapat menghentikan replikasi sambil menyimpan salinan tabel sebagai entitas independen, dan Anda juga tidak dapat menjeda replikasi.

Bagaimana DynamoDB Streams berinteraksi dengan tabel global?

Setiap tabel global menghasilkan aliran independen berdasarkan semua penulisannya, dari mana pun tabel tersebut dimulai. Anda dapat menggunakan aliran DynamoDB di satu Wilayah atau di

semua Wilayah (secara independen). Jika ingin memproses operasi tulis lokal tetapi tidak direplikasi, Anda dapat menambahkan atribut Wilayah Anda sendiri ke setiap item untuk mengidentifikasi Wilayah penulisan. Anda kemudian dapat menggunakan filter peristiwa Lambda untuk memanggil fungsi Lambda hanya untuk operasi tulis di Wilayah lokal. Ini akan membantu dalam operasi penyisipan dan pembaruan, tetapi sayangnya tidak membantu operasi penghapusan.

Bagaimana tabel global menangani transaksi?

Operasi transaksional memberikan jaminan atomicity, consistency, isolation, dan durability (ACID) hanya di Wilayah tempat operasi tulis pertama kali terjadi. Transaksi tidak didukung di seluruh Wilayah dalam tabel global. Misalnya, jika Anda memiliki tabel global dengan replika di Wilayah AS Timur (Ohio) dan AS Barat (Oregon) serta melakukan operasi `TransactWriteItems` di Wilayah AS Timur (Ohio), Anda mungkin melihat transaksi yang selesai sebagian di Wilayah AS Barat (Oregon) seiring perubahan direplikasi. Perubahan direplikasi ke Wilayah lain hanya setelah perubahan tersebut dibuat di Wilayah sumber.

Bagaimana tabel global berinteraksi dengan cache DynamoDB Accelerator (DAX)?

Tabel global melewati DAX dengan memperbarui DynamoDB secara langsung, sehingga DAX tidak mengetahui jika menyimpan data yang sudah usang. Cache DAX disegarkan hanya ketika TTL cache kedaluwarsa.

Apakah tag pada tabel disebarakan?

Tidak, tag tidak disebarakan secara otomatis.

Apakah saya harus mencadangkan tabel di semua Wilayah atau cukup satu Wilayah?

Jawabannya tergantung pada tujuan pencadangan. Jika Anda ingin memastikan ketahanan data, DynamoDB sudah menyediakan perlindungan itu. Layanan ini memastikan ketahanan. Jika Anda ingin menyimpan snapshot untuk catatan historis (misalnya, untuk memenuhi persyaratan peraturan), membuat cadangan di satu Wilayah sudah cukup. Anda dapat menyalin cadangan ke Wilayah tambahan dengan menggunakan AWS Backup. Jika Anda ingin memulihkan data yang terhapus atau diubah secara keliru, gunakan [pemulihan titik waktu \(PITR\) DynamoDB](#) dalam satu Wilayah.

Bagaimana cara menerapkan tabel global menggunakan? AWS CloudFormation

CloudFormation merupakan tabel DynamoDB dan tabel global sebagai dua sumber daya terpisah: `AWS::DynamoDB::Table` dan `AWS::DynamoDB::GlobalTable`. Salah satu pendekatannya adalah membuat semua tabel yang berpotensi bersifat global dengan menggunakan konsep

GlobalTable. Anda kemudian dapat menyimpannya sebagai tabel mandiri pada awalnya, dan menambahkan Wilayah nanti jika diperlukan.

Dalam CloudFormation, setiap tabel global dikendalikan oleh satu tumpukan, dalam satu Wilayah, terlepas dari jumlah replika. Saat Anda menerapkan template Anda, CloudFormation membuat dan memperbarui semua replika sebagai bagian dari operasi tumpukan tunggal. Jangan men-deploy sumber daya [AWS::DynamoDB::GlobalTable](#) yang sama di beberapa Wilayah.

Tindakan tersebut tidak didukung dan akan mengakibatkan kesalahan. Jika men-deploy templat aplikasi di beberapa Wilayah, Anda dapat menggunakan ketentuan untuk membuat sumber daya `AWS::DynamoDB::GlobalTable` di satu Wilayah. Alternatifnya, Anda dapat memilih untuk menentukan sumber daya `AWS::DynamoDB::GlobalTable` Anda dalam tumpukan yang terpisah dari tumpukan aplikasi Anda, dan memastikan bahwa sumber daya tersebut disebar ke satu Wilayah.

Jika Anda memiliki tabel reguler dan Anda ingin mengonversinya menjadi tabel global sambil menjaganya agar tetap dikelola CloudFormation kemudian mengatur kebijakan penghapusan ke Retain, hapus tabel dari tumpukan, ubah tabel menjadi tabel global di konsol, lalu impor tabel global sebagai sumber daya baru ke tumpukan.

Replikasi lintas akun tidak didukung saat ini.

Praktik terbaik untuk mengelola bidang kontrol di DynamoDB

Note

DynamoDB memperkenalkan batas throttling bidang kontrol 2.500 permintaan per detik dengan opsi percobaan ulang. Lihat di bawah ini untuk detail tambahan.

Operasi bidang kontrol DynamoDB memungkinkan Anda mengelola tabel DynamoDB serta objek yang bergantung pada tabel seperti indeks. Untuk informasi selengkapnya tentang operasi ini, lihat [Bidang kontrol](#).

Dalam beberapa situasi, Anda mungkin perlu mengambil tindakan dan menggunakan data yang dikembalikan oleh panggilan bidang kontrol sebagai bagian dari logika bisnis Anda. Misalnya, Anda mungkin perlu mengetahui nilai `ProvisionedThroughput` yang dikembalikan oleh `DescribeTable`. Dalam situasi ini, ikuti praktik terbaik berikut:

- Jangan mengkueri bidang kontrol DynamoDB secara berlebihan.

- Jangan menggabungkan panggilan bidang kontrol dan panggilan bidang data dalam kode yang sama.
- Tangani throttling pada permintaan bidang kontrol dan coba lagi dengan backoff.
- Invokasi dan lacak perubahan pada sumber daya tertentu dari satu klien.
- Daripada mengambil data untuk tabel yang sama beberapa kali dalam interval pendek, simpan data dalam cache untuk diproses.

Praktik Terbaik untuk Memahami Laporan AWS Penagihan dan Penggunaan Anda

Dokumen ini menjelaskan kode UsageType penagihan untuk biaya yang terkait dengan DynamoDB.

AWS menyediakan laporan biaya dan penggunaan (CUR) yang berisi data untuk layanan yang digunakan. Anda dapat menggunakan AWS Cost and Usage Report untuk mempublikasikan laporan penagihan ke Amazon S3 dalam format CSV. Saat menyiapkan CUR, Anda dapat memilih untuk memecah periode waktu berdasarkan jam, hari, atau bulan, dan Anda dapat memilih apakah Anda ingin memecah penggunaan berdasarkan ID sumber daya atau tidak. Untuk detail lebih lanjut tentang menghasilkan CUR, silakan lihat [Membuat Laporan Biaya dan Penggunaan](#)

Dalam ekspor CSV, Anda akan menemukan atribut yang relevan terdaftar untuk setiap baris. Berikut ini adalah contoh atribut yang dapat disertakan:

- lineitem/UsageStartDate: Tanggal dan waktu mulai untuk item baris di UTC, inklusif.
- lineitem/UsageEndDate: Tanggal dan waktu akhir untuk item baris yang sesuai di UTC, eksklusif.
- lineitem/: ProductCode Untuk DynamoDB ini akan menjadi "DB" AmazonDynamo
- lineitem/UsageType: Kode deskripsi khusus untuk jenis penggunaan, sebagaimana disebutkan dalam dokumen ini
- LineItem/Operation: Nama yang memberikan konteks muatan seperti nama operasi yang dikenakan biaya (opsional).
- lineitem/ResourceId: Pengidentifikasi untuk sumber daya yang dikeluarkan penggunaan. Tersedia jika CUR menyertakan rincian berdasarkan ID sumber daya.
- lineitem/UsageAmount: Jumlah penggunaan yang terjadi selama periode waktu yang ditentukan.
- lineitem/UnblendedCost: Biaya penggunaan ini.
- lineitem/LineItemDescription: Deskripsi tekstual dari item baris.

Untuk informasi selengkapnya tentang kamus data CUR, lihat [Laporan Biaya dan Penggunaan \(CUR\) 2.0](#). Perhatikan bahwa nama yang tepat bervariasi tergantung pada konteksnya.

A UsageType adalah string dengan nilai seperti `ReadCapacityUnit-Hrs`, `USW2-ReadRequestUnitsEU-WriteCapacityUnit-Hrs`, atau `USE1-TimedPITRStorage-ByteHrs`. Setiap jenis penggunaan dimulai dengan awalan Region opsional. Jika tidak ada, itu menunjukkan Wilayah `us-timur-1`. Jika ada, tabel di bawah ini memetakan kode Region penagihan pendek ke kode Region konvensional dan nama.

Misalnya, penggunaan bernama `USW2-ReadRequestUnits` menunjukkan unit permintaan baca yang dikonsumsi di `us-west-2`.

Kode Wilayah Penagihan	Kode Wilayah	Nama Wilayah
AFS1	af-south-1	Afrika (Cape Town)
KERA1	ap-east-1	Asia Pasifik (Hong Kong)
APN1	ap-northeast-1	Asia Pasifik (Tokyo)
APN2	ap-northeast-2	Asia Pasifik (Seoul)
APN3	ap-northeast-3	Asia Pasifik (Osaka)
APS1	ap-south-1	Asia Pasifik (Mumbai)
APS2	ap-south-2	Asia Pasifik (Hyderabad)
APS3	ap-southeast-1	Asia Pasifik (Singapura)
APS4	ap-southeast-2	Asia Pasifik (Sydney)
APS5	ap-southeast-3	Asia Pasifik (Jakarta)
APS6	ap-southeast-4	Asia Pasifik (Melbourne)
BISA1	ca-central-1	Kanada (Pusat)
EU	eu-central-1	Eropa (Frankfurt)
EUC1	eu-central-2	Eropa (Zürich)

Kode Wilayah Penagihan	Kode Wilayah	Nama Wilayah
EUN1	eu-north-1	Eropa (Stockholm)
EUS1	eu-south-1	Eropa (Milan)
EUS2	eu-south-2	Eropa (Spanyol)
EUW1	eu-west-1	Eropa (Irlandia)
EUW2	eu-west-2	Eropa (London)
EUW3	eu-west-3	Eropa (Paris)
ILC1	il-sentral-1	Israel (Tel Aviv)
MEC1	me-central-1	Timur Tengah (UEA)
MES1	me-south-1	Timur Tengah (Bahrain)
SAE1	sa-east-1	Amerika Selatan (Sao Paulo)
USE1 (default)	us-east-1	AS Timur (Virginia Utara)
MENGGUNAKAN2	us-east-2	AS Timur (Ohio)
UGE1	us-gov-east-1	Pemerintah AS Timur
UGW1	us-gov-west-1	Pemerintah AS Barat
USW1	us-west-1	AS Barat (California Utara)
USW2	us-west-2	AS Barat (Oregon)

Pada bagian berikut, kita menggunakan REG-UsageType pola ketika akan melalui biaya untuk DynamoDB, di mana REG menentukan wilayah di mana penggunaan terjadi dan UsageType adalah kode untuk jenis biaya. Misalnya jika Anda melihat item baris USW1- ReadCapacityUnit-Hrs di file CSV Anda, itu berarti penggunaan dilakukan di AS-West-1 untuk kapasitas baca yang disediakan. Dalam hal ini daftar akan mengatakan REG-ReadCapacityUnit-Hrs.

Topik

- [Kapasitas Throughput](#)
- [Pengaliran](#)
- [Penyimpanan](#)
- [Pencadangan dan Pemulihan](#)
- [Transfer Data](#)
- [CloudWatch Wawasan Kontributor](#)
- [DynamoDB Accelerator \(DAX\)](#)

Kapasitas Throughput

Kapasitas yang Disediakan Membaca dan Menulis

Saat Anda membuat tabel DynamoDB dalam mode kapasitas yang disediakan, Anda menentukan kapasitas baca dan tulis yang Anda harapkan dibutuhkan aplikasi Anda. Jenis penggunaan tergantung pada kelas tabel Anda (Standard atau Standard-Infrequent Access). Anda menyediakan membaca dan menulis berdasarkan tingkat konsumsi per detik, tetapi biaya tersebut diberi harga per jam berdasarkan kapasitas yang disediakan.

UsageType	Unit	Granularitas	Deskripsi
REG- -Jam ReadCapacityUnit	RCU-jam	Jam	Biaya untuk pembacaan dalam mode kapasitas yang disediakan menggunakan kelas tabel Standar.
REG-IA- -Jam ReadCapacityUnit	RCU-jam	Jam	Biaya untuk pembacaan dalam mode kapasitas yang disediakan menggunakan kelas tabel IA standar.
REG- -Jam WriteCapacityUnit	Jam WCU	Jam	Biaya untuk menulis dalam

UsageType	Unit	Granularitas	Deskripsi
			mode kapasitas yang disediakan menggunakan kelas tabel Standar.
REG-IA- -Jam WriteCapacityUnit	Jam WCU	Jam	Biaya untuk menulis dalam mode kapasitas yang disediakan menggunakan kelas tabel IA standar.

Kapasitas Cadangan Membaca dan Menulis

Dengan kapasitas yang dicadangkan, Anda membayar biaya satu kali di muka dan berkomitmen pada tingkat penggunaan minimum yang disediakan selama jangka waktu tertentu. Kapasitas cadangan ditagih dengan tarif per jam diskon. Kapasitas apa pun yang Anda sediakan melebihi kapasitas cadangan Anda akan ditagih sesuai tarif kapasitas standar yang disediakan. Kapasitas cadangan tersedia untuk unit kapasitas baca dan tulis satu wilayah, yang disediakan (RCU dan WCU) pada tabel DynamoDB yang menggunakan kelas tabel standar. Kapasitas cadangan 1 tahun dan 3 tahun ditagih menggunakan SKU yang sama.

UsageType	Unit	Granularitas	Deskripsi
REG-:dynamodb.read HeavyUsage	RCU-jam	Di muka lalu bulanan	Biaya untuk kapasitas cadangan berbunyi: biaya di muka satu kali dan biaya bulanan pada awal setiap bulan yang mencakup semua jam RCU yang didiskon selama sebulan. Akan memiliki item baris REG- ReadCapac

UsageType	Unit	Granularitas	Deskripsi
			ityUnit -Hrs tanpa biaya yang cocok.
REG-:dynamodb.write HeavyUsage	WCU-jam	Di muka lalu bulanan	Biaya untuk kapasitas cadangan menulis: biaya di muka satu kali dan biaya bulanan pada awal setiap bulan yang mencakup semua jam WCU yang didiskon selama sebulan. Akan memiliki item baris REG- WriteCapacityUnit -Hrs tanpa biaya yang cocok.

Kapasitas Sesuai Permintaan Membaca dan Menulis

Saat Anda membuat tabel DynamoDB dalam mode kapasitas sesuai permintaan, Anda hanya membayar untuk membaca dan menulis yang dilakukan aplikasi Anda. Harga untuk permintaan baca dan tulis tergantung pada kelas tabel Anda.

UsageType	Unit	Granularitas	Deskripsi
REG- ReadRequestUnits	RRU	Unit	Biaya untuk membaca dalam mode kapasitas sesuai permintaan dengan kelas tabel Standar.
REG-IA- ReadRequestUnits	RRU	Unit	Biaya untuk membaca dalam mode kapasitas sesuai

UsageType	Unit	Granularitas	Deskripsi
			permintaan dengan kelas tabel IA standar.
REG- WriteRequ estUnits	WRU	Unit	Biaya untuk menulis dalam mode kapasitas sesuai permintaan dengan kelas tabel Standar.
REG-IA- WriteRequ estUnits	WRU	Unit	Biaya untuk menulis dalam mode kapasitas sesuai permintaan dengan kelas tabel IA standar.

Tabel Global Membaca dan Menulis

DynamoDB mengenakan biaya untuk penggunaan tabel global berdasarkan sumber daya yang digunakan pada setiap tabel replika. Untuk tabel global yang disediakan, permintaan tulis untuk tabel global diukur dalam WCU yang direplikasi (RWCU) alih-alih WCU standar dan penulisan ke indeks sekunder global dalam tabel global diukur dalam WCU. Untuk tabel global sesuai permintaan, permintaan tulis diukur dalam WRU yang direplikasi (rWRU) alih-alih WRU standar. Jumlah RWCU atau RWRU yang dikonsumsi untuk replikasi tergantung pada versi tabel global yang Anda gunakan. Harga tergantung pada kelas meja Anda.

Menulis ke indeks sekunder global (GSI) ditagih menggunakan unit tulis standar (WCU dan WRU). Permintaan baca dan penyimpanan data ditagih identik dengan tabel satu wilayah.

Jika Anda menambahkan replika tabel untuk membuat atau memperluas tabel global di Wilayah baru, DynamoDB mengenakan biaya untuk pemulihan tabel di Wilayah tambahan per gigabyte data yang dipulihkan. Data yang Dipulihkan dibebankan sebagai REG- RestoreDataSize -Bytes. Silakan merujuk ke [Menggunakan cadangan Sesuai Permintaan dan DynamoDB Permintaan](#) untuk detailnya. Replikasi Lintas Wilayah dan menambahkan replika ke tabel yang berisi data juga dikenakan biaya untuk transfer data keluar.

Saat memilih mode kapasitas sesuai permintaan untuk tabel global DynamoDB, Anda hanya membayar sumber daya yang digunakan aplikasi pada setiap tabel replika.

UsageType	Unit	Granularitas	Deskripsi
REG- -Jam ReplWrite CapacityUnit	RWCU-jam	Jam	Tabel global, disediakan, Kelas tabel standar.
REG-IA- -Jam ReplWriteCapacityUnit	RWCU-jam	Jam	Tabel global, disediakan, kelas tabel IA standar.
REG- ReplWrite RequestUnits	RwRU	Unit	Tabel global, sesuai permintaan, Kelas tabel standar.
REG-IA- ReplWrite RequestUnits	RwRU	Unit	Tabel global, sesuai permintaan, kelas tabel Standar-IA

Pengaliran

DynamoDB memiliki dua teknologi streaming, DynamoDB Streams dan Kinesis. Masing-masing memiliki harga terpisah.

DynamoDB Streams mengenakan biaya untuk membaca data dalam unit permintaan baca. Setiap panggilan GetRecords API ditagih sebagai permintaan baca aliran. Anda tidak dikenakan biaya untuk panggilan GetRecords API yang dipanggil oleh AWS Lambda sebagai bagian dari pemicu DynamoDB atau oleh tabel global DynamoDB sebagai bagian dari replikasi.

UsageType	Unit	Granularitas	Deskripsi
Reg-aliran- RequestsCount	Hitungan	Unit	Baca unit permintaan untuk DynamoDB Streams.

Amazon Kinesis Data Streams mengisi daya dalam unit pengambilan data perubahan. DynamoDB membebaskan satu unit pengambilan data perubahan untuk setiap penulisan (hingga 1 KB). Untuk

item yang lebih besar dari 1 KB, diperlukan unit pengambilan data perubahan tambahan. Anda hanya membayar untuk penulisan yang dilakukan aplikasi Anda tanpa harus mengelola kapasitas throughput di atas meja.

UsageType	Unit	Granularitas	Deskripsi
REG- ChangeDataCaptureUnits - Kinesis	Unit CDC	Unit	Ubah unit pengambilan data untuk Kinesis Data Streams.

Penyimpanan

DynamoDB mengukur ukuran data yang dapat ditagih dengan menambahkan ukuran byte mentah data Anda ditambah overhead penyimpanan per item yang bergantung pada fitur yang telah Anda aktifkan.

Note

Nilai penggunaan penyimpanan di CUR akan lebih tinggi dibandingkan dengan nilai penyimpanan saat menggunakan `DescribeTable`, karena `DescribeTable` tidak termasuk overhead penyimpanan per item.

Penyimpanan dihitung setiap jam tetapi dihargai setiap bulan seperti yang dihitung dari rata-rata biaya per jam.

Meskipun penyimpanan UsageType digunakan ByteHrs sebagai akhiran, penggunaan penyimpanan di CUR diukur dalam GB dan dihargai oleh GB-bulan.

UsageType	Unit	Granularitas	Deskripsi
REG- - TimedStorage ByteHrs	GB	Bulan	Jumlah penyimpanan yang digunakan oleh tabel dan indeks DynamoDB Anda, untuk tabel dengan kelas tabel Standar.

UsageType	Unit	Granularitas	Deskripsi
REG-IA- - TimedStorage ByteHrs	GB	Bulan	Jumlah penyimpanan yang digunakan oleh tabel dan indeks DynamoDB Anda, untuk tabel dengan kelas tabel Standard-IA.

Pencadangan dan Pemulihan

DynamoDB menawarkan dua jenis backup: Point In Time Recovery (PITR) backup dan on-demand backup. Pengguna juga dapat memulihkan dari cadangan tersebut ke tabel DynamoDB. Biaya di bawah ini mengacu pada pencadangan dan pemulihan.

Biaya penyimpanan cadangan dikeluarkan pada awal bulan dengan penyesuaian yang dilakukan sepanjang bulan karena cadangan ditambahkan atau dihapus. Lihat blog [Memahami Amazon DynamoDB On-Demand Backup and Billing](#) untuk informasi selengkapnya

UsageType	Unit	Granularitas	Deskripsi
REG- - TimedBackupStorage ByteHrs	GB	Bulan	Penyimpanan yang dikonsumsi oleh cadangan sesuai permintaan dari tabel DynamoDB dan Indeks Sekunder Lokal Anda.
TimedPitrStorage-ByteHrs	GB	Bulan	Penyimpanan yang digunakan oleh backup point-in-time recovery (PITR). DynamoDB memantau ukuran tabel berkemamp

UsageType	Unit	Granularitas	Deskripsi
			uan PITR Anda secara terus menerus sepanjang bulan untuk menentukan biaya cadangan dan tagihan penyimpanan Anda selama PITR diaktifkan.
REG- RestoreDataSize -Byte	GB	Ukuran	Ukuran total data yang dipulihkan (termasuk data tabel, indeks sekunder lokal, dan indeks sekunder global) diukur dalam GB dari cadangan DynamoDB.

AWS Backup

AWS Backup adalah layanan pencadangan yang dikelola sepenuhnya yang memudahkan untuk memusatkan dan mengotomatiskan cadangan data di seluruh AWS layanan di cloud maupun di tempat. AWS Backup dikenakan biaya untuk penyimpanan (penyimpanan hangat atau dingin), aktivitas restorasi, dan transfer data lintas wilayah. UsageTypeBiaya berikut muncul di bawah “AWS Backup” dan ProductCode bukan “AmazonDynamoDB”.

UsageType	Unit	Granularitas	Deskripsi
REG- WarmStorage - ByteHrs -DynamoDB	GB	Bulan	Penyimpanan yang digunakan oleh backup DynamoDB dikelola AWS Backup oleh sepanjang bulan,

UsageType	Unit	Granularitas	Deskripsi
			diukur dalam GB-bulan.
REG- CrossRegion - WarmBytes - DynamoDB	GB	Ukuran	Data ditransfer ke AWS Wilayah yang berbeda baik dalam akun yang sama atau ke AWS akun yang berbeda. Biaya transfer Lintas Wilayah terjadi saat menyalin cadangan dari satu Wilayah ke Wilayah lain. Tagihan selalu ditagih ke akun tempat data ditransfer.
Reg-Restore- -DynamoDB WarmBytes	GB	Ukuran	Ukuran total data yang dipulihkan dari penyimpanan hangat, diukur dalam GB.
REG- ColdStorage - ByteHrs -DynamoDB	GB	Bulan	Cold storage yang digunakan oleh DynamoDB backup dikelola AWS Backup oleh sepanjang bulan, diukur dalam GB-bulan.
Reg-Restore- - DynamoDB ColdBytes	GB	Bulan	Ukuran total data yang dipulihkan dari cold storage, diukur dalam GB.

Ekspor dan Impor

Anda dapat mengekspor data dari DynamoDB ke Amazon S3 atau mengimpor data dari Amazon S3 ke tabel DynamoDB baru.

Meskipun UsageType penggunaan Bytes sebagai akhiran, penggunaan ekspor dan impor di CUR diukur dan diberi harga dalam GB.

UsageType	Unit	Granularitas	Deskripsi
REG- ExportDataSize-Byte	GB	Ukuran	Biaya untuk mengekspor data ke S3. DynamoDB mengenakan biaya untuk data yang Anda ekspor berdasarkan ukuran tabel dasar DynamoDB (data tabel dan indeks sekunder lokal) pada titik waktu yang ditentukan ketika ekspor dibuat.
REG- ImportDataSize-Byte	GB	Ukuran	Biaya untuk mengimpor data dari S3. Ukuran dihitung berdasarkan ukuran objek yang tidak terkompresi dari data dalam Amazon S3. Tidak ada biaya tambahan untuk mengimpor ke tabel dengan GSI.

UsageType	Unit	Granularitas	Deskripsi
REG- IncrementalExportDataSize - Byte	GB	Ukuran	Biaya untuk ukuran data yang diproses dari cadangan berkelanjutan untuk menghasilkan ekspor tambahan.

Transfer Data

Aktivitas transfer data mungkin muncul terkait dengan layanan DynamoDB. DynamoDB tidak mengenakan biaya untuk transfer data masuk, dan tidak mengenakan biaya untuk data yang ditransfer antara DynamoDB dan layanan AWS lain dalam Wilayah yang AWS sama (dengan kata lain, \$0,00 per GB). Data yang ditransfer di seluruh AWS Wilayah (seperti antara DynamoDB di Wilayah AS Timur [Virginia N.] dan Amazon EC2 di Wilayah Uni Eropa [Irlandia]) dikenakan biaya pada kedua sisi transfer.

UsageType	Unit	Granularitas	Deskripsi
REG- DataTransfer - Dalam Byte	GB	Unit	Data ditransfer ke DynamoDB dari internet.
REG- DataTransfer - Out-Bytes	GB	Unit	Data ditransfer keluar dari DynamoDB ke internet.

CloudWatch Wawasan Kontributor

CloudWatch Contributor Insights for DynamoDB adalah alat diagnostik untuk mengidentifikasi kunci yang paling sering diakses dan dibatasi di tabel DynamoDB Anda. UsageTypeBiaya berikut muncul di bawah “AmazonCloudWatch” dan ProductCode bukan “AmazonDynamoDB”.

UsageType	Unit	Granularitas	Deskripsi
REG-CW: ContributorEventsManaged	Acara diproses	Unit	Jumlah peristiwa DynamoDB diproses. Misalnya untuk tabel dengan CloudWatch Contributor Insights diaktifkan, kapan saja item dibaca atau ditulis, itu dihitung sebagai satu peristiwa. Jika tabel memiliki kunci pengurutan, itu menghasilkan biaya untuk dua peristiwa.
REG-CW: ContributorRulesManaged	Hitungan aturan	Bulan	DynamoDB membuat aturan untuk mengidentifikasi item yang paling banyak diakses dan kunci yang paling dibatasi saat Anda mengaktifkan Wawasan Kontributor atau Cloud Watch. Biaya ini dikenakan untuk aturan yang ditambahkan untuk setiap entitas (tabel dan GSI) yang dikonfigurasi untuk mencatat CloudWatch wawasan kontributor.

DynamoDB Accelerator (DAX)

DynamoDB Accelerator (DAX) ditagih berdasarkan jam berdasarkan jenis instance yang dipilih untuk layanan. Biaya di bawah ini mengacu pada instans DynamoDB Accelerator yang disediakan. UsageTypeBiaya berikut muncul di bawah “AmazonDAX” ProductCode daripada “AmazonDynamoDB”.

UsageType	Unit	Granularitas	Deskripsi
REG NodeUsage -:dax- <INSTANCE TYPE>	Jam nod	Jam	Penggunaan per jam dari jenis instance tertentu. Harga adalah per node-jam yang dikonsumsi, dari saat node diluncurkan hingga dihentikan. Setiap jam simpul sebagian yang dikonsumsi akan ditagih sebagai satu jam penuh. Biaya DAX untuk setiap node dalam cluster DAX. Jika Anda memiliki klaster dengan beberapa node, Anda akan melihat beberapa item baris dalam laporan penagihan Anda.

Jenis instance akan menjadi nilai seperti salah satu dari tabel berikut. Untuk detail tentang jenis node, lihat [Simpul](#).

<INSTANCETYPE>		
r3.2xlarge	r4.8xlarge	r5.8xlarge
r3.4xlarge	r4.large	r5.large
r3.8xlarge	r4.xlarge	r5.xlarge
r3.2xlarge	r5.12xlarge	t2.medium
r3.4xlarge	r4.large	r5.large
r3.xlarge	r5.16xlarge	t2.small
r4.16xlarge	r5.24xlarge	t3.medium
r4.2xlarge	r5.2xlarge	t3.small
r4.4xlarge	r5.4xlarge	

Menggunakan DynamoDB dengan layanan AWS lain

Amazon DynamoDB terintegrasi dengan layanan AWS lainnya, yang memungkinkan Anda mengotomatisasi tugas berulang atau membuat aplikasi yang mencakup beberapa layanan.

Topik

- [konfigurasiAWSkredensial pada berkas Anda menggunakan Amazon Cognito](#)
- [Memuat data dari DynamoDB ke Amazon Redshift](#)
- [Mengolah data DynamoDB dengan Apache Hive di Amazon EMR](#)
- [Integrasi dengan Amazon S3](#)
- [Integrasi DynamoDB Zero-ETL dengan Amazon Service OpenSearch](#)
- [Praktik terbaik integrasi dengan DynamoDB](#)

konfigurasiAWSkredensial pada berkas Anda menggunakan Amazon Cognito

Cara yang disarankan untuk mendapatkan kredensial AWS untuk aplikasi web dan seluler Anda adalah dengan menggunakan Amazon Cognito. Amazon Cognito membantu Anda menghindari hardcoding kredensial AWS pada berkas Anda. Ia menggunakan AWS Identity and Access Management (IAM) role untuk menghasilkan kredensial sementara bagi pengguna yang diautentikasi dan tidak diautentikasi aplikasi Anda.

Misalnya, untuk mengkonfigurasi JavaScript file untuk menggunakan peran tidak terautentikasi Amazon Cognito agar dapat mengakses layanan web Amazon DynamoDB, lakukan hal berikut ini.

Untuk mengonfigurasi kredensial untuk diintegrasikan dengan Amazon Cognito

1. Buat kumpulan identitas Amazon Cognito yang memungkinkan identitas yang tidak terautentikasi.

```
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"
  }

```

- Salin kebijakan berikut ini pada file bernama `myCognitoPolicy.json`. Ganti ID kumpulan identitas (`us-west-2:12345678-1ab2-123a-1234-a12345ab12`) dengan `IdentityPoolId` Anda sendiri yang diperoleh pada langkah sebelumnya.

```

{
  "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"
        }
      }
    }
  ]
}

```

- Buat IAM role yang mengasumsikan kebijakan sebelumnya. Dengan cara ini, Amazon Cognito menjadi entitas tepercaya yang dapat mengasumsikan peran `Cognito_DynamoPoolUnauth`.

```

aws iam create-role --role-name Cognito_DynamoPoolUnauth \
--assume-role-policy-document file://PathToFile/myCognitoPolicy.json --output json

```

- Berikan peran `Cognito_DynamoPoolUnauth` akses penuh ke DynamoDB dengan melampirkan kebijakan terkelola (`AmazonDynamoDBFullAccess`).

```

aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess \
--role-name Cognito_DynamoPoolUnauth

```

Note

Atau, Anda dapat memberikan akses detail ke DynamoDB. Untuk informasi selengkapnya, lihat [Menggunakan ketentuan kebijakan IAM ketentuan untuk kontrol akses yang sangat baik](#).

5. Mendapatkan dan menyalin IAM role Amazon Resource Name (ARN).

```
aws iam get-role --role-name Cognito_DynamoPoolUnauth --output json
```

6. Tambahkan peran Cognito_DynamoPoolUnauth ke kumpulan identitas DynamoPool. Format untuk menentukannya adalah KeyName=string, di mana KeyName adalah unauthenticated dan string adalah peran ARN yang diperoleh pada langkah sebelumnya.

```
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. Tentukan kredensial Amazon Cognito pada berkas Anda. Memodifikasi IdentityPoolId dan RoleArn dengan tepat.

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({  
IdentityPoolId: "us-west-2:12345678-1ab2-123a-1234-a12345ab12",  
RoleArn: "arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth"  
});
```

Sekarang Anda dapat menjalankan JavaScript program terhadap layanan web DynamoDB menggunakan kredensial Amazon Cognito. Untuk informasi selengkapnya, lihat [Pengaturan kredensial di peramban web](#) di dalam AWS SDK for JavaScript Panduan Memulai.

Memuat data dari DynamoDB ke Amazon Redshift

Amazon Redshift melengkapi Amazon DynamoDB dengan kemampuan kecerdasan bisnis canggih dan antarmuka berbasis SQL yang kuat. Ketika Anda menyalin data dari tabel DynamoDB ke Amazon Redshift, Anda dapat melakukan kueri analisis data yang kompleks pada data tersebut, termasuk bergabung dengan tabel lain di kluster Amazon Redshift Anda.

Dalam hal throughput yang ditetapkan, salinan operasi dari tabel DynamoDB dihitung berdasarkan kapasitas baca tabel tersebut. Setelah data disalin, kueri SQL Anda di Amazon Redshift tidak memengaruhi DynamoDB dengan cara apa pun. Hal ini karena pertanyaan Anda bertindak berdasarkan salinan data dari DynamoDB, bukan pada DynamoDB itu sendiri.

Sebelum Anda dapat memuat data dari tabel DynamoDB, Anda harus terlebih dahulu membuat tabel Amazon Redshift untuk berfungsi sebagai tujuan untuk data. Perlu diingat bahwa Anda menyalin data dari lingkungan NoSQL ke lingkungan SQL, dan bahwa ada aturan tertentu dalam satu lingkungan yang tidak berlaku di lingkungan lain. Berikut beberapa perbedaan yang perlu dipertimbangkan:

- Nama tabel DynamoDB dapat berisi hingga 255 karakter, termasuk karakter '.' (titik) dan '-' (tanda hubung), dan peka terhadap huruf besar-kecil. Nama tabel Amazon Redshift dibatasi 127 karakter, tidak dapat berisi titik atau tanda hubung dan tidak peka terhadap huruf besar-kecil. Selain itu, nama tabel tidak dapat konflik dengan kata-kata berhak cipta Amazon Redshift.
- DynamoDB tidak mendukung konsep SQL NULL. Anda perlu menentukan bagaimana Amazon Redshift menafsirkan nilai atribut kosong atau tidak ada isi di DynamoDB, dengan memperlakukannya sebagai NULL maupun sebagai bidang kosong.
- Jenis data DynamoDB tidak berkorespondensi langsung dengan orang-orang dari Amazon Redshift. Anda perlu memastikan bahwa setiap kolom dalam tabel Amazon Redshift memiliki jenis dan ukuran data yang benar untuk mengakomodasi data dari DynamoDB.

Berikut adalah contoh perintah COPY dari Amazon Redshift SQL:

```
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;
```

Dalam contoh ini, tabel sumber di DynamoDB adalah `my-favorite-movies-table`. Tabel target di Amazon Redshift adalah `favoritemovies`. Klausul `readratio 50` mengatur persentase throughput yang disediakan yang digunakan; dalam kasus ini, perintah COPY akan menggunakan tidak lebih dari 50 persen unit kapasitas baca yang disediakan untuk `my-favorite-movies-table`. Sebaiknya tetapkan rasio ini ke nilai yang kurang dari rata-rata penyediaan throughput yang tidak digunakan.

Untuk petunjuk mendetail tentang cara memuat data dari DynamoDB ke Amazon Redshift, lihat bagian berikut di [Panduan Developer Database Amazon Redshift](#):

- [Memuat data dari tabel DynamoDB](#)
- [Perintah COPY](#)
- [Contoh COPY](#)

Mengolah data DynamoDB dengan Apache Hive di Amazon EMR

Amazon DynamoDB terintegrasi dengan Apache Hive, aplikasi data warehousing yang berjalan pada Amazon EMR. Hive dapat membaca dan menulis data dalam tabel DynamoDB, memungkinkan Anda untuk:

- Mengkueri data DynamoDB hidup menggunakan bahasa seperti SQL (HiveQL).
- Salin data dari tabel DynamoDB ke bucket Amazon S3, dan sebaliknya.
- Salin data dari tabel DynamoDB ke Hadoop Distributed File System (HDFS) (Sistem File Terdistribusi Hadoop (HDFS)), dan sebaliknya.
- Melakukan operasi gabungan pada tabel DynamoDB.

Topik

- [Gambaran Umum](#)
- [Tutorial: Bekerja dengan Amazon DynamoDB dan Apache Hive](#)
- [Membuat tabel eksternal di Hive](#)
- [Mengolah Pernyataan HiveQL](#)
- [Melakukan kueri data dalam DynamoDB](#)
- [Menyalin data ke dan dari Amazon DynamoDB](#)
- [Penyempurnaan kinerja](#)

Gambaran Umum

Amazon EMR adalah layanan yang mempermudah proses data dalam jumlah besar dengan cepat dan hemat biaya. Untuk menggunakan Amazon EMR, Anda perlu meluncurkan kluster yang dikelola dari instans Amazon EC2 yang menjalankan kerangka kerja sumber terbuka Hadoop. Hadoop adalah aplikasi terdistribusi yang mengimplementasikan MapReduce algoritme, di mana tugas dipetakan ke beberapa simpul dalam kluster. Setiap simpul memproses kerja yang ditetapkan, yang paralel dengan simpul lain. Terakhir, output dikurangi pada simpul tunggal, sehingga menghasilkan hasil akhir.

Anda dapat memilih untuk memulai kluster Amazon EMR Anda sehingga menjadi persisten atau sementara:

- kluster persisten berjalan sampai Anda memamatkannya. kluster persisten ideal untuk analisis data, data warehousing, atau penggunaan interaktif lainnya.
- kluster transien berjalan cukup lama untuk memproses alur kerja, dan kemudian mematikan secara otomatis. kluster transien ideal untuk tugas-tugas pemrosesan berkala, seperti menjalankan skrip.

Untuk informasi tentang arsitektur dan administrasi Amazon EMR, lihat [Panduan Pengelolaan Amazon EMR](#).

Ketika Anda meluncurkan kluster Amazon EMR, Anda menentukan nomor awal dan tipe instans Amazon EC2. Anda juga menentukan aplikasi terdistribusi lainnya (selain Hadoop itu sendiri) yang ingin Anda jalankan di kluster. Aplikasi ini termasuk Hue, Mahout, Pig, Spark, dan banyak lagi.

Untuk informasi tentang aplikasi untuk Amazon EMR, lihat [Panduan Rilis Amazon EMR](#).

Tergantung pada konfigurasi kluster, Anda mungkin memiliki satu atau lebih tipe simpul berikut:

- Simpul pemimpin — Mengelola kluster, mengkoordinasikan distribusi MapReduce executable dan subset data mentah, ke inti dan tugas kelompok instans. Hal ini juga melacak status setiap tugas yang dilakukan dan memantau Health kelompok instans. Hanya ada satu simpul pemimpin dalam kluster.
- Simpul inti — Menjalankan MapReduce tugas dan menyimpan data menggunakan Sistem File Terdistribusi Hadoop (HDFS) (Sistem File Terdistribusi Hadoop (HDFS)).
- Simpul tugas (opsional) — Menjalankan MapReduce tugas.

Tutorial: Bekerja dengan Amazon DynamoDB dan Apache Hive

Dalam tutorial ini, Anda akan meluncurkan kluster Amazon EMR, dan kemudian menggunakan Apache Hive untuk memproses data yang disimpan dalam tabel DynamoDB.

Hive adalah aplikasi gudang data untuk Hadoop yang memungkinkan Anda untuk memproses dan menganalisis data dari berbagai sumber. Hive menyediakan bahasa SQL seperti, HiveQL, yang memungkinkan Anda bekerja dengan data yang disimpan secara lokal di kluster Amazon EMR atau sumber data eksternal (seperti Amazon DynamoDB).

Untuk informasi selengkapnya, lihat [tutorial Hive](#).

Topik

- [Sebelum Anda memulai](#)
- [Langkah 1: Buat pasangan kunci Amazon EC2](#)
- [Langkah 2: Luncurkan klaster Amazon EMR](#)
- [Langkah 3: Connect ke Simpul Pemimpin](#)
- [Langkah 4: Memuat data ke HDFS](#)
- [Langkah 5: Salin data ke DynamoDB](#)
- [Langkah 6: Kueri-lah data dalam tabel DynamoDB](#)
- [Langkah 7: \(Opsional\) membersihkan](#)

Sebelum Anda memulai

Dalam tutorial ini, Anda akan melakukan langkah-langkah berikut:

- Akun AWS. Jika Anda tidak memilikinya, lihat [Mendaftar untuk AWS](#).
- Klien SSH (Secure Shell). Anda menggunakan klien SSH untuk terhubung ke simpul pemimpin klaster Amazon EMR dan menjalankan perintah interaktif. Klien SSH tersedia secara default pada sebagian besar instalasi Linux, Unix, dan Mac OS X. Pengguna Windows dapat mengunduh dan menginstal klien [PuTTY](#), yang memiliki dukungan SSH.

Langkah selanjutnya

[Langkah 1: Buat pasangan kunci Amazon EC2](#)

Langkah 1: Buat pasangan kunci Amazon EC2

Pada langkah ini, Anda akan membuat pasangan kunci Amazon EC2, Anda perlu terhubung ke pemimpin simpul Amazon EMR dan menjalankan perintah Hive.

1. Masuk ke AWS Management Console dan buka konsol Amazon EC2 di <https://console.aws.amazon.com/ec2/>.
2. Pilih wilayah (misalnya, US West (Oregon)). Ini harus menjadi wilayah yang sama di mana tabel DynamoDB Anda berada.
3. Di panel navigasi, pilih pasangan kunci.
4. Pilih Buat pasangan kunci.

5. Di Nama pasangan kunci, ketik nama untuk pasangan kunci Anda (misalnya, `mykeypair`), lalu pilih Buat.
6. Unduh file kunci privat. Nama file akan berakhir dengan `.pem` (seperti `mykeypair.pem`). Simpan file kunci privat di tempat yang aman. Anda akan membutuhkannya untuk mengakses kluster Amazon EMR yang Anda luncurkan dengan pasangan kunci ini.

 Important

Jika Anda kehilangan pasangan kunci, Anda tidak dapat terhubung ke simpul pemimpin kluster Amazon EMR Anda.

Untuk informasi selengkapnya, lihat [pasangan kunci Amazon EC2](#) di Panduan Pengguna Amazon EC2 untuk Instans Linux.

Langkah selanjutnya

[Langkah 2: Luncurkan kluster Amazon EMR](#)

Langkah 2: Luncurkan kluster Amazon EMR

Pada langkah ini, Anda akan mengkonfigurasi dan meluncurkan kluster Amazon EMR. Hive dan handler penyimpanan untuk DynamoDB sudah akan diinstal pada kluster.

1. Buka konsol Amazon EMR di <https://console.aws.amazon.com/emr>.
2. Pilih Buat kluster.
3. Di halaman Buat kluster - Opsi Cepat, lakukan hal berikut:
 - a. Di Nama kluster, ketik nama untuk kluster Anda (misalnya: `My EMR cluster`).
 - b. Di pasangan kunci EC2, pilih pasangan kunci yang Anda buat sebelumnya.

Biarkan pengaturan lain pada defaultnya.

4. Pilih Buat kluster.

Ini akan memakan waktu beberapa menit untuk meluncurkan kluster Anda. Anda dapat menggunakan detail kluster di konsol Amazon EMR untuk memantau kemajuannya.

Ketika status klaster berubah menjadi `Waiting`, klaster telah siap untuk digunakan.

File log klaster dan Amazon S3

klaster Amazon EMR menghasilkan file log yang berisi informasi tentang status klaster dan debugging informasi. Pengaturan default untuk Buat klaster - Opsi Cepat termasuk menyiapkan Amazon EMR logging.

Jika salah satu tidak sudah ada, AWS Management Console membuat bucket Amazon S3. Nama bucket adalah `aws-logs-account-id-region`, tempatnya `account-id` adalah AWS Nomor rekening dan wilayah `region` di mana Anda meluncurkan klaster (misalnya, `aws-logs-123456789012-us-west-2`).

Note

Anda dapat menggunakan konsol Amazon S3 untuk melihat file log. Untuk informasi selengkapnya, lihat [Lihat File Log](#) di Panduan Pengelolaan Amazon EMR.

Anda dapat menggunakan bucket ini untuk tujuan selain pencatatan. Misalnya, Anda dapat menggunakan bucket sebagai lokasi untuk menyimpan script Hive atau sebagai tujuan ketika mengekspor data dari Amazon DynamoDB ke Amazon S3.

Langkah selanjutnya

[Langkah 3: Connect ke Simpul Pemimpin](#)

Langkah 3: Connect ke Simpul Pemimpin

Ketika status klaster Amazon EMR Anda berubah menjadi `Waiting`, Anda akan dapat connect ke simpul pemimpin menggunakan SSH dan melakukan operasi baris perintah.

1. Di konsol Amazon EMR, pilih nama klaster Anda untuk melihat statusnya.
2. Di halaman Detail Klaster, temukan bidang DNS publik pemimpin. Ini adalah nama DNS publik untuk simpul pemimpin klaster Amazon EMR Anda.
3. Di sebelah kanan nama DNS, pilih tautan SSH.
4. Ikuti instruksi di [Connect ke Pemimpin Simpul Menggunakan SSH](#).

Tergantung pada sistem operasi Anda, pilih tab Windows atau tab Mac/Linux, dan ikuti petunjuk untuk menghubungkan ke simpul pemimpin.

Setelah Anda terhubung ke simpul pemimpin menggunakan SSH atau PuTTY, Anda akan melihat prompt perintah yang mirip dengan berikut ini:

```
[hadoop@ip-192-0-2-0 ~]$
```

Langkah selanjutnya

[Langkah 4: Memuat data ke HDFS](#)

Langkah 4: Memuat data ke HDFS

Pada langkah ini, Anda akan menyalin file data ke Hadoop Distributed File System (HDFS) (Sistem File Terdistribusi Hadoop (HDFS)), dan kemudian membuat tabel Hive eksternal yang memetakan ke file data.

Unduh data sampel

1. Unduh arsip data sampel (`features.zip`):

```
wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/features.zip
```

2. Ekstrak file `features.txt` dari arsip:

```
unzip features.zip
```

3. Lihat beberapa baris pertama dari file `features.txt`:

```
head features.txt
```

Hasilnya harus terlihat serupa dengan ini:

```
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
```

File `features.txt` berisi subset data dari United States Board on Geographic Names (http://geonames.usgs.gov/domestic/download_data.htm). Bidang di setiap baris mewakili berikut ini:

- ID Fitur (pengenal unik)
 - Nama
 - Kelas (danau; hutan; pengaliran; dan sebagainya)
 - Negara Bagian
 - Garis lintang (derajat)
 - Garis bujur (derajat)
 - Tinggi (dalam kaki)
4. Di prompt perintah, masukkan perintah berikut:

```
hive
```

Prompt perintah perubahan ini: `hive>`

5. Masukkan pernyataan HiveQL berikut untuk membuat tabel Hive asli:

```
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. Masukkan pernyataan HiveQL berikut untuk memuat tabel dengan data:

```
LOAD DATA
LOCAL
INPATH './features.txt'
OVERWRITE
INTO TABLE hive_features;
```

7. Anda sekarang memiliki tabel Hive asli diisi dengan data dari file `features.txt`. Untuk memverifikasi, masukkan pernyataan HiveQL berikut:

```
SELECT state_alpha, COUNT(*)
FROM hive_features
GROUP BY state_alpha;
```

Output harus menunjukkan daftar negara bagian dan jumlah fitur geografis di masing-masing tempat.

Langkah selanjutnya

[Langkah 5: Salin data ke DynamoDB](#)

Langkah 5: Salin data ke DynamoDB

Pada langkah ini, Anda akan menyalin data dari tabel Hive (`hive_features`) ke tabel baru di DynamoDB.

1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Pilih Buat Tabel.
3. Di halaman Membuat tabel DynamoDB, lakukan hal berikut:
 - a. Di tipe Tabel **Features**.
 - b. Untuk Kunci primer, di bidang Tombol partisi, tipe **Id**. Mengatur tipe data untuk Angka.

Kosongkan Gunakan Pengaturan Default. Untuk Kapasitas yang Disediakan, ketik berikut ini:

- Unit Kapasitas Baca — 10
- Unit Kapasitas Tulis — 10

Pilih Buat.

4. Pada prompt Hive, masukkan pernyataan HiveQL berikut:

```
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  
);
```

Anda sekarang telah mendirikan pemetaan antara Hive dan tabel Fitur di DynamoDB.

5. Masukkan pernyataan HiveQL berikut untuk mengimpor data ke DynamoDB:

```
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;
```

Hive akan mengirimkan MapReduce tugas, yang akan diproses oleh kluster Amazon EMR Anda. Proses ini akan memakan waktu beberapa menit.

6. Verifikasi bahwa data telah dimuat ke DynamoDB:
 - a. Di panel navigasi konsol DynamoDB, pilih Tabel.
 - b. Pilih Tabel Fitur, lalu pilih tab Item untuk melihat data.

Langkah selanjutnya

[Langkah 6: Kueri-lah data dalam tabel DynamoDB](#)

Langkah 6: Kueri-lah data dalam tabel DynamoDB

Pada langkah ini, Anda akan menggunakan HiveQL untuk mengkueri tabel Fitur di DynamoDB. Coba kueri Hive berikut:

1. Semua tipe fitur (`feature_class`) dalam urutan abjad:

```
SELECT DISTINCT feature_class
FROM ddb_features
ORDER BY feature_class;
```

2. Semua danau yang dimulai dengan huruf "M":

```
SELECT feature_name, state_alpha
FROM ddb_features
WHERE feature_class = 'Lake'
AND feature_name LIKE 'M%'
ORDER BY feature_name;
```

3. Negara dengan setidaknya tiga fitur lebih tinggi dari satu mil (5.280 kaki):

```
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;
```

Langkah selanjutnya

[Langkah 7: \(Opsional\) membersihkan](#)

Langkah 7: (Opsional) membersihkan

Sekarang bahwa Anda telah menyelesaikan tutorial, Anda dapat terus membaca bagian ini untuk mempelajari lebih lanjut tentang bekerja dengan data DynamoDB di Amazon EMR. Anda mungkin memutuskan untuk menjaga kluster Amazon EMR Anda dan berjalan saat Anda melakukan hal ini.

Jika Anda tidak perlu kluster lagi, Anda harus mengakhiri itu dan menghapus sumber daya yang terkait. Ini akan membantu Anda menghindari biaya untuk sumber daya yang tidak Anda butuhkan.

1. Mengakhiri kluster Amazon EMR:
 - a. Buka konsol Amazon EMR di <https://console.aws.amazon.com/emr>.
 - b. Pilih kluster Amazon EMR, pilih Akhiri, lalu konfirmasi.
2. Hapus tabel Fitur di DynamoDB:
 - a. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
 - b. Di panel navigasi, pilih Tabel.
 - c. Pilih tabel Fitur. Dari menu Tindakan, pilih Hapus Tabel.
3. Hapus bucket Amazon S3 yang berisi file log Amazon EMR:
 - a. Buka konsol Amazon S3 di <https://console.aws.amazon.com/s3/>.
 - b. Dari daftar bucket, pilih `aws-logs-accountID-region`, di mana *accountID* adalah AWS nomor akun dan *wilayah* Anda adalah wilayah di mana Anda meluncurkan kluster.
 - c. Dari menu Tindakan, pilih Hapus.

Membuat tabel eksternal di Hive

Di [Tutorial: Bekerja dengan Amazon DynamoDB dan Apache Hive](#), Anda membuat tabel Hive eksternal yang dipetakan ke tabel DynamoDB. Ketika Anda mengeluarkan pernyataan HiveQL terhadap tabel eksternal, operasi baca dan tulis dilewatkan ke tabel DynamoDB.

Anda dapat memikirkan tabel eksternal sebagai pointer ke sumber data yang dikelola dan disimpan di tempat lain. Dalam hal ini, sumber data dasar adalah tabel DynamoDB. (Tabel harus sudah ada sebelumnya. Anda tidak dapat membuat, memperbarui, atau menghapus tabel DynamoDB dari dalam Hive.) Anda menggunakan pernyataan `CREATE EXTERNAL TABLE` untuk membuat tabel eksternal. Setelah itu, Anda dapat menggunakan HiveQL untuk bekerja dengan data di DynamoDB, seolah-olah data disimpan secara lokal dalam Hive.

Note

Anda dapat menggunakan pernyataan `INSERT` untuk memasukkan data ke dalam tabel eksternal dan pernyataan `SELECT` untuk memilih data darinya. Namun, Anda tidak dapat menggunakan pernyataan `UPDATE` atau `DELETE` untuk memanipulasi data dalam tabel.

Jika Anda tidak lagi membutuhkan tabel eksternal, Anda dapat menghapusnya menggunakan pernyataan `DROP TABLE`. Dalam kasus ini, `DROP TABLE` hanya menghapus tabel eksternal di Hive. Ini tidak mempengaruhi tabel DynamoDB dasar atau data.

Topik

- [BUAT sintaks TABEL EKSTERNAL](#)
- [Pemetaan tipe data](#)

BUAT sintaks TABEL EKSTERNAL

Berikut ini menunjukkan sintaks HiveQL untuk membuat tabel Hive eksternal yang memetakan ke tabel DynamoDB:

```
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..."
);
```

Baris 1 adalah awal dari pernyataan `CREATE EXTERNAL TABLE`, di mana Anda menyediakan nama tabel Hive (`hive_table`) yang ingin Anda buat.

Baris 2 menentukan kolom dan tipe data untuk `hive_table`. Anda perlu menentukan kolom dan tipe data yang sesuai dengan atribut dalam tabel DynamoDB.

Baris 3 adalah klausal `STORED BY`, di mana Anda menentukan kelas yang menangani pengelolaan data antara Hive dan tabel DynamoDB. Untuk DynamoDB, `STORED BY` harus diatur ke `'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'`.

Baris 4 adalah awal dari klausa `TBLPROPERTIES`, di mana Anda menentukan parameter berikut untuk `DynamoDBStorageHandler`:

- `dynamodb.table.name` nama dari tabel DynamoDB.

- `dynamodb.column.mapping` pasang nama kolom dalam tabel Hive dan atribut mereka yang sesuai di tabel DynamoDB. Setiap pasangan adalah bentuk `hive_column_name:dynamodb_attribute_name`, dan pasangan dipisahkan dengan koma.

Perhatikan hal-hal berikut:

- Nama kolom tabel Hive tidak harus sama dengan yang ada di tabel DynamoDB.
- Nama kolom tabel Hive tidak harus sama dengan yang ada di tabel DynamoDB.
- Tabel yang ditentukan oleh `dynamodb.table.name` harus ada di DynamoDB.
- Untuk `dynamodb.column.mapping`:
 - Anda harus memetakan atribut skema kunci untuk tabel DynamoDB. Ini termasuk kunci partisi dan kunci pengurutan (jika ada).
 - Anda tidak harus memetakan yang bukan atribut kunci dari tabel DynamoDB. Namun, Anda tidak akan melihat data apapun dari atribut-atribut ketika Anda mengkueri tabel Hive.
 - Jika tipe data dari kolom tabel Hive dan atribut DynamoDB tidak kompatibel, Anda akan melihat NULL dalam kolom ini ketika Anda mengkueri tabel Hive.

Note

Pernyataan `CREATE EXTERNAL TABLE` tidak melakukan validasi apapun pada klausul `TBLPROPERTIES`. Nilai-nilai yang Anda berikan untuk `dynamodb.table.name` dan `dynamodb.column.mapping` hanya dievaluasi oleh kelas `DynamoDBStorageHandler` Anda mencoba mengakses tabel.

Pemetaan tipe data

Tabel berikut menunjukkan tipe data DynamoDB dan tipe data Hive yang kompatibel:

Jenis data DynamoDB	Tipe Data Hive
String	STRING
Bilangan	BIGINT atau DOUBLE
Biner	BINARY

Jenis data DynamoDB	Tipe Data Hive
Set String	ARRAY<STRING>
Set Bilangan	ARRAY<BIGINT> atau ARRAY<DOUBLE>
Set Biner	ARRAY<BINARY>

Note

Tipe data DynamoDB berikut tidak didukung oleh `DynamoDBStorageHandler`, sehingga mereka tidak dapat digunakan dengan `dynamodb.column.mapping`:

- Peta
- Daftar
- Boolean
- Nol

Namun, jika Anda perlu bekerja dengan tipe data ini, Anda dapat membuat satu entitas bernama `Item` yang mewakili seluruh item DynamoDB sebagai peta string untuk kunci dan nilai di peta. Untuk informasi selengkapnya, lihat [Menyalin data tanpa pemetaan kolom](#)

Jika Anda ingin memetakan atribut DynamoDB dari tipe Bilangan, Anda harus memilih tipe Hive yang sesuai:

- Tipe `BIGINT` Hive adalah untuk bilangan bulat ditandatangani 8-byte. Itu sama dengan tipe `Long` data di Java.
- Tipe `DOUBLE` Hive untuk bilangan floating-point presisi ganda 8-bit. Itu sama dengan tipe `double` data di Java.

Jika Anda memiliki data numerik yang disimpan dalam DynamoDB yang memiliki presisi lebih tinggi dari tipe data Hive yang Anda pilih, kemudian mengakses data DynamoDB dapat menyebabkan hilangnya presisi.

Jika Anda mengekspor data tipe Biner dari DynamoDB ke (Amazon S3) atau HDFS, data disimpan sebagai string yang diencode Base64. Jika Anda mengimpor data dari Amazon S3 atau HDFS ke tipe Biner DynamoDB, Anda harus memastikan data diencode sebagai string Base64.

Mengolah Pernyataan HiveQL

Hive adalah sebuah aplikasi yang berjalan pada Hadoop, yang merupakan kerangka batch berorientasi untuk menjalankan MapReduce tugas. Ketika Anda mengeluarkan pernyataan HiveQL, Hive menentukan apakah dapat mengembalikan hasil segera atau apakah harus mengirimkan MapReduce tugas.

Misalnya, pertimbangkan opsi tabel `ddb_features` (dari [Tutorial: Bekerja dengan Amazon DynamoDB dan Apache Hive](#)). Kueri Hive berikut mencetak singkatan status dan setiap bilangan puncak:

```
SELECT state_alpha, count(*)
FROM ddb_features
WHERE feature_class = 'Summit'
GROUP BY state_alpha;
```

Hive tidak segera mengembalikan hasil. Sebaliknya, itu akan mengajukan MapReduce tugas, yang diproses oleh kerangka kerja Hadoop. Hive akan menunggu sampai tugas selesai sebelum menunjukkan hasil dari kueri:

```
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)
```

Pembatalan tugas

Ketika Hive meluncurkan tugas Hadoop, itu akan mencetak output dari tugas itu. Status penyelesaian tugas diperbarui saat tugas berlangsung. Dalam beberapa kasus, status mungkin tidak diperbarui untuk waktu yang lama. (Hal ini dapat terjadi ketika Anda mengkueri tabel DynamoDB besar yang memiliki ketetapan pengaturan kapasitas baca rendah.)

Jika Anda perlu membatalkan tugas sebelum selesai, Anda dapat mengetik **Ctrl+C** setiap saat.

Melakukan kueri data dalam DynamoDB

Contoh berikut menunjukkan beberapa cara yang dapat Anda gunakan dalam menggunakan HiveQL untuk mengkueri data yang disimpan di DynamoDB.

Contoh-contoh ini mengacu pada tabel `ddb_features` di tutorial ([Langkah 5: Salin data ke DynamoDB](#)).

Topik

- [Menggunakan fungsi agregat](#)
- [Menggunakan GROUP BY dan Klausul HAVING](#)
- [Menggabungkan dua tabel DynamoDB](#)
- [Menggabungkan tabel dari sumber yang berbeda](#)

Menggunakan fungsi agregat

HiveQL menyediakan fungsi built-in untuk meringkas nilai-nilai data. Misalnya, Anda dapat menggunakan fungsi MAX untuk menemukan nilai terbesar untuk kolom yang dipilih. Contoh berikut mengembalikan elevasi fitur tertinggi di negara bagian Colorado.

```
SELECT MAX(elev_in_ft)
```

```
FROM ddb_features
WHERE state_alpha = 'CO';
```

Menggunakan GROUP BY dan Klausul HAVING

Anda dapat menggunakan klausul GROUP BY untuk mengumpulkan data di beberapa riwayat. Hal ini sering digunakan dengan fungsi agregat seperti SUM, COUNT, MIN, atau MAX. Anda juga dapat menggunakan klausul HAVING untuk membuang hasil yang tidak memenuhi kriteria tertentu.

Contoh berikut mengembalikan daftar elevasi tertinggi dari negara-negara yang memiliki lebih dari lima fitur di tabel ddb_features.

```
SELECT state_alpha, max(elev_in_ft)
FROM ddb_features
GROUP BY state_alpha
HAVING count(*) >= 5;
```

Menggabungkan dua tabel DynamoDB

Contoh berikut memetakan tabel Hive lain (east_coast_states) ke tabel di DynamoDB. Pernyataan SELECT adalah gabungan di dua tabel ini. Gabungan dihitung pada klaster dan dikembalikan. Gabungan tidak terjadi di DynamoDB.

Pertimbangkan tabel DynamoDB EastCoastStates yang berisi data berikut:

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

Mari kita asumsikan tabel tersedia sebagai tabel eksternal Hive bernama `east_coast_states`:

```
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");
```

Gabungan berikut mengembalikan negara-negara di Pantai Timur Amerika Serikat yang memiliki setidaknya tiga fitur:

```
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;
```

Menggabungkan tabel dari sumber yang berbeda

Pada contoh berikut, `s3_east_coast_states` adalah tabel Hive terkait dengan file CSV yang disimpan di Amazon S3. Tabel `ddb_features` dikaitkan dengan data dalam DynamoDB. Contoh berikut menggabungkan dua tabel ini, mengembalikan fitur geografis dari negara-negara yang namanya dimulai dengan "New."

```
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%';
```

Menyalin data ke dan dari Amazon DynamoDB

Di [Tutorial: Bekerja dengan Amazon DynamoDB dan Apache Hive](#), Anda menyalin data dari tabel Hive asli ke tabel DynamoDB eksternal, dan kemudian mengkueri tabel DynamoDB eksternal. Tabel adalah eksternal karena ada di luar Hive. Bahkan jika Anda memasang tabel Hive yang memetakan untuk itu, tabel di DynamoDB tetap tidak akan terpengaruh.

Hive adalah solusi yang sangat baik untuk menyalin data antara tabel DynamoDB, bucket Amazon S3, tabel Hive asli, dan Hadoop Distributed File System (HDFS) (Sistem File Terdistribusi Hadoop (HDFS)). Bagian ini menyediakan contoh operasi ini.

Topik

- [Menyalin data antara DynamoDB dan tabel Hive asli](#)
- [Menyalin data antara DynamoDB dan Amazon S3](#)
- [Menyalin data antara DynamoDB dan HDFS](#)
- [Mengggunakan kompresi data](#)
- [Membaca data karakter UTF-8 yang tidak dapat dicetak](#)

Menyalin data antara DynamoDB dan tabel Hive asli

Jika Anda memiliki data dalam tabel DynamoDB, Anda dapat menyalin data ke tabel Hive asli. Ini akan menyediakan snapshot dari data, saat Anda menyalinnya.

Anda mungkin memutuskan untuk melakukan hal ini jika Anda perlu melakukan banyak pengkuerian HiveQL, tetapi tidak ingin mengkonsumsi kapasitas throughput yang ditetapkan dari DynamoDB. Karena data dalam tabel Hive asli adalah salinan data dari DynamoDB, dan bukan data “live”, kueri Anda seharusnya tidak menduga bahwa data adalah data up-to-date.

Note

Contoh di bagian ini ditulis dengan asumsi Anda mengikuti langkah-langkah di [Tutorial: Bekerja dengan Amazon DynamoDB dan Apache Hive](#) dan memiliki tabel eksternal di DynamoDB bernama `ddb_features`.

Example Dari DynamoDB untuk Tabel Hive asli

Anda dapat membuat tabel Hive asli dan mengisinya dengan data dari `ddb_features` seperti ini:

```
CREATE TABLE features_snapshot AS
SELECT * FROM ddb_features;
```

Anda kemudian dapat menyegarkan data kapan saja:

```
INSERT OVERWRITE TABLE features_snapshot
```

```
SELECT * FROM ddb_features;
```

Dalam contoh ini, subkueri `SELECT * FROM ddb_features` akan mengambil semua data dari `ddb_features`. Jika Anda hanya ingin menyalin subset dari data, Anda dapat menggunakan klausal `WHERE` dalam subkueri.

Contoh berikut membuat tabel Hive asli, yang hanya berisi beberapa atribut untuk lake dan puncak:

```
CREATE TABLE lakes_and_summits AS
SELECT feature_name, feature_class, state_alpha
FROM ddb_features
WHERE feature_class IN ('Lake', 'Summit');
```

Example Dari tabel Hive asli ke DynamoDB

Gunakan pernyataan HiveQL berikut untuk menyalin data dari tabel Hive asli untuk `ddb_features`:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM features_snapshot;
```

Menyalin data antara DynamoDB dan Amazon S3

Jika Anda memiliki data dalam tabel DynamoDB, Anda dapat menggunakan Hive untuk menyalin data ke bucket Amazon S3.

Anda mungkin melakukan ini jika Anda ingin membuat arsip data dalam tabel DynamoDB Anda. Misalnya, Anda memiliki lingkungan uji di mana Anda perlu bekerja dengan satu set dasar data uji di DynamoDB. Anda dapat menyalin data dasar ke bucket Amazon S3, dan kemudian jalankan pengujian Anda. Setelah itu, Anda dapat mengatur ulang lingkungan pengujian dengan memulihkan data dasar dari bucket Amazon S3 untuk DynamoDB.

Jika Anda bekerja melalui [Tutorial: Bekerja dengan Amazon DynamoDB dan Apache Hive](#), maka Anda sudah memiliki bucket Amazon S3 yang berisi log Amazon EMR. Anda dapat menggunakan bucket ini untuk contoh di bagian ini, jika Anda tahu jalur root untuk bucket:

1. Buka konsol Amazon EMR di <https://console.aws.amazon.com/emr>.
2. Untuk Nama, pilih klaster Anda.
3. URI tercantum di URI Log di bawah detail Konfigurasi.
4. Buat catatan tentang jalur root bucket. Konvensi penamaan adalah:

```
s3://aws-logs-accountID-region
```

di mana *accountID* adalah AWS ID akun dan wilayah adalah AWS wilayah untuk bucket.

Note

Untuk contoh ini, kita akan menggunakan subpath dalam bucket, seperti dalam contoh ini:

```
s3://aws-logs-123456789012-us-west-2/hive-test
```

Contoh di bagian ini ditulis dengan asumsi Anda mengikuti langkah-langkah ditutorial dan memiliki tabel eksternal di DynamoDB bernama `ddb_features`.

Topik

- [Menyalin data menggunakan format default Hive](#)
- [Menyalin data dengan format yang ditentukan pengguna](#)
- [Menyalin data tanpa pemetaan kolom](#)
- [Melihat data di Amazon S3](#)

Menyalin data menggunakan format default Hive

Example Dari DynamoDB ke Amazon S3

Gunakan pernyataan `INSERT OVERWRITE` untuk menulis langsung ke Amazon S3.

```
INSERT OVERWRITE DIRECTORY 's3://aws-logs-123456789012-us-west-2/hive-test'  
SELECT * FROM ddb_features;
```

File data di Amazon S3 terlihat seperti ini:

```
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
```

Setiap bidang dipisahkan oleh karakter SOH (awal mulai, 0x01). Di file, SOH muncul sebagai `^A`.

Example Dari Amazon S3 ke DynamoDB

1. Buat tabel eksternal yang menunjuk ke data yang belum diformat di Amazon S3.

```
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. Salin data ke DynamoDB.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_unformatted;
```

Menyalin data dengan format yang ditentukan pengguna

Jika Anda ingin menentukan karakter pemisah bidang Anda sendiri, Anda dapat membuat tabel eksternal yang memetakan ke bucket Amazon S3. Anda mungkin menggunakan teknik ini untuk membuat file data dengan nilai yang dipisahkan oleh koma (CSV).

Example Dari DynamoDB ke Amazon S3

1. Buat tabel eksternal Hive yang memetakan ke Amazon S3. Ketika Anda melakukan ini, pastikan bahwa tipe data konsisten dengan yang ada di tabel eksternal DynamoDB.

```
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. Salin data dari DynamoDB.

```
INSERT OVERWRITE TABLE s3_features_csv
SELECT * FROM ddb_features;
```

File data di Amazon S3 terlihat seperti ini:

```
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 Dari Amazon S3 ke DynamoDB

Dengan pernyataan HiveQL tunggal, Anda dapat mengisi tabel DynamoDB menggunakan data dari Amazon S3:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_csv;
```

Menyalin data tanpa pemetaan kolom

Anda dapat menyalin data dari DynamoDB dalam format mentah dan menulis ke Amazon S3 tanpa menentukan tipe data atau pemetaan kolom. Anda dapat menggunakan metode ini untuk membuat arsip data DynamoDB dan menyimpannya di Amazon S3.

Example Dari DynamoDB ke Amazon S3

1. Membuat tabel eksternal yang terkait dengan tabel DynamoDB Anda. (Tidak ada `dynamodb.column.mapping` dalam pernyataan HiveQL ini.)

```
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. Buat tabel eksternal lain yang terkait dengan bucket Amazon S3.

```
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. Salin data dari DynamoDB ke Amazon S3.

```
INSERT OVERWRITE TABLE s3_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

File data di Amazon S3 terlihat seperti ini:

```
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
```

Setiap bidang dimulai dengan karakter STX (awal teks, 0x02) dan berakhir dengan karakter ETX (akhir teks, 0x03). Dalam file, STX muncul sebagai **^B** dan ETX muncul sebagai **^C**.

Example Dari Amazon S3 ke DynamoDB

Dengan pernyataan HiveQL tunggal, Anda dapat mengisi tabel DynamoDB menggunakan data dari Amazon S3:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM s3_features_no_mapping;
```

Melihat data di Amazon S3

Jika Anda menggunakan SSH untuk terhubung ke simpul pemimpin, Anda dapat menggunakan AWS Command Line Interface (AWS CLI) untuk mengakses data yang ditulis Hive ke Amazon S3.

Langkah-langkah berikut ditulis dengan asumsi Anda telah menyalin data dari DynamoDB ke Amazon S3 menggunakan salah satu prosedur di bagian ini.

1. Jika Anda saat ini pada prompt perintah Hive, keluar ke prompt perintah Linux.

```
hive> exit;
```

2. Buat daftar isi direktori uji Hive di bucket Amazon S3. (Di sinilah Hive menyalin data dari DynamoDB.)

```
aws s3 ls s3://aws-logs-123456789012-us-west-2/hive-test/
```

Respons harus terlihat serupa dengan ini:

```
2016-11-01 23:19:54 81983 000000_0
```

Nama file (000000_0) adalah sistem yang dihasilkan.

3. (Opsional) Anda dapat menyalin file data dari Amazon S3 ke sistem file lokal pada simpul pemimpin. Setelah Anda melakukan ini, Anda dapat menggunakan utilitas baris perintah Linux standar untuk bekerja dengan data dalam file.

```
aws s3 cp s3://aws-logs-123456789012-us-west-2/hive-test/000000_0 .
```

Respons harus terlihat serupa dengan ini:

```
download: s3://aws-logs-123456789012-us-west-2/hive-test/000000_0
to ./000000_0
```

Note

Sistem file lokal pada simpul pemimpin memiliki kapasitas terbatas. Jangan gunakan perintah ini dengan file yang lebih besar dari ruang yang tersedia di sistem file lokal.

Menyalin data antara DynamoDB dan HDFS

Jika Anda memiliki data dalam tabel DynamoDB, Anda dapat menggunakan Hive untuk menyalin data ke Hadoop Distributed File System (HDFS) (Sistem File Terdistribusi Hadoop (HDFS)).

Anda mungkin melakukan ini jika Anda menjalankan MapReduce tugas yang memerlukan data dari DynamoDB. Jika Anda menyalin data dari DynamoDB ke HDFS, Hadoop dapat memprosesnya, menggunakan semua simpul yang tersedia di kluster Amazon EMR secara paralel. Ketika MapReduce tugas selesai, Anda kemudian dapat menulis hasil dari HDFS ke DDB.

Dalam contoh berikut, Hive akan membaca dari dan menulis ke direktori HDFS berikut: `/user/hadoop/hive-test`

Note

Contoh di bagian ini ditulis dengan asumsi Anda mengikuti langkah-langkah di [Tutorial: Bekerja dengan Amazon DynamoDB dan Apache Hive](#) dan Anda memiliki tabel eksternal di DynamoDB bernama `ddb_features`.

Topik

- [Menyalin data menggunakan format default Hive](#)
- [Menyalin data dengan format yang ditentukan pengguna](#)
- [Menyalin data tanpa pemetaan kolom](#)
- [Mengakses data dalam HDFS](#)

Menyalin data menggunakan format default Hive

Example Dari DynamoDB untuk HDFS

Gunakan pernyataan `INSERT OVERWRITE` untuk menulis langsung ke HDFS.

```
INSERT OVERWRITE DIRECTORY 'hdfs:///user/hadoop/hive-test'  
SELECT * FROM ddb_features;
```

File data di HDFS terlihat seperti ini:

```
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
```

Setiap bidang dipisahkan oleh karakter SOH (awal mulai, 0x01). Di file, SOH muncul sebagai `^A`.

Example Dari HDFS ke DynamoDB

1. Buat tabel eksternal yang memetakan data yang tidak diformat dalam HDFS.

```
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. Salin data ke DynamoDB.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_unformatted;
```

Menyalin data dengan format yang ditentukan pengguna

Jika Anda ingin menggunakan karakter pemisah bidang yang berbeda, Anda dapat membuat tabel eksternal yang memetakan ke direktori HDFS. Anda mungkin menggunakan teknik ini untuk membuat file data dengan nilai yang dipisahkan oleh koma (CSV).

Example Dari DynamoDB untuk HDFS

1. Buat tabel eksternal Hive yang memetakan untuk HDFS. Ketika Anda melakukan ini, pastikan bahwa tipe data konsisten dengan yang ada di tabel eksternal DynamoDB.

```
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. Salin data dari DynamoDB.

```
INSERT OVERWRITE TABLE hdfs_features_csv
SELECT * FROM ddb_features;
```

File data di HDFS terlihat seperti ini:

```
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 Dari HDFS ke DynamoDB

Dengan pernyataan HiveQL tunggal, Anda dapat mengisi tabel DynamoDB menggunakan data dari HDFS:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_csv;
```

Menyalin data tanpa pemetaan kolom

Anda dapat menyalin data dari DynamoDB dalam format mentah dan menulis ke HDFS tanpa menentukan tipe data atau pemetaan kolom. Anda dapat menggunakan metode ini untuk membuat arsip data DynamoDB dan menyimpannya di HDFS.

Note

Jika tabel DynamoDB Anda berisi atribut tipe Peta, Daftar, Boolean atau Null, maka ini adalah satu-satunya cara Anda dapat menggunakan Hive untuk menyalin data dari DynamoDB ke HDFS.

Example Dari DynamoDB untuk HDFS

1. Membuat tabel eksternal yang terkait dengan tabel DynamoDB Anda. (Tidak ada `dynamodb.column.mapping` dalam pernyataan HiveQL ini.)

```
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. Buat tabel eksternal lain yang terkait dengan direktori HDFS Anda.

```
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. Salin data dari DynamoDB ke HDFS.

```
INSERT OVERWRITE TABLE hdfs_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

File data di HDFS terlihat seperti ini:

```
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
```

Setiap bidang dimulai dengan karakter STX (awal teks, 0x02) dan berakhir dengan karakter ETX (akhir teks, 0x03). Dalam file, STX muncul sebagai **^B** dan ETX muncul sebagai **^C**.

Example Dari HDFS ke DynamoDB

Dengan pernyataan HiveQL tunggal, Anda dapat mengisi tabel DynamoDB menggunakan data dari HDFS:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM hdfs_features_no_mapping;
```

Mengakses data dalam HDFS

HDFS adalah sistem file terdistribusi, dapat diakses oleh semua simpul dalam kluster Amazon EMR. Jika Anda menggunakan SSH untuk terhubung ke simpul pemimpin, Anda dapat menggunakan alat baris perintah untuk mengakses data yang Hive tulis ke HDFS.

HDFS bukan hal yang sama dengan sistem file lokal pada simpul pemimpin. Anda tidak dapat bekerja dengan file dan direktori dalam HDFS menggunakan perintah Linux standar (seperti `cat`, `cp`, `mv`, atau `rm`). Sebaliknya, Anda melakukan tugas-tugas ini menggunakan perintah `hadoop fs`.

Langkah-langkah berikut ditulis dengan asumsi Anda telah menyalin data dari DynamoDB ke HDFS menggunakan salah satu prosedur di bagian ini.

1. Jika Anda saat ini pada prompt perintah Hive, keluar ke prompt perintah Linux.

```
hive> exit;
```

2. Daftar isi direktori `/pengguna/hadoop/uji-hive` di HDFS. (Di sinilah Hive menyalin data dari DynamoDB.)

```
hadoop fs -ls /user/hadoop/hive-test
```

Respons harus terlihat serupa dengan ini:

```
Found 1 items
-rw-r--r-- 1 hadoop hadoop 29504 2016-06-08 23:40 /user/hadoop/hive-test/000000_0
```

Nama file (`000000_0`) adalah sistem yang dihasilkan.

3. Lihat kandungan file:

```
hadoop fs -cat /user/hadoop/hive-test/000000_0
```

Note

Dalam contoh ini, file relatif kecil (sekitar 29 KB). Hati-hati saat Anda menggunakan perintah ini dengan file yang sangat besar atau berisi karakter yang tidak dapat dicetak.

- (Opsional) Anda dapat menyalin file data dari HDFS ke sistem file lokal pada simpul pemimpin. Setelah Anda melakukan ini, Anda dapat menggunakan utilitas baris perintah Linux standar untuk bekerja dengan data dalam file.

```
hadoop fs -get /user/hadoop/hive-test/000000_0
```

Perintah ini tidak akan menimpa file.

Note

Sistem file lokal pada simpul pemimpin memiliki kapasitas terbatas. Jangan gunakan perintah ini dengan file yang lebih besar dari ruang yang tersedia di sistem file lokal.

Menggunakan kompresi data

Bila Anda menggunakan Hive untuk menyalin data di antara sumber data yang berbeda, Anda dapat meminta kompresi on-the-fly data. Hive menyediakan beberapa codec kompresi. Anda dapat memilih salah satu selama sesi Hive Anda. Saat melakukannya, data dikompres dalam format yang ditentukan.

Contoh berikut kompres data menggunakan algoritme Lempel-Ziv-Oberhumer (LZO).

```
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;
```

File yang dihasilkan di Amazon S3 pada akhirnya akan memiliki nama yang dihasilkan sistem dengan `.lzo` (misalnya, `8d436957-57ba-4af7-840c-96c2fc7bb6f5-000000.lzo`).

Codec kompresi yang tersedia adalah:

- `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`

Membaca data karakter UTF-8 yang tidak dapat dicetak

Untuk membaca dan menulis data karakter UTF-8 yang tidak dapat dicetak, Anda dapat menggunakan klausa `STORED AS SEQUENCEFILE` saat Anda membuat tabel Hive. A `SequenceFile` adalah format file biner Hadoop. Anda perlu menggunakan Hadoop untuk membaca file ini. Contoh berikut menunjukkan cara mengekspor data dari DynamoDB ke Amazon S3. Anda dapat menggunakan fungsi ini untuk menangani karakter yang diekode UTF-8 yang tidak dapat dicetak.

```
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;
```

Penyempurnaan kinerja

Ketika Anda membuat tabel eksternal Hive yang memetakan ke tabel DynamoDB, Anda tidak mengkonsumsi pembacaan atau penulisan kapasitas dari DynamoDB. Namun, aktivitas membaca dan menulis pada tabel Hive (seperti `INSERT` atau `SELECT`) diterjemahkan langsung menjadi operasi membaca dan menulis pada tabel DynamoDB dasar.

Apache Hive di Amazon EMR mengimplementasikan logika sendiri untuk menyeimbangkan beban I/O pada tabel DynamoDB dan berusaha untuk meminimalkan kemungkinan melebihi throughput yang ditetapkan tabel ini. Pada akhir setiap kueri Hive, Amazon EMR mengembalikan metrik waktu

proses, termasuk jumlah kali throughput Anda yang ditetapkan terlampaui. Anda dapat menggunakan informasi ini, bersama dengan CloudWatch metrik di tabel DynamoDB, untuk meningkatkan kinerja dalam permintaan berikutnya.

Konsol Amazon EMR menyediakan alat pemantauan dasar untuk klaster Anda. Untuk informasi selengkapnya, lihat [Lihat dan Pantau klaster](#) di Panduan Pengelolaan Amazon EMR.

Anda juga dapat memantau kinerja klaster dan Hadoop Anda menggunakan alat berbasis web, seperti Hue, Ganglia, dan antarmuka web Hadoop. Untuk informasi selengkapnya, lihat [Lihat Antarmuka Web yang di-host di klaster Amazon EMR](#) di Panduan Pengelolaan Amazon EMR.

Bagian ini menjelaskan langkah-langkah yang dapat Anda ambil untuk penyetelan kinerja operasi Hive pada tabel DynamoDB eksternal.

Topik

- [Throughput yang disediakan DynamoDB](#)
- [Menyesuaikan pemeta](#)
- [Topik Tambahan](#)

Throughput yang disediakan DynamoDB

Ketika Anda mengeluarkan pernyataan HiveQL terhadap tabel DynamoDB eksternal, kelas `DynamoDBStorageHandler` membuat permintaan API level rendah DynamoDB menjadi sesuai, yang mengkonsumsi throughput yang disediakan. Jika tidak ada cukup kapasitas membaca atau menulis pada tabel DynamoDB, permintaan akan di-throttle, sehingga kinerja HiveQL menjadi lambat. Untuk alasan ini, Anda harus memastikan bahwa tabel memiliki kapasitas throughput yang cukup.

Misalnya, anggap bahwa Anda telah menetapkan 100 unit kapasitas baca untuk tabel DynamoDB Anda. Ini akan memungkinkan Anda membaca 409.600 byte per detik (100×4 KB ukuran unit kapasitas baca). Sekarang anggaplah bahwa tabel berisi 20 GB data (21.474.836.480 byte) dan Anda ingin menggunakan pernyataan `SELECT` untuk memilih semua data menggunakan HiveQL. Anda dapat memperkirakan berapa lama kueri akan berjalan seperti ini:

$$21.474.836.480/409.600 = 52.429 \text{ detik} = 14,56 \text{ jam}$$

Dalam skenario ini, tabel DynamoDB adalah kemacetan. Ini tidak akan membantu untuk menambahkan lebih banyak simpul Amazon EMR, karena throughput Hive dibatasi hanya 409.600 byte per detik. Satu-satunya cara untuk mengurangi waktu yang dibutuhkan untuk pernyataan `SELECT` adalah untuk meningkatkan kapasitas baca yang ditetapkan tabel DynamoDB.

Anda dapat melakukan perhitungan yang sama untuk memperkirakan berapa lama waktu yang dibutuhkan untuk data massal beban ke dalam tabel eksternal Hive yang dipetakan ke tabel DynamoDB. Tentukan jumlah total unit kapasitas tulis yang dibutuhkan per item (kurang dari 1KB = 1, 1-2KB = 2, dll), dan kalikan dengan jumlah item yang akan dimuat. Ini akan memberi Anda jumlah unit kapasitas tulis yang diperlukan. Bagilah angka itu dengan jumlah unit kapasitas tulis yang dialokasikan per detik. Ini akan menghasilkan jumlah detik yang dibutuhkan untuk memuat tabel.

Anda harus memantau CloudWatch metrik untuk tabel Anda. Untuk ikhtisar singkat di konsol DynamoDB, pilih tabel Anda dan kemudian pilih tab Metrik. Dari sini, Anda dapat melihat unit kapasitas baca tulis dikonsumsi dan permintaan baca dan tulis di-throttle.

kapasitas baca

Amazon EMR mengelola beban permintaan terhadap tabel DynamoDB Anda, sesuai dengan pengaturan throughput yang ditetapkan tabel ini. Namun, jika Anda melihat sejumlah besar pesan `ProvisionedThroughputExceeded` dalam output tugas, Anda dapat menyesuaikan laju baca default. Untuk melakukannya, Anda dapat memodifikasi variabel konfigurasi `dynamodb.throughput.read.percent`. Anda dapat menggunakan perintah SET untuk mengatur variabel ini pada prompt perintah Hive:

```
SET dynamodb.throughput.read.percent=1.0;
```

Variabel ini berlanjut untuk sesi Hive saat ini saja. Jika Anda keluar dari Hive dan kembali lagi setelah beberapa saat, `dynamodb.throughput.read.percent` akan kembali ke nilai default.

Nilai dari `dynamodb.throughput.read.percent` bisa antara 0.1 dan 1.5, inklusif. 0.5 mewakili tingkat baca default, yang berarti bahwa Hive akan mencoba untuk mengkonsumsi setengah dari kapasitas baca tabel. Jika Anda meningkatkan nilai di atas 0.5, Hive akan meningkatkan tingkat permintaan; menurunkan nilai di bawah 0.5 menurunkan tingkat permintaan baca. (Tingkat baca sebenarnya akan bervariasi, tergantung pada faktor-faktor seperti apakah ada distribusi kunci seragam dalam tabel DynamoDB.)

Jika Anda melihat bahwa Hive sering mengurangi kapasitas baca yang ditetapkan tabel, atau jika permintaan baca Anda di-throttle terlalu banyak, cobalah mengurangi `dynamodb.throughput.read.percent` di bawah 0.5. Jika Anda memiliki kapasitas baca yang cukup dalam tabel dan ingin operasi HiveQL lebih responsif, Anda dapat mengatur nilai di atas 0.5.

Kapasitas Tulis

Amazon EMR mengelola beban permintaan terhadap tabel DynamoDB Anda, sesuai dengan pengaturan throughput yang ditetapkan tabel ini. Namun, jika Anda melihat sejumlah besar pesan `ProvisionedThroughputExceeded` dalam output tugas, Anda dapat menyesuaikan laju tulis default. Untuk melakukannya, Anda dapat memodifikasi variabel konfigurasi `dynamodb.throughput.write.percent`. Anda dapat menggunakan perintah SET untuk mengatur variabel ini pada prompt perintah Hive:

```
SET dynamodb.throughput.write.percent=1.0;
```

Variabel ini berlanjut untuk sesi Hive saat ini saja. Jika Anda keluar dari Hive dan kembali lagi setelah beberapa saat, `dynamodb.throughput.write.percent` akan kembali ke nilai default.

Nilai dari `dynamodb.throughput.write.percent` bisa antara 0.1 dan 1.5, inklusif. 0.5 mewakili tingkat tulis default, yang berarti bahwa Hive akan mencoba untuk mengkonsumsi setengah dari kapasitas tulis tabel. Jika Anda meningkatkan nilai di atas 0.5, Hive akan meningkatkan tingkat permintaan; menurunkan nilai di bawah 0.5 menurunkan tingkat permintaan tulis. (Laju tulis sebenarnya akan bervariasi, tergantung pada faktor-faktor seperti apakah ada distribusi kunci seragam dalam tabel DynamoDB.)

Jika Anda melihat bahwa Hive sering mengurangi kapasitas tulis yang ditetapkan tabel, atau jika permintaan baca Anda di-throttle terlalu banyak, cobalah mengurangi `dynamodb.throughput.write.percent` di bawah 0.5. Jika Anda memiliki kapasitas tulis yang cukup dalam tabel dan ingin operasi HiveQL lebih responsif, Anda dapat mengatur nilai di atas 0.5.

Ketika Anda menulis data ke DynamoDB menggunakan Hive, pastikan bahwa jumlah unit kapasitas tulis lebih besar dari jumlah pemeta di klaster. Sebagai contoh, pertimbangkan sebuah klaster Amazon EMR yang terdiri dari 10 simpul m1.xlarge. Tipe simpul m1.xlarge menyediakan 8 tugas pemeta, sehingga klaster akan memiliki total 80 pemeta (10 × 8). Jika tabel DynamoDB Anda memiliki kurang dari 80 unit kapasitas tulis, maka operasi tulis Hive bisa mengkonsumsi semua throughput tulis untuk tabel itu.

Untuk menentukan jumlah pemeta untuk Amazon EMR tipe simpul, lihat [Konfigurasi Tugas](#) di Panduan Developer Amazon EMR.

Untuk informasi selengkapnya tentang pemeta, lihat [Menyesuaikan pemeta](#).

Menyesuaikan pemeta

Ketika Hive meluncurkan tugas Hadoop, tugas diproses oleh satu tugas pemeta atau lebih. Dengan asumsi bahwa tabel DynamoDB Anda memiliki kapasitas throughput yang cukup, Anda dapat mengubah jumlah pemeta di klaster, ini berpotensi meningkatkan kinerja.

Note

Jumlah tugas pemeta digunakan dalam tugas Hadoop dipengaruhi oleh pembagian input, di mana Hadoop membagi data menjadi blok logis. Jika Hadoop tidak melakukan split input yang cukup, maka operasi tulis Anda mungkin tidak dapat mengkonsumsi semua throughput tulis yang tersedia di tabel DynamoDB.

Menambah jumlah pemeta

Setiap pemeta dalam Amazon EMR memiliki tingkat baca maksimum 1 MiB per detik. Jumlah pemeta di klaster tergantung pada ukuran simpul di klaster Anda. (Untuk informasi tentang ukuran simpul dan jumlah pemeta per simpul, lihat [Konfigurasi Tugas](#) di Panduan Developer Amazon EMR.)

Jika tabel DynamoDB Anda memiliki kapasitas throughput yang cukup untuk membaca, Anda dapat mencoba meningkatkan jumlah pemeta dengan melakukan salah satu dari berikut ini:

- Meningkatkan ukuran simpul dalam klaster Anda. Sebagai contoh, jika klaster Anda menggunakan simpul `m1.large` (tiga pemeta per simpul), Anda dapat mencoba meningkatkan ke simpul `m1.xlarge` (delapan pemeta per simpul).
- Meningkatkan jumlah simpul dalam klaster Anda. Misalnya, jika Anda memiliki tiga klaster simpul `m1.xlarge`, Anda memiliki total 24 pemeta yang tersedia. Jika Anda ingin dua kali lipat ukuran klaster, dengan tipe simpul yang sama, Anda akan memiliki 48 pemeta.

Anda dapat menggunakan AWS Management Console untuk mengelola ukuran atau jumlah simpul dalam klaster Anda. (Anda mungkin perlu memulai ulang klaster untuk menerapkan perubahan ini.)

Cara lain untuk meningkatkan jumlah pemeta adalah dengan memodifikasi `mapred.tasktracker.map.tasks.maximum` konfigurasi parameter Hadoop. (Ini adalah parameter Hadoop, bukan parameter Hive. Anda tidak dapat memodifikasi secara interaktif dari prompt perintah.) Jika Anda meningkatkan nilai `mapred.tasktracker.map.tasks.maximum`, Anda dapat meningkatkan jumlah pemeta tanpa meningkatkan ukuran atau jumlah simpul. Namun, simpul klaster kemungkinan bisa kehabisan memori jika Anda menetapkan nilai terlalu tinggi.

Anda menetapkan nilai untuk `mapred.tasktracker.map.tasks.maximum` sebagai tindakan bootstrap ketika Anda pertama kali meluncurkan kluster Amazon EMR Anda. Untuk informasi selengkapnya, lihat [\(Opsional\) Buat Tindakan Bootstrap untuk Menginstal Perangkat Lunak Tambahan](#) di Panduan Pengelolaan Amazon EMR.

Mengurangi jumlah pemeta

Jika Anda menggunakan pernyataan `SELECT` untuk memilih data dari tabel Hive eksternal yang memetakan untuk DynamoDB, tugas Hadoop dapat menggunakan sebanyak tugas yang diperlukan, hingga jumlah maksimum pemeta di kluster. Dalam skenario ini, ada kemungkinan bahwa permintaan Hive berjalan lama dapat mengkonsumsi semua kapasitas baca ditetapkan tabel DynamoDB, berdampak negatif terhadap pengguna lain.

Anda dapat menggunakan parameter `dynamodb.max.map.tasks` untuk menetapkan batas atas untuk tugas peta:

```
SET dynamodb.max.map.tasks=1
```

Nilai ini harus sama dengan atau lebih besar dari 1. Ketika Hive memproses kueri Anda, tugas Hadoop yang dihasilkan akan menggunakan tidak lebih dari `dynamodb.max.map.tasks` saat melakukan pembacaan dari tabel DynamoDB.

Topik Tambahan

Berikut ini adalah beberapa cara untuk menyetel aplikasi yang menggunakan Hive untuk mengakses DynamoDB.

Durasi coba lagi

Secara default, Hive akan menjalankan kembali tugas Hadoop jika sama sekali belum mengembalikan hasil apapun dari DynamoDB dalam waktu dua menit. Anda dapat menyesuaikan interval ini dengan memodifikasi parameter `dynamodb.retry.duration`:

```
SET dynamodb.retry.duration=2;
```

Nilai harus bilangan bulat nol, mewakili jumlah menit dalam interval coba lagi. Default `dynamodb.retry.duration` adalah 2 (menit).

Permintaan data paralel

Beberapa permintaan data, baik lebih dari satu pengguna atau lebih dari satu aplikasi ke satu tabel dapat menguras throughput yang disediakan pembacaan dan kinerja menjadi lambat.

Durasi proses

konsistensi data di DynamoDB tergantung pada pesanan operasi baca dan tulis pada setiap simpul. Saat kueri Hive sedang berjalan, aplikasi lain mungkin memuat data baru ke dalam tabel DynamoDB atau memodifikasi atau menghapus data yang ada. Dalam hal ini, hasil kueri Hive mungkin tidak mencerminkan perubahan yang dibuat untuk data saat kueri sedang berjalan.

Waktu permintaan

Penjadwalan kueri Hive yang mengakses tabel DynamoDB ketika ada permintaan yang lebih rendah pada tabel DynamoDB meningkatkan kinerja. Misalnya, jika sebagian besar pengguna aplikasi Anda tinggal di San Francisco, Anda dapat memilih untuk mengekspor data harian pada pukul 4:00 pagi PST ketika sebagian besar pengguna masih sedang tidur dan tidak memperbarui riwayat dalam database DynamoDB Anda.

Integrasi dengan Amazon S3

Kemampuan impor dan ekspor Amazon DynamoDB memberikan cara sederhana dan efisien untuk memindahkan data antara tabel Amazon S3 dan DynamoDB tanpa menulis kode apa pun.

Fitur impor dan ekspor DynamoDB membantu Anda memindahkan, mengubah, dan menyalin akun tabel DynamoDB. Anda dapat mengimpor dari sumber S3, dan Anda dapat mengekspor data tabel DynamoDB ke Amazon S3 dan menggunakan layanan AWS seperti Athena, Amazon, dan untuk menganalisis data Anda serta mengekstrak wawasan yang dapat ditindaklanjuti. SageMaker AWS Lake Formation Anda juga dapat mengimpor data langsung ke tabel DynamoDB baru untuk membangun aplikasi baru dengan performa satu digit milidetik dalam skala besar, memfasilitasi berbagi data antara tabel dan akun, serta menyederhanakan rencana pemulihan bencana dan kelangsungan bisnis Anda.

Topik

- [Impor data DynamoDB dari Amazon S3: cara kerjanya](#)
- [Ekspor data DynamoDB ke Amazon S3: cara kerjanya](#)

Impor data DynamoDB dari Amazon S3: cara kerjanya

Untuk mengimpor data ke DynamoDB, data Anda harus berada dalam bucket Amazon S3 dalam format CSV, DynamoDB JSON, atau Amazon Ion. Data dapat dikompresi dalam format ZSTD atau GZIP, atau dapat langsung diimpor dalam bentuk tidak terkompresi. Sumber data dapat berupa satu objek Amazon S3 atau beberapa objek Amazon S3 yang menggunakan awalan yang sama.

Data Anda akan diimpor ke tabel DynamoDB baru, yang akan dibuat saat Anda memulai permintaan impor. Anda dapat membuat tabel ini dengan indeks sekunder, lalu mengkueri dan memperbarui data Anda di seluruh indeks primer dan sekunder segera setelah impor selesai. Anda juga dapat menambahkan replika tabel global setelah impor selesai.

Note

Selama proses impor Amazon S3, DynamoDB membuat tabel target baru yang akan diimpor. Impor ke tabel yang ada saat ini tidak didukung oleh fitur ini.

Impor dari Amazon S3 tidak menggunakan kapasitas tulis pada tabel baru, sehingga Anda tidak perlu menyediakan kapasitas tambahan apa pun untuk mengimpor data ke DynamoDB. Harga impor data didasarkan pada ukuran data sumber yang tidak terkompresi di Amazon S3, yang diproses sebagai hasil impor. Item yang diproses namun gagal dimuat ke dalam tabel karena format atau ketidakkonsistenan lainnya dalam data sumber juga ditagih sebagai bagian dari proses impor. Lihat [harga Amazon DynamoDB](#) untuk detailnya.

Anda dapat mengimpor data dari bucket Amazon S3 yang dimiliki oleh akun berbeda jika Anda memiliki izin yang benar untuk membaca dari bucket tertentu. Tabel baru mungkin juga berada di Wilayah yang berbeda dari bucket Amazon S3 sumber. Untuk informasi selengkapnya, lihat [Pengaturan dan izin Amazon Simple Storage Service](#).

Waktu impor berhubungan langsung dengan karakteristik data Anda di Amazon S3. Hal ini mencakup ukuran data, format data, skema kompresi, keseragaman distribusi data, jumlah objek Amazon S3, dan variabel terkait lainnya. Secara khusus, kumpulan data dengan kunci yang terdistribusi secara seragam akan lebih cepat diimpor dibandingkan kumpulan data yang miring. Misalnya, jika kunci indeks sekunder Anda menggunakan bulan dalam setahun untuk mempartisi, dan semua data Anda berasal dari bulan Desember, maka mengimpor data ini mungkin memerlukan waktu yang jauh lebih lama.

Atribut yang terkait dengan kunci diharapkan unik pada tabel dasar. Jika ada kunci yang tidak unik, impor akan menimpa item terkait hingga hanya penempatan terakhir yang tersisa. Misalnya, jika kunci primer adalah bulan dan beberapa item disetel ke bulan September, setiap item baru akan menimpa item yang ditulis sebelumnya dan hanya satu item dengan kunci primer "bulan" yang diatur ke bulan September yang akan tersisa. Dalam kasus tersebut, jumlah item yang diproses dalam deskripsi tabel impor tidak akan cocok dengan jumlah item dalam tabel target.

AWS CloudTrail mencatat semua tindakan konsol dan API untuk impor tabel. Untuk informasi selengkapnya, lihat [Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail](#).

Video berikut adalah pengantar untuk mengimpor langsung dari Amazon S3 ke DynamoDB.

[Mengimpor dari Amazon S3](#)

Topik

- [Meminta impor tabel di DynamoDB](#)
- [Format impor Amazon S3 untuk DynamoDB](#)
- [Kuota format impor dan validasi](#)
- [Praktik terbaik untuk mengimpor dari Amazon S3 ke DynamoDB](#)

Meminta impor tabel di DynamoDB

Import DynamoDB memungkinkan Anda mengimpor data dari bucket Amazon S3 ke tabel DynamoDB baru. Anda dapat meminta impor tabel menggunakan [Konsol DynamoDB](#), [CLI](#), [CloudFormation](#) atau [DynamoDB API](#).

Jika ingin menggunakan AWS CLI, Anda harus mengonfigurasinya terlebih dahulu. Untuk informasi selengkapnya, lihat [Mengakses DynamoDB](#).

Note

- Fitur Tabel Impor berinteraksi dengan beberapa AWS Layanan seperti Amazon S3 dan CloudWatch. Sebelum memulai impor, pastikan bahwa pengguna atau peran yang memanggil API impor memiliki izin untuk semua layanan dan sumber daya yang bergantung pada fitur tersebut.
- Jangan memodifikasi objek Amazon S3 saat impor sedang berlangsung, karena ini dapat menyebabkan operasi gagal atau dibatalkan.

Untuk informasi selengkapnya tentang kesalahan dan pemecahan masalah, lihat [Kuota format impor dan validasi](#)

Topik

- [Menyiapkan izin IAM](#)
- [Meminta impor menggunakan AWS Management Console](#)
- [Mendapatkan detail tentang impor sebelumnya di AWS Management Console](#)
- [Meminta impor menggunakan AWS CLI](#)
- [Mendapatkan detail tentang impor sebelumnya di AWS CLI](#)

Menyiapkan izin IAM

Anda dapat mengimpor data dari bucket Amazon S3 yang memungkinkan Anda membacanya. bucket sumber tidak harus berada di Wilayah yang sama atau memiliki pemilik yang sama dengan tabel sumber. Anda AWS Identity and Access Management (IAM) harus menyertakan tindakan yang relevan pada bucket Amazon S3 sumber, dan diperlukan CloudWatch izin untuk memberikan informasi debugging. Contoh kebijakan ditunjukkan di bawah ini.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDynamoDBImportAction",
      "Effect": "Allow",
      "Action": [
        "dynamodb:ImportTable",
        "dynamodb:DescribeImport",
        "dynamodb:ListImports"
      ],
      "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/*"
  }
]
}

```

Izin Amazon S3

Saat memulai impor di sumber bucket Amazon S3 yang dimiliki oleh akun lain, pastikan peran atau pengguna memiliki akses ke objek Amazon S3. Anda dapat memeriksanya dengan menjalankan `Amazon S3GetObject` perintah dan menggunakan kredensialnya. Saat menggunakan API, parameter pemilik bucket Amazon S3 default ke ID akun pengguna saat ini. Untuk impor lintas akun, pastikan parameter ini diisi dengan benar dengan ID akun pemilik bucket. Kode berikut adalah contoh kebijakan bucket Amazon S3 di akun sumber.

```

{
  "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:::awsexamplebucket1/*"
    }
  ]
}

```

```
}  
  ]  
}
```

AWS Key Management Service

Saat membuat tabel baru untuk impor, jika Anda memilih enkripsi pada kunci istirahat yang tidak dimiliki oleh DynamoDB maka Anda harus memberikan AWS KMS izin yang diperlukan untuk mengoperasikan tabel DynamoDB yang dienkripsi dengan kunci yang dikelola pelanggan. Untuk informasi lebih lanjut, lihat [Otorisasi penggunaan Anda AWS KMS kunci](#). Jika objek Amazon S3 dienkripsi dengan enkripsi sisi server KMS (SSE-KMS), pastikan bahwa peran atau pengguna yang memulai impor memiliki akses untuk mendekripsi menggunakan AWS KMS kunci. Fitur ini tidak mendukung objek Amazon S3 yang dienkripsi oleh kunci enkripsi yang disediakan pelanggan (SSE-C).

CloudWatch izin

Peran atau pengguna yang memulai impor perlu membuat dan mengelola izin untuk grup log dan aliran log yang terkait dengan impor.

Meminta impor menggunakan AWS Management Console

Contoh berikut menunjukkan bagaimana menggunakan konsol DynamoDB untuk mengimpor data yang ada ke tabel baru bernama `MusicCollection`.

Untuk meminta impor tabel

1. Masuk ke AWS Management Console, lalu buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.
2. Di panel navigasi di sisi kiri konsol, pilih `Impor` dari `S3`.
3. Pada halaman yang muncul, pilih `Impor` dari `S3`.
4. Pilih `Impor` dari `S3`.
5. Masukkan URL sumber Amazon S3.

Jika bucket sumber dimiliki oleh akun Anda, Anda dapat menemukannya dengan menggunakan bucket sumber Jelajahi S3 tombol. Bergantian, Anda dapat memasukkan URL bucket di `buckets3://bucket/prefixformat`. The `prefix` adalah awalan kunci Amazon S3 yang merupakan nama objek Amazon S3 yang ingin Anda impor, atau awalan kunci yang dibagikan oleh semua objek Amazon S3 yang ingin Anda impor.

6. Tentukan apakah Anda adalah Pemilik bucket S3. Jika bucket sumber dimiliki oleh akun lain, pilih Berbeda AWS Akun. Kemudian masukkan ID akun pemilik bucket.
7. Di bawah Impor kompresi file, pilih salah satu Tidak ada kompresi, GZIP atau ZSTD sesuai.
8. Pilih format file Impor yang sesuai. Pilihannya adalah DynamoDB JSON, Amazon Ion atau CSV. Jika Anda memilih CSV, Anda akan memiliki dua opsi tambahan: **Sundulan CSV** dan **Karakter pembatas CSV**.

Untuk **Sundulan CSV**, pilih apakah header akan diambil dari baris pertama file atau disesuaikan. Jika Anda memilih **Sesuaikan header** Anda, Anda dapat menentukan nilai header yang ingin Anda impor. Header CSV yang ditentukan oleh metode ini peka huruf besar/kecil dan diharapkan berisi kunci dari tabel target.

Untuk **Karakter pembatas CSV**, Anda mengatur karakter yang akan memisahkan item. Koma dipilih secara default. Jika Anda memilih **Karakter pembatas kustom**, pembatas harus cocok dengan pola regex: `[, ; : | \t]`.

9. Pilih **Berikutnya** tombol dan pilih opsi untuk tabel baru yang akan dibuat untuk menyimpan data Anda.

Note

Primary Key dan Sort Key harus cocok dengan atribut dalam file, atau impor akan gagal. Atributnya peka huruf besar/kecil.

10. Pilih **Berikutnya** lagi untuk meninjau opsi impor Anda, lalu klik **Impor** untuk memulai tugas impor. Pertama-tama Anda akan melihat tabel baru Anda yang tercantum di “Tabel” dengan status “Membuat”. Pada saat ini meja tidak dapat diakses.
11. Setelah impor selesai, status akan ditampilkan sebagai “Aktif” dan Anda dapat mulai menggunakan tabel.

Mendapatkan detail tentang impor sebelumnya di AWS Management Console

Anda dapat menemukan informasi tentang tugas impor yang pernah Anda jalankan di masa lalu dengan mengklik **Impor** dari S3 di bilah sisi navigasi, lalu pilih **Import** tab. Panel impor berisi daftar semua impor yang Anda buat dalam 90 hari terakhir. Memilih ARN tugas yang tercantum di tab **Impor** akan mengambil informasi tentang impor itu, termasuk pengaturan konfigurasi lanjutan yang Anda pilih.

Meminta impor menggunakan AWS CLI

Contoh berikut mengimpor data berformat CSV dari bucket S3 yang disebut bucket dengan awalan awalan ke tabel baru yang disebut target-table.

```
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

Jika Anda memilih untuk mengenkripsi impor Anda menggunakan kunci yang dilindungi oleh AWS Key Management Service (AWS KMS), kuncinya harus di Wilayah yang sama dengan bucket Amazon S3 tujuan.

Mendapatkan detail tentang impor sebelumnya di AWS CLI

Anda dapat menemukan informasi tentang tugas impor yang pernah Anda jalankan di masa lalu dengan menggunakan `list-imports` perintah. Perintah ini mengembalikan daftar semua impor yang Anda buat dalam 90 hari terakhir. Perhatikan bahwa meskipun metadata tugas impor berakhir setelah 90 hari dan pekerjaan yang lebih lama dari itu tidak lagi ditemukan dalam daftar ini, DynamoDB tidak menghapus objek di bucket Amazon S3 Anda atau tabel yang lebih lama dari itu tidak lagi ditemukan dalam daftar ini, DynamoDB tidak menghapus objek di bucket Amazon S3 Anda atau tabel yang lebih lama dari itu tidak lagi ditemukan dalam daftar ini, DynamoDB tidak menghapus objek di bucket Amazon S3 Anda atau tabel yang lebih lama dari itu tidak lagi ditemukan dalam daftar ini.

```
aws dynamodb list-imports
```

Untuk mengambil informasi rinci tentang tugas impor tertentu, termasuk pengaturan konfigurasi lanjutan, gunakan `describe-import` perintah.

```
aws dynamodb describe-import \
    --import-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/exp
```

Format impor Amazon S3 untuk DynamoDB

DynamoDB dapat mengimpor data dalam tiga format: CSV, DynamoDB JSON, dan Amazon Ion.

Topik

- [CSV](#)
- [DynamoDB Json](#)
- [Amazon Ion](#)

CSV

File dalam format CSV terdiri dari beberapa item yang dibatasi oleh baris baru. Secara default, DynamoDB menafsirkan baris pertama dari file impor sebagai header dan mengharapkan kolom akan dibatasi oleh koma. Anda juga dapat menentukan header yang akan diterapkan, asalkan sesuai dengan jumlah kolom dalam file. Jika Anda mendefinisikan header secara eksplisit, baris pertama file akan diimpor sebagai nilai.

Note

Saat mengimpor dari file CSV, semua kolom selain rentang hash dan kunci tabel dasar serta indeks sekunder Anda diimpor sebagai string DynamoDB.

Menghindari tanda kutip ganda

Setiap karakter tanda kutip ganda yang ada di file CSV harus dihindari. Jika tidak dihindari, seperti pada contoh berikut ini, impor akan gagal:

```
id,value
"123",Women's Full Lenth Dress
```

Impor yang sama ini akan berhasil jika tanda kutip dihindari dengan menambah dua rangkaian tanda kutip ganda:

```
id,value
""123"" ,Women's Full Lenth Dress
```

Setelah teks dihindari dan diimpor dengan benar, teks tersebut akan muncul seperti di file CSV asli:

```
id,value
"123",Women's Full Lenth Dress
```

DynamoDB Json

File dalam format DynamoDB JSON dapat terdiri dari beberapa objek Item. Setiap objek individual berada dalam format JSON marshall standar DynamoDB, dan baris baru digunakan sebagai pembatas item. Sebagai fitur tambahan, ekspor dari titik waktu tertentu didukung sebagai sumber impor secara default.

Note

Baris baru digunakan sebagai pembatas item untuk file dalam format DynamoDB JSON dan tidak boleh digunakan dalam objek item.

```
{
  "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

Baris baru digunakan sebagai pembatas item untuk file dalam format DynamoDB JSON dan tidak boleh digunakan dalam objek item.

```
{
  "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": "444-4444444444"  
      },  
      "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 Ion](#) adalah format serialisasi data hierarkis yang kaya jenis, dapat dijelaskan sendiri, yang dibuat untuk mengatasi tantangan perkembangan pesat, pemisahan, dan efisiensi yang dihadapi setiap hari saat merekayasa arsitektur berorientasi layanan berskala besar.

Saat Anda mengimpor data dalam format Ion, tipe data Ion dipetakan ke tipe data DynamoDB di tabel DynamoDB baru.

	Konversi tipe data Ion ke DynamoDB	B
1	Ion Data Type	DynamoDB Representation
2	string	String (s)
3	bool	Boolean (BOOL)
4	decimal	Number (N)
5	blob	Binary (B)

	Konversi tipe data Ion ke DynamoDB	B
6	<code>list</code> (with type annotation <code>\$dynamodb_SS</code> , <code>\$dynamodb_NS</code> , or <code>\$dynamodb_BS</code>)	Set (SS, NS, BS)
7	<code>list</code>	List
8	<code>struct</code>	Map

Item dalam file Ion dibatasi oleh baris baru. Setiap baris dimulai dengan penanda versi Ion, diikuti dengan item dalam format Ion.

Note

Dalam contoh berikut, item dari file dalam format Ion telah diformat pada beberapa baris demi keterbacaan.

```
$ion_1_0 {
  Item:{
    Authors:$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"
  }
}
```

Kuota format impor dan validasi

Kuota impor

Impor DynamoDB dari Amazon S3 dapat mendukung hingga 50 pekerjaan impor bersamaan dengan total ukuran objek sumber impor sebesar 15 TB sekaligus di wilayah us-east-1, us-west-2, dan eu-west-1. Di seluruh wilayah lainnya, didukung hingga 50 tugas impor bersamaan dengan ukuran total 1 TB. Setiap pekerjaan impor dapat mengambil hingga 50.000 objek Amazon S3 di semua wilayah. Kuota default ini berlaku untuk setiap akun. Jika Anda merasa perlu merevisi kuota ini, silakan hubungi tim akun Anda, dan ini akan dipertimbangkan atas dasar case-by-case. Untuk detail selengkapnya tentang batasan DynamoDB, lihat [Kuota Layanan](#).

Kesalahan validasi

Selama proses impor, DynamoDB mungkin mengalami kesalahan saat mengurai data Anda. Untuk setiap kesalahan, DynamoDB memancarkan CloudWatch log dan menyimpan hitungan jumlah total kesalahan yang ditemui. Jika objek Amazon S3 itu sendiri salah format atau jika isinya tidak dapat membentuk item DynamoDB, kami dapat melewati pemrosesan bagian objek yang tersisa.

Note

Jika sumber data Amazon S3 memiliki beberapa item yang berbagi kunci yang sama, item tersebut akan ditimpa hingga tersisa satu item. Hal ini dapat terlihat seolah-olah 1 item diimpor dan item lainnya diabaikan. Item duplikat akan ditimpa secara acak, tidak dihitung sebagai kesalahan, dan tidak dipancarkan ke log. CloudWatch

Setelah impor selesai, Anda dapat melihat jumlah total item yang diimpor, jumlah total kesalahan, dan jumlah total item yang diproses. Untuk pemecahan masalah selengkapnya, Anda juga dapat memeriksa ukuran total item yang diimpor dan ukuran total data yang diproses.

Ada tiga kategori kesalahan impor: kesalahan validasi API, kesalahan validasi data, dan kesalahan konfigurasi.

Kesalahan validasi API

Kesalahan validasi API adalah kesalahan tingkat item dari API sinkronisasi. Penyebab umumnya adalah masalah izin, tidak adanya parameter yang diperlukan, dan kegagalan validasi parameter. Detail tentang alasan kegagalan panggilan API terdapat dalam pengecualian yang diberikan oleh permintaan `ImportTable`.

Kesalahan validasi data

Kesalahan validasi data dapat terjadi pada tingkat item atau tingkat file. Selama impor, item divalidasi berdasarkan aturan DynamoDB sebelum diimpor ke tabel target. Ketika item gagal validasi dan tidak diimpor, tugas impor akan melewati item tersebut dan melanjutkan ke item berikutnya. Di akhir pekerjaan, status impor disetel ke GAGAL dengan `FailureCode`, `ItemValidationError` dan `FailureMessage` "Beberapa item gagal memeriksa validasi dan tidak diimpor. Silakan periksa log CloudWatch kesalahan untuk lebih jelasnya."

Penyebab umum kesalahan validasi data mencakup objek tidak dapat diurai, format objek salah (input menentukan `DYNAMODB_JSON` namun objek tidak ada di `DYNAMODB_JSON`), dan skema tidak cocok dengan kunci tabel sumber yang ditentukan.

Kesalahan konfigurasi

Kesalahan konfigurasi biasanya merupakan kesalahan alur kerja karena validasi izin. Alur kerja Impor memeriksa beberapa izin setelah menerima permintaan. Jika ada masalah saat memanggil dependensi yang diperlukan seperti Amazon S3 CloudWatch atau proses menandai status impor sebagai GAGAL. `failureCode` dan `failureMessage` menunjukkan alasan kegagalan. Jika berlaku, pesan kegagalan juga berisi id permintaan yang dapat Anda gunakan untuk menyelidiki alasan kegagalan CloudTrail.

Kesalahan konfigurasi umum termasuk memiliki URL yang salah untuk bucket Amazon S3, dan tidak memiliki izin untuk mengakses bucket Amazon S3, Log CloudWatch, AWS KMS dan kunci yang digunakan untuk mendekripsi objek Amazon S3. Untuk informasi lebih lanjut lihat [Menggunakan dan kunci data](#).

Memvalidasi objek sumber Amazon S3

Untuk memvalidasi objek S3 sumber, lakukan langkah-langkah berikut.

1. Validasi format data dan jenis kompresi

- Pastikan semua objek Amazon S3 yang cocok dengan awalan yang ditentukan memiliki format yang sama (`DYNAMODB_JSON`, `DYNAMODB_ION`, `CSV`)
- Pastikan semua objek Amazon S3 yang cocok dengan awalan yang ditentukan dikompresi dengan cara yang sama (`GZIP`, `ZSTD`, `NONE`)

Note

Objek Amazon S3 tidak perlu memiliki ekstensi yang sesuai (.csv/.json/.ion/.gz/.zstd dll) karena format input yang ditentukan dalam panggilan diutamakan. ImportTable

2. Validasi bahwa data impor sesuai dengan skema tabel yang diinginkan

- Pastikan setiap item dalam data sumber memiliki kunci primer. Kunci urutan bersifat opsional untuk impor.
- Pastikan jenis atribut yang terkait dengan kunci primer dan kunci urutan apa pun cocok dengan jenis atribut dalam Tabel dan skema GSI, seperti yang ditentukan dalam parameter pembuatan tabel

Pemecahan Masalah**CloudWatch log**

Untuk pekerjaan Impor yang gagal, pesan kesalahan terperinci akan diposting ke CloudWatch log. Untuk mengakses log ini, pertama-tama ambil ImportArn dari output dan deskripsikan impor menggunakan perintah ini:

```
aws dynamodb describe-import --import-arn arn:aws:dynamodb:us-east-1:ACCOUNT:table/
target-table/import/01658528578619-c4d4e311
}
```

Contoh output:

```
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."
}
}
```

Ambil grup log dan id impor dari respons di atas dan gunakan untuk mengambil log kesalahan. ID impor adalah elemen jalur terakhir dari bidang `ImportArn`. Nama grup log adalah `/aws-dynamodb/imports`. Nama log stream kesalahan adalah `import-id/error`. Untuk contoh ini, akan menjadi `01658528578619-c4d4e311/error`.

Tidak ada pk kunci dalam item

Jika objek S3 sumber tidak berisi kunci primer yang disediakan sebagai parameter, impor akan gagal. Misalnya, ketika Anda menentukan kunci primer untuk impor sebagai nama kolom "pk".

```
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"'
```

Kolom “pk” hilang dari objek sumber `import-test.csv` yang memiliki konten berikut:

```
title,artist,year_of_release
The Dark Side of the Moon,Pink Floyd,1973
```

Impor ini akan gagal karena kesalahan validasi item karena kunci primer yang hilang di sumber data.

Contoh log CloudWatch kesalahan:

```
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"
}
```

Log kesalahan ini menunjukkan bahwa “Satu atau lebih nilai parameter tidak valid: Kunci pk dalam item tidak ada”. Karena tugas impor ini gagal, tabel “tabel target” sekarang ada dan kosong karena tidak ada item yang diimpor. Item pertama diproses dan objek gagal Validasi Item.

Untuk memperbaiki masalah ini, pertama-tama hapus “tabel target” jika tidak diperlukan lagi. Kemudian gunakan nama kolom kunci primer yang ada di objek sumber, atau perbarui data sumber menjadi:

```
pk,title,artist,year_of_release
```

```
Albums::Rock::Classic::1973::AlbumId::ALB25,The Dark Side of the Moon,Pink Floyd,1973
```

Tabel target ada

Saat Anda memulai pekerjaan impor dan menerima respons sebagai berikut:

```
An error occurred (ResourceInUseException) when calling the ImportTable operation:  
Table already exists: target-table
```

Untuk memperbaiki kesalahan ini, Anda harus memilih nama tabel yang belum ada dan mencoba mengimpor lagi.

Bucket yang ditentukan tidak ada

Jika bucket sumber tidak ada, impor akan gagal dan mencatat detail pesan kesalahan CloudWatch.

Contoh menjelaskan impor:

```
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: 0bqS1LeIMJpQqHLRX2C5Sy7n+8g6iGPwy7ixg7eEeTuEkg/+chU/JF+RbliWytMlkU1UcuCLTrI=; Proxy: null)"
}
```

`FailureCode` adalah `S3NoSuchBucket`, dengan `FailureMessage` berisi detail seperti id permintaan dan layanan yang memunculkan kesalahan. Karena kesalahan diketahui sebelum data diimpor ke tabel, tabel DynamoDB baru tidak dibuat. Dalam beberapa kasus, ketika kesalahan ini terjadi setelah impor data dimulai, tabel dengan data yang diimpor sebagian akan dipertahankan.

Untuk memperbaiki kesalahan ini, pastikan bucket Amazon S3 sumber ada, lalu mulai ulang proses impor.

Praktik terbaik untuk mengimpor dari Amazon S3 ke DynamoDB

Berikut ini adalah praktik terbaik untuk mengimpor data dari Amazon S3 ke DynamoDB.

Tetap di bawah batas 50.000 objek S3

Setiap pekerjaan impor mendukung maksimum 50.000 objek S3. Jika dataset Anda berisi lebih dari 50.000 objek, pertimbangkan untuk mengkonsolidasikannya menjadi objek yang lebih besar.

Hindari objek S3 yang terlalu besar

Objek S3 diimpor secara paralel. Memiliki banyak objek S3 berukuran sedang memungkinkan eksekusi paralel tanpa overhead yang berlebihan. Untuk item di bawah 1 KB, pertimbangkan untuk menempatkan 4.000.000 item ke dalam setiap objek S3. Jika Anda memiliki ukuran rata-rata item yang lebih besar, tempatkan item yang lebih sedikit secara proporsional ke dalam setiap objek S3.

Mengacak data yang diurutkan

Jika objek S3 menyimpan data dalam urutan yang diurutkan, objek tersebut dapat membuat partisi penerapan panas. Ini adalah situasi di mana satu partisi menerima semua aktivitas, lalu partisi berikutnya setelah itu, dan seterusnya. Data dalam urutan yang diurutkan didefinisikan sebagai item secara berurutan dalam objek S3 yang akan ditulis ke partisi target yang sama selama impor. Salah satu situasi umum ketika data diurutkan adalah file CSV di mana item diurutkan berdasarkan kunci partisi sehingga item berulang berbagi kunci partisi yang sama.

Untuk menghindari partisi penerapan panas, kami sarankan Anda mengacak urutan dalam kasus ini. Hal ini dapat meningkatkan performa dengan menyebarkan operasi tulis. Untuk informasi selengkapnya, lihat [Mendistribusikan aktivitas tulis secara efisien selama pengunggahan data](#).

Kompres data untuk menjaga ukuran total objek S3 di bawah batas Regional

Dalam [proses impor dari S3](#), terdapat batasan jumlah total ukuran data objek S3 yang akan diimpor. Batasannya adalah 15 TB di Wilayah us-east-1, us-west-2, dan eu-west-1, dan 1 TB di semua Wilayah lainnya. Batasnya didasarkan pada ukuran objek S3 mentah.

Kompresi memungkinkan lebih banyak data mentah sesuai dengan batasnya. Jika kompresi saja tidak cukup untuk memenuhi impor dalam batas tersebut, Anda juga dapat menghubungi [AWS Premium Support](#) untuk peningkatan kuota.

Waspada bagaimana ukuran item memengaruhi performa

Jika ukuran item rata-rata Anda sangat kecil (di bawah 200 byte), proses impor mungkin memakan waktu sedikit lebih lama dibandingkan ukuran item yang lebih besar.

Pertimbangkan untuk mengimpor tanpa Indeks Sekunder Global

Durasi tugas impor mungkin bergantung pada keberadaan satu atau beberapa indeks sekunder global (GSI). Jika Anda berencana membuat indeks dengan kunci partisi yang memiliki kardinalitas rendah, Anda mungkin melihat impor yang lebih cepat jika Anda menunda pembuatan indeks hingga tugas impor selesai (daripada memasukkannya ke dalam tugas impor).

Note

Membuat GSI selama impor tidak dikenakan biaya tulis (membuat GSI setelah impor akan dikenakan).

Ekspor data DynamoDB ke Amazon S3: cara kerjanya

Ekspor DynamoDB ke S3 adalah solusi yang dikelola sepenuhnya untuk mengekspor data DynamoDB Anda ke bucket Amazon S3 dalam skala besar. Menggunakan ekspor DynamoDB ke S3, Anda dapat mengekspor data dari tabel Amazon DynamoDB kapan saja dalam jendela pemulihan ([PITR](#)) ke [point-in-time bucket](#) Amazon S3. Anda perlu mengaktifkan PITR di meja Anda untuk menggunakan fungsionalitas ekspor. Fitur ini memungkinkan Anda untuk melakukan analitik dan kueri kompleks pada data Anda menggunakan AWS layanan lain seperti AthenaAWS Glue,, SageMaker Amazon, Amazon EMR, dan. AWS Lake Formation

Ekspor DynamoDB ke S3 memungkinkan Anda untuk mengekspor data lengkap dan tambahan dari tabel DynamoDB Anda. Ekspor tidak mengkonsumsi [unit kapasitas baca \(RCU\)](#) apa pun dan tidak berdampak pada kinerja dan ketersediaan meja. Format file ekspor yang didukung adalah format DynamoDB JSON dan Amazon Ion. Anda juga dapat mengekspor data ke bucket S3 yang dimiliki oleh AWS akun lain dan ke AWS wilayah lain. Data Anda selalu dienkripsi end-to-end.

Ekspor penuh DynamoDB dibebankan berdasarkan ukuran tabel DynamoDB (data tabel dan indeks sekunder lokal) pada titik waktu ekspor dilakukan. Ekspor inkremental DynamoDB dibebankan berdasarkan ukuran data yang diproses dari backup berkelanjutan Anda untuk periode waktu yang diekspor. Biaya tambahan berlaku untuk menyimpan data yang diekspor di Amazon S3 dan PUT untuk permintaan yang dibuat terhadap bucket Amazon S3 Anda. Untuk informasi selengkapnya tentang tagihan ini, lihat [harga Amazon DynamoDB](#) dan harga Amazon [S3](#).

Untuk spesifik tentang kuota layanan, lihat. [Ekspor tabel ke Amazon S3](#)

Topik

- [Meminta ekspor tabel di DynamoDB](#)
- [Format output ekspor tabel DynamoDB](#)

Meminta ekspor tabel di DynamoDB

Ekspor tabel DynamoDB memungkinkan Anda mengekspor data tabel ke bucket Amazon S3, agar dapat melakukan analisis dan kueri kompleks pada data Anda menggunakan AWS layanan seperti Athena, Amazon, Amazon EMR, dan. AWS Glue SageMaker AWS Lake Formation Anda dapat meminta ekspor tabel menggunakan AWS Management Console, AWS CLI, atau API DynamoDB.

DynamoDB mendukung ekspor penuh dan ekspor inkremental:

- Dengan ekspor penuh, Anda dapat mengekspor snapshot lengkap tabel Anda dari titik waktu mana pun dalam jendela point-in-time pemulihan (PITR) ke bucket Amazon S3 Anda.
- Dengan ekspor tambahan, Anda dapat mengekspor data dari tabel DynamoDB yang diubah, diperbarui, atau dihapus antara periode waktu tertentu, dalam jendela PITR, ke bucket Amazon S3 Anda.

Topik

- [Prasyarat](#)
- [Meminta ekspor menggunakan AWS Management Console](#)
- [Mendapatkan perincian tentang ekspor masa lalu di AWS Management Console](#)
- [Meminta ekspor menggunakan AWS CLI](#)
- [Mendapatkan perincian tentang ekspor masa lalu di AWS CLI](#)
- [Meminta ekspor menggunakan SDK AWS](#)
- [Mendapatkan perincian tentang ekspor sebelumnya menggunakan SDK AWS](#)

Prasyarat

Aktifkan PITR

Untuk menggunakan fitur ekspor ke S3, Anda harus mengaktifkan PITR di meja Anda. Untuk detail tentang cara mengaktifkan PITR, lihat [oint-in-time Pemulihan P](#). Jika Anda meminta ekspor untuk tabel yang tidak mengaktifkan PITR, permintaan Anda akan gagal dengan pesan pengecualian: “Terjadi kesalahan (PointInTimeRecoveryUnavailableException) saat memanggil ExportTableToPointInTime operasi: Pemulihan titik waktu tidak diaktifkan untuk tabel 'my-dynamodb-table”.

Siapkan izin S3

Anda dapat mengekspor data tabel Anda ke setiap bucket Amazon S3 yang izinnya Anda miliki untuk menulis. Bucket tujuan tidak harus berada di AWS Wilayah yang sama atau memiliki pemilik yang sama dengan pemilik tabel sumber. Kebijakan AWS Identity and Access Management (IAM) Anda harus memungkinkan Anda untuk dapat melakukan tindakan S3 (s3:AbortMultipartUpload,s3:PutObject, dan s3:PutObjectAcl) dan tindakan ekspor DynamoDB (dynamodb:ExportTableToPointInTime). Berikut adalah contoh kebijakan sampel yang akan memberikan izin kepada pengguna Anda untuk melakukan ekspor ke bucket S3.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDynamoDBExportAction",
      "Effect": "Allow",
      "Action": "dynamodb:ExportTableToPointInTime",
      "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table"
    },
    {
      "Sid": "AllowWriteToDestinationBucket",
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::your-bucket/*"
    }
  ]
}
```

Jika Anda perlu menulis ke bucket S3 yang ada di akun lain atau Anda tidak memiliki izin untuk menulis, pemilik bucket S3 perlu menambahkan kebijakan bucket untuk memungkinkan Anda mengekspor dari DynamoDB ke bucket tersebut. Berikut ini adalah contoh kebijakan pada bucket S3 target.

```
{
  "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:::awsexamplebucket1/*"
  }
]
```

Mencabut izin ini saat ekspor sedang berlangsung akan menghasilkan file parsial.

Note

Jika tabel atau bucket yang Anda ekspor dienkripsi dengan kunci terkelola pelanggan, kebijakan kunci KMS tersebut harus memberikan izin DynamoDB untuk menggunakannya. Izin ini diberikan melalui Pengguna/Peran IAM yang memicu pekerjaan ekspor. Untuk informasi selengkapnya tentang enkripsi termasuk praktik terbaik, lihat [Cara DynamoDB AWS KMS menggunakan dan Menggunakan kunci KMS kustom](#).

Meminta ekspor menggunakan AWS Management Console

Contoh berikut menunjukkan cara menggunakan konsol DynamoDB untuk mengekspor tabel yang ada bernama MusicCollection.

Note

Prosedur ini mengasumsikan bahwa Anda telah mengaktifkan point-in-time pemulihan. Untuk mengaktifkannya untuk MusicCollection tabel, pada tab Ikhtisar tabel, di bagian Rincian tabel, pilih Aktifkan untuk oint-in-timepemulihan P.

Untuk meminta ekspor tabel

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/>.

2. Di panel navigasi di sisi kiri konsol, pilih Ekspor ke S3.
3. Pilih tombol Ekspor ke S3.
4. Pilih tabel sumber dan bucket S3 tujuan. Jika bucket tujuan dimiliki oleh akun Anda, Anda dapat menggunakan tombol Jelajahi S3 untuk menemukannya. Jika tidak, masukkan URL bucket menggunakan folder **prefix** is an opsional untuk membantu menjaga bucket tujuan Anda tetap teratur. `s3://bucketname/prefix` format.
5. Pilih Ekspor penuh atau Ekspor tambahan. Ekspor penuh menghasilkan snapshot tabel lengkap dari tabel Anda seperti pada titik waktu yang Anda tentukan. Ekspor inkremental menghasilkan perubahan yang dibuat pada tabel Anda selama periode ekspor yang ditentukan. Output Anda dipadatkan sedemikian rupa sehingga hanya akan berisi status akhir item dari periode ekspor. Item hanya akan muncul sekali dalam ekspor meskipun memiliki beberapa pembaruan dalam periode ekspor yang sama.

Full export

1. Pilih titik waktu Anda ingin mengekspor snapshot tabel lengkap dari. Ini dapat berupa titik waktu mana pun dalam jendela PITR. Atau, Anda dapat memilih Waktu saat ini untuk mengekspor snapshot terbaru.

Export settings

Full export
Export the table data in its current state, or from any specific point up to 35 days ago.

Incremental export
Export any table data that's changed within a specific time period.

Export from a specific point in time [Info](#)

Current time

Export from an earlier point in time
Your earliest export point is the same as the earliest restore point for your table.

(UTC+01:00)

For date, use YYYY/MM/DD format. For time, use 24-hour format.

2. Untuk format file yang diekspor, pilih antara DynamoDB JSON dan Amazon Ion. Secara default, data tabel akan diekspor dalam format DynamoDB JSON dari waktu pemulihan tertentu dalam jendela pemulihan titik waktu dan dienkripsi menggunakan kunci Amazon S3 (SSE-S3). Anda dapat mengubah pengaturan ini:

Note

Jika Anda memilih untuk mengenkripsi ekspor menggunakan kunci yang dilindungi oleh AWS Key Management Service (AWS KMS), kunci harus berada di Region yang sama dengan bucket S3 tujuan.

Exported file format | **Info**

DynamoDB JSON

Amazon Ion

Open-source text format, which is a superset of JSON.

Incremental export

1. Pilih periode Ekspor yang ingin Anda ekspor data tambahan. Pilih waktu mulai di dalam jendela PITR. Durasi periode ekspor harus minimal 15 menit dan tidak lebih dari 24 jam. Waktu mulai periode ekspor adalah inklusif dan waktu akhir eksklusif.

Export settings

Full export


Export the table data in its current state, or from any specific point up to 35 days ago.

Incremental export

Export any table data that's changed within a specific time period.

Export period

Specify when the incremental export starts and ends. Your earliest export point is the same as the earliest restore point for your table.

 2023-09-01T12:00:00+01:00 — 2023-09-02T12:00:00+01:00

2. Pilih antara mode Absolute atau mode Relatif.
 - a. Mode absolut akan mengekspor data tambahan untuk periode waktu yang Anda tentukan.

Relative mode
Absolute mode

<
August 2023
September 2023
>

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
	1	2	3	4	5	6					1	2	3
7	8	9	10	11	12	13	4	5	6	7	8	9	10
14	15	16	17	18	19	20	11	12	13	14	15	16	17
21	22	23	24	25	26	27	18	19	20	21	22	23	24
28	29	30	31				25	26	27	28	29	30	

Start date

Start time

End date

End time

The export period must be between 15 minutes and 24 hours. For date, use YYYY/MM/DD. For time, use 24 hr format.

Clear
Cancel
Apply

- b. Mode relatif akan mengekspor data tambahan untuk periode ekspor yang relatif terhadap waktu pengiriman pekerjaan ekspor Anda.

Relative mode | Absolute mode

Choose a range

- Last 1 hour
- Last 6 hours
- Last 12 hours
- Last 24 hours
- Custom range

Set a custom range in the past

Clear

Cancel

Apply

3. Untuk format file yang diekspor, pilih antara DynamoDB JSON dan Amazon Ion. Secara default, data tabel akan diekspor dalam format DynamoDB JSON dari waktu pemulihan tertentu dalam jendela pemulihan titik waktu dan dienkripsi menggunakan kunci Amazon S3 (SSE-S3). Anda dapat mengubah pengaturan ini:

Note

Jika Anda memilih untuk mengenkripsi ekspor menggunakan kunci yang dilindungi oleh AWS Key Management Service (AWS KMS), kunci harus berada di Region yang sama dengan bucket S3 tujuan.

Exported file format | **Info**

DynamoDB JSON

Amazon Ion

Open-source text format, which is a superset of JSON.

4. Untuk jenis tampilan Ekspor, pilih Gambar baru dan lama atau Gambar baru saja. Gambar baru memberikan status item terbaru. Gambar lama memberikan status item tepat sebelum “tanggal dan waktu mulai” yang ditentukan. Pengaturan default adalah Gambar baru dan lama. Untuk informasi lebih lanjut tentang gambar baru dan gambar lama, lihat [Output ekspor inkremental](#).

Export view type

New and old images

New images only

6. Klik tombol Ekspor untuk memulai ekspor.

Data yang diekspor tidak konsisten secara transaksional. Ini berarti bahwa operasi transaksi Anda dapat terpecah antara dua output ekspor. Anda mungkin melihat subset item yang dimodifikasi oleh operasi transaksi yang tercermin dalam ekspor, sementara subset modifikasi lain dalam transaksi yang sama tidak tercermin dalam permintaan ekspor yang sama. Ekspor bersifat konsisten pada akhirnya. Jika transaksi robek selama ekspor, Anda dijamin memiliki sisa transaksi dalam ekspor berikutnya yang berdekatan, tanpa duplikat. Periode waktu yang digunakan untuk ekspor didasarkan pada jam sistem internal dan dapat bervariasi satu menit dari jam lokal aplikasi Anda.

Mendapatkan perincian tentang ekspor masa lalu di AWS Management Console

Anda dapat menemukan informasi tentang tugas ekspor yang telah dijalankan sebelumnya dengan mengklik bagian Ekspor ke S3 di bar samping navigasi. Bagian ini berisi daftar semua ekspor yang Anda buat dalam 90 hari terakhir. Memilih ARN tugas yang tercantum dalam tab Ekspor akan mengambil informasi tentang ekspor tersebut, termasuk pengaturan konfigurasi lanjutan yang Anda pilih. Perhatikan bahwa meskipun metadata tugas ekspor berakhir setelah 90 hari dan pekerjaan yang lebih lama dari itu tidak lagi ditemukan dalam daftar ini, objek di bucket S3 Anda tetap ada selama kebijakan bucket memungkinkan. DynamoDB tidak pernah menghapus objek apa pun yang dibuatnya dalam bucket S3 Anda selama ekspor.

Meminta ekspor menggunakan AWS CLI

Contoh berikut menunjukkan cara menggunakan AWS CLI untuk mengekspor tabel yang ada bernama MusicCollection ke bucket S3 yang disebut ddb-export-musiccollection.

Note

Prosedur ini mengasumsikan bahwa Anda telah mengaktifkan point-in-time pemulihan. Untuk mengaktifkannya untuk tabel `MusicCollection`, jalankan perintah berikut.

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

Full export

Perintah berikut mengekspor `MusicCollection` ke bucket S3 yang disebut `ddb-export-musiccollection-9012345678` dengan awalan `2020-Nov`. Data tabel akan diekspor dalam format DynamoDB JSON dari waktu tertentu dalam jendela pemulihan titik waktu dan dienkripsi menggunakan kunci Amazon S3 (SSE-S3).

Note

Jika meminta ekspor tabel lintas akun, pastikan untuk menyertakan opsi. `--s3-bucket-owner`

```
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

Perintah berikut melakukan ekspor inkremental dengan menyediakan yang baru `--export-type` dan `--incremental-export-specification`. Gantikan nilai Anda sendiri untuk nilai apa pun yang dicetak miring. Waktu ditentukan sebagai detik sejak zaman.

```
aws dynamodb export-table-to-point-in-time \  
  --table-arn arn:aws:dynamodb:REGION:ACCOUNT:table/TABLENAME \  
  --export-type INCREMENTAL_EXPORT \  
  --incremental-export-specification INCREMENTAL_EXPORT_SPECIFICATION
```

```
--s3-bucket BUCKET --s3-prefix PREFIX \  
--incremental-export-specification  
ExportFromTime=1693569600,ExportToTime=1693656000,ExportViewType=NEW_AND_OLD_IMAGES  
\  
--export-type INCREMENTAL_EXPORT
```

Note

Jika Anda memilih untuk mengenkripsi ekspor menggunakan kunci yang dilindungi oleh AWS Key Management Service (AWS KMS), kunci harus berada di Region yang sama dengan bucket S3 tujuan.

Mendapatkan perincian tentang ekspor masa lalu di AWS CLI

Anda dapat menemukan informasi tentang permintaan ekspor yang telah dijalankan sebelumnya menggunakan `list-exports` Perintah. Perintah ini menampilkan daftar semua ekspor yang telah Anda buat dalam 90 hari terakhir. Perhatikan bahwa meskipun metadata tugas ekspor berakhir setelah 90 hari dan pekerjaan yang lebih lama dari itu yang tidak lagi dikembalikan oleh perintah `list-exports`, objek di bucket S3 Anda tetap ada selama kebijakan bucket memungkinkan. DynamoDB tidak pernah menghapus objek apa pun yang dibuatnya dalam bucket S3 Anda selama ekspor.

Ekspor memiliki status PENDING sampai berhasil atau gagal. Jika mereka berhasil, statusnya akan berubah menjadi COMPLETED. Jika gagal, statusnya akan berubah menjadi FAILED dengan a `failure_message` dan `failure_reason`.

Pada contoh berikut, kita menggunakan parameter `table-arn` opsional untuk hanya mencantumkan ekspor dari tabel tertentu.

```
aws dynamodb list-exports \  
--table-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog
```

Untuk mengambil informasi terperinci tentang tugas ekspor tertentu, termasuk pengaturan konfigurasi lanjutan, gunakan perintah `describe-export`.

```
aws dynamodb describe-export \  
--export-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/  
export/01234567890123-a1b2c3d4
```

Meminta ekspor menggunakan SDK AWS

Gunakan cuplikan kode ini untuk meminta ekspor tabel menggunakan AWS SDK pilihan Anda.

Python

Ekspor penuh

```
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'
)
```

Ekspor tambahan

```
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'  
)
```

Mendapatkan perincian tentang ekspor sebelumnya menggunakan SDK AWS

Gunakan cuplikan kode ini untuk mendapatkan detail tentang ekspor tabel sebelumnya menggunakan AWS SDK pilihan Anda.

Python

Ekspor penuh

```
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',  
    )  
)
```

Ekspor tambahan

```
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']  
)
```

Format output ekspor tabel DynamoDB

Sebuah ekspor tabel DynamoDB mencakup file manifes selain file yang berisi data tabel Anda. File ini semua disimpan di bucket Amazon S3 yang Anda tentukan di [permintaan ekspor](#). Bagian berikut menjelaskan format dan isi dari setiap objek output.

Output ekspor penuh

File manifes

DynamoDB membuat dua file manifes, bersama dengan file checksum, dalam bucket S3 yang ditentukan untuk setiap permintaan ekspor.

```
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
```

Anda memilih **export-prefix** saat meminta ekspor tabel. Ini membantu Anda menjaga kerapian file di bucket S3 tujuan. **ExportId** adalah token unik yang dihasilkan oleh layanan untuk memastikan bahwa beberapa ekspor ke bucket S3 dan `export-prefix` ekspor yang sama tidak saling menimpa.

Akskor akan membuat setidaknya 1 file per partisi. Untuk partisi yang kosong, permintaan ekspor Anda akan membuat file kosong. Semua item di setiap file berasal dari keypace hash partisi tertentu.

Note

DynamoDB juga membuat sebuah file kosong bernama `_started` dalam direktori yang sama sebagai file manifes. File ini memverifikasi bahwa bucket tujuan dapat ditulis dan bahwa ekspor telah dimulai. File ini bisa dihapus dengan aman.

Manifes ringkasan

File `manifest-summary.json` berisi informasi ringkasan tentang pekerjaan ekspor. Ini memungkinkan Anda mengetahui file data mana di folder data bersama yang terkait dengan ekspor ini. Formatnya adalah sebagai berikut:


```
{
  "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": "AWS DynamoDB/01693685827463-2d8752fd/manifest-files.json",
  "billedSizeBytes": 0,
  "itemCount": 8,
  "outputFormat": "DYNAMODB_JSON",
  "exportType": "FULL_EXPORT"
}
```

Manifes file

File `manifest-files.json` berisi informasi tentang file yang berisi data tabel Anda yang diekspor. File ada dalam format [baris JSON](#), jadi baris baru digunakan sebagai pembatas item. Pada contoh berikut, perincian satu file data dari manifes file diformat pada beberapa baris untuk keterbacaan.

```
{
  "itemCount": 8,
  "md5Checksum": "sQMSPeILNgoQmarvDFonGQ==",
  "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
  "dataFileS3Key": "AWS DynamoDB/01693685827463-2d8752fd/data/asdl123dasas.json.gz"
}
```

File data

DynamoDB dapat mengekspor data tabel Anda dalam dua format: DynamoDB JSON dan Amazon Ion. Terlepas dari format yang Anda pilih, data Anda akan ditulis ke beberapa file terkompresi yang dinamai berdasarkan kunci. File-file ini juga tercantum dalam `manifest-files.json` file.

Struktur direktori bucket S3 Anda setelah ekspor penuh akan berisi semua file manifes dan file data Anda di bawah folder `export` Id.

```
DestinationBucket/DestinationPrefix
.
### AWS DynamoDB
### 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

Sebuah ekspor tabel dalam format DynamoDB JSON terdiri dari beberapa objek Item. Setiap objek individual berada dalam format JSON marshalled standar DynamoDB.

Saat membuat parser kustom untuk data ekspor DynamoDB JSON, formatnya adalah [garis JSON](#). Ini berarti bahwa baris baru digunakan sebagai pembatas item. Banyak AWS layanan, seperti Athena dan AWS Glue, akan mengurai format ini secara otomatis.

Dalam contoh berikut, satu item dari ekspor JSON DynamoDB telah diformat pada beberapa baris demi keterbacaan.

```
{
  "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 Ion](#) adalah format serialisasi data hierarkis yang kaya jenis, dapat dijelaskan sendiri, yang dibuat untuk mengatasi tantangan perkembangan pesat, pemisahan, dan efisiensi yang dihadapi setiap hari saat merekayasa arsitektur berorientasi layanan berskala besar. DynamoDB mendukung ekspor data tabel dalam [format teks](#) Ion, yang merupakan superset dari JSON.

Saat Anda mengekspor tabel ke format Ion, tipe data DynamoDB yang digunakan dalam tabel dipetakan ke [tipe data Ion](#). Set DynamoDB menggunakan [anotasi tipe Ion](#) untuk memperjelas tipe data yang digunakan dalam tabel sumber.

Konversi tipe data DynamoDB ke Ion

Tipe data DynamoDB	Perwakilan Ion
String (S)	string
Boolean (BOOL)	bool
Angka (N)	desimal
Biner (B)	blob
Set (SS, NS, BS)	daftar (dengan anotasi tipe \$dynamodb_SS, \$dynamodb_NS, or \$dynamodb_BS)

Tipe data DynamoDB	Perwakilan Ion
Daftar	daftar
Peta	struct

Item dalam ekspor Ion dibatasi oleh baris baru. Setiap baris dimulai dengan penanda versi Ion, diikuti dengan item dalam format Ion. Dalam contoh berikut, item dari ekspor Ion telah diformat dalam beberapa baris demi keterbacaan.

```
$ion_1_0 {
  Item:{
    Authors:$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"
  }
}
```

Output ekspor inkremental

File manifes

DynamoDB membuat dua file manifes, bersama dengan file checksum, dalam bucket S3 yang ditentukan untuk setiap permintaan ekspor.

```
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
```

Anda memilih **export-prefix** saat meminta ekspor tabel. Ini membantu Anda menjaga kerapian file di bucket S3 tujuan. **ExportId** adalah token unik yang dihasilkan oleh layanan untuk memastikan bahwa beberapa ekspor ke bucket S3 dan **export-prefix** ekspor yang sama tidak saling menimpa.

Akspor akan membuat setidaknya 1 file per partisi. Untuk partisi yang kosong, permintaan ekspor Anda akan membuat file kosong. Semua item di setiap file berasal dari keyspace hash partisi tertentu.

Note

DynamoDB juga membuat sebuah file kosong bernama `_started` dalam direktori yang sama sebagai file manifes. File ini memverifikasi bahwa bucket tujuan dapat ditulis dan bahwa ekspor telah dimulai. File ini bisa dihapus dengan aman.

Manifes ringkasan

File `manifest-summary.json` berisi informasi ringkasan tentang pekerjaan ekspor. Ini memungkinkan Anda mengetahui file data mana di folder data bersama yang terkait dengan ekspor ini. Formatnya adalah sebagai berikut:

```
{
  "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"
}
```

Manifes file

File `manifest-files.json` berisi informasi tentang file yang berisi data tabel Anda yang diekspor. File ada dalam format [baris JSON](#), jadi baris baru digunakan sebagai pembatas item. Pada contoh berikut, perincian satu file data dari manifes file diformat pada beberapa baris untuk keterbacaan.

```
{
  "itemCount": 8,
  "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
  "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
  "dataFileS3Key": "AWSDynamoDB/data/sgad6417s6vss4p7owp0471bcq.json.gz"
}
```

File data

DynamoDB dapat mengekspor data tabel Anda dalam dua format: DynamoDB JSON dan Amazon Ion. Terlepas dari format yang Anda pilih, data Anda akan ditulis ke beberapa file terkompresi yang dinamai berdasarkan kunci. File-file ini juga tercantum dalam `manifest-files.json` file.

File data untuk ekspor inkremental semuanya ada di folder data umum di bucket S3 Anda. File manifes Anda berada di bawah folder ID ekspor Anda.

```
DestinationBucket/DestinationPrefix
.
### AWS DynamoDB
### 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
# ...
```

Saat Anda mengekspor file, output setiap item menyertakan stempel waktu yang menunjukkan kapan item tersebut diperbarui di tabel Anda dan struktur data yang menunjukkan apakah itu adalah operasi `insert`, `update`, atau `delete`. Stempel waktu didasarkan pada jam sistem internal dan dapat bervariasi dari jam aplikasi Anda. Untuk ekspor tambahan, Anda dapat memilih antara dua jenis tampilan ekspor untuk struktur keluaran Anda: gambar baru dan lama atau hanya gambar baru.

- Gambar baru memberikan status terkini item
- Gambar lama menunjukkan status item tepat sebelum tanggal dan waktu mulai yang ditentukan

Jenis tampilan dapat membantu jika Anda ingin melihat bagaimana item diubah dalam periode ekspor. Ini juga dapat berguna untuk memperbarui sistem hilir Anda secara efisien, terutama jika sistem hilir tersebut memiliki kunci partisi yang tidak sama dengan kunci partisi DynamoDB Anda.

Anda dapat menyimpulkan apakah item dalam output ekspor tambahan Anda adalah `insert`, `update`, atau `delete` dengan melihat struktur output. Struktur ekspor inkremental dan operasi yang sesuai dirangkum dalam tabel di bawah ini untuk kedua jenis tampilan ekspor.

Operasi	Hanya gambar baru	Gambar baru dan lama
Sisipkan	Tombol + gambar baru	Tombol + gambar baru
Perbarui	Tombol + gambar baru	Kunci + gambar baru + gambar lama
Hapus	Kunci	Tombol + gambar lama
Sisipkan + hapus	Tidak ada output	Tidak ada output

DynamoDB Json

Ekspor tabel dalam format JSON DynamoDB terdiri dari stempel waktu metadata yang menunjukkan waktu penulisan item, diikuti dengan kunci item dan nilainya. Berikut ini menunjukkan contoh output JSON DynamoDB menggunakan tipe tampilan ekspor sebagai gambar Baru dan Lama.

```
// 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"
},
"Key": {
  "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"
  },
  "Key": {
    "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"
  },
  "Key": {
    "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 Ion](#) adalah format serialisasi data hierarkis yang kaya jenis, dapat dijelaskan sendiri, yang dibuat untuk mengatasi tantangan perkembangan pesat, pemisahan, dan efisiensi yang dihadapi setiap hari saat merekayasa arsitektur berorientasi layanan berskala besar. DynamoDB mendukung ekspor data tabel dalam [format teks Ion](#), yang merupakan superset dari JSON.

Saat Anda mengekspor tabel ke format Ion, tipe data DynamoDB yang digunakan dalam tabel dipetakan ke [tipe data Ion](#). Set DynamoDB menggunakan [anotasi tipe Ion](#) untuk memperjelas tipe data yang digunakan dalam tabel sumber.

Konversi tipe data DynamoDB ke Ion

Tipe data DynamoDB	Perwakilan Ion
String (S)	string
Boolean (BOOL)	bool
Angka (N)	desimal
Biner (B)	blob
Set (SS, NS, BS)	daftar (dengan anotasi tipe \$dynamodb_SS, \$dynamodb_NS, or \$dynamodb_BS)
Daftar	daftar
Peta	struct

Item dalam ekspor Ion dibatasi oleh baris baru. Setiap baris dimulai dengan penanda versi Ion, diikuti dengan item dalam format Ion. Dalam contoh berikut, item dari ekspor Ion telah diformat dalam beberapa baris demi keterbacaan.

```
$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"
    }
  }
}
```

Integrasi DynamoDB Zero-ETL dengan Amazon Service OpenSearch

Amazon DynamoDB menawarkan integrasi nol-ETL dengan OpenSearch Amazon Service melalui plugin DynamoDB untuk Ingestion. OpenSearch Amazon OpenSearch Ingestion menawarkan pengalaman tanpa kode yang dikelola sepenuhnya untuk menyerap data ke Layanan Amazon OpenSearch

Dengan plugin DynamoDB OpenSearch untuk Ingestion, Anda dapat menggunakan satu atau lebih tabel DynamoDB sebagai sumber untuk menelan satu atau lebih indeks Layanan. OpenSearch Anda dapat menelusuri dan mengonfigurasi pipeline OpenSearch Ingestion Anda dengan DynamoDB sebagai sumber dari Integrasi OpenSearch Ingestion atau DynamoDB di AWS Management Console

- Mulailah dengan OpenSearch Ingestion dengan mengikuti panduan Memulai [OpenSearch Ingestion](#).

- [Pelajari tentang prasyarat dan semua opsi konfigurasi untuk plugin DynamoDB di plugin DynamoDB untuk dokumentasi Ingestion. OpenSearch](#)

Cara kerjanya

Plugin menggunakan ekspor [DynamoDB ke Amazon S3](#) untuk membuat snapshot awal untuk dimuat. OpenSearch Setelah snapshot dimuat, plugin menggunakan DynamoDB Streams untuk mereplikasi perubahan lebih lanjut dalam waktu dekat. Setiap item diproses sebagai acara di OpenSearch Ingestion dan dapat dimodifikasi dengan plugin prosesor. Anda dapat menghapus atribut atau membuat atribut gabungan dan mengirimkannya ke indeks berbeda melalui rute.

Anda harus mengaktifkan [point-in-time pemulihan \(PITR\)](#) untuk menggunakan ekspor ke Amazon S3. Anda juga harus mengaktifkan [DynamoDB Streams](#) (dengan opsi gambar baru & lama dipilih) untuk dapat menggunakannya. Dimungkinkan untuk membuat alur tanpa mengambil snapshot dengan mengecualikan pengaturan ekspor.

Anda juga dapat membuat alur hanya dengan snapshot dan tanpa pembaruan dengan mengecualikan pengaturan aliran. Plugin tidak menggunakan throughput baca atau tulis di meja Anda, sehingga aman digunakan tanpa memengaruhi lalu lintas produksi Anda. Ada batasan jumlah konsumen paralel pada aliran yang harus Anda pertimbangkan sebelum membuat integrasi ini atau lainnya. Untuk pertimbangan lain, lihat [the section called “Praktik terbaik integrasi”](#).

Untuk pipeline sederhana, satu OpenSearch Compute Unit (OCU) dapat memproses sekitar 1 MB per detik penulisan. Hal ini setara dengan sekitar 1000 unit permintaan tulis (WCU). Bergantung pada kompleksitas alur Anda dan faktor lainnya, Anda mungkin mencapai lebih atau kurang dari angka ini.

OpenSearch Ingestion mendukung antrian huruf mati (DLQ) untuk peristiwa yang menyebabkan kesalahan yang tidak dapat dipulihkan. Selain itu, pipeline dapat melanjutkan dari tempat yang ditinggalkannya tanpa campur tangan pengguna meskipun ada gangguan layanan dengan DynamoDB, pipeline, atau Amazon Service. OpenSearch

Jika gangguan berlangsung lebih dari 24 jam, hal ini dapat menyebabkan hilangnya pembaruan. Namun, alur akan terus memproses pembaruan yang masih tersedia saat ketersediaan dipulihkan. Anda perlu melakukan pembuatan indeks baru untuk memperbaiki penyimpangan apa pun karena peristiwa yang dijatuhkan kecuali mereka berada dalam antrean surat mati.

Untuk semua pengaturan dan detail untuk plugin, lihat Dokumentasi plugin [OpenSearchDynamoDB Ingestion](#).

Terintegrasi menciptakan pengalaman melalui konsol

DynamoDB OpenSearch dan Layanan memiliki pengalaman terintegrasi dalam AWS Management Console, yang merampingkan proses memulai. Ketika Anda melalui langkah-langkah ini, layanan akan secara otomatis memilih cetak biru DynamoDB dan menambahkan informasi DynamoDB yang sesuai untuk Anda.

Untuk membuat integrasi, ikuti panduan [Memulai OpenSearch Ingestion](#). Saat Anda sampai ke [Langkah 3: Buat alur](#), ganti Langkah 1 dan 2 dengan langkah-langkah berikut:

1. Navigasikan ke konsol DynamoDB.
2. Di panel navigasi sebelah kiri, pilih Integrasi.
3. Pilih tabel DynamoDB yang ingin Anda tiru. OpenSearch
4. Pilih Buat.

Dari sini, Anda dapat melanjutkan dengan sisa tutorial.

Langkah selanjutnya

Untuk pemahaman yang lebih baik tentang bagaimana DynamoDB terintegrasi OpenSearch dengan Layanan, lihat berikut ini:

- [Memulai dengan Amazon OpenSearch Ingestion](#)
- [Konfigurasi dan persyaratan plugin DynamoDB](#)

Menangani perubahan yang menyebabkan gangguan pada indeks Anda

OpenSearch dapat secara dinamis menambahkan atribut baru ke indeks Anda. Namun, setelah template pemetaan Anda disetel untuk kunci tertentu, Anda harus mengambil tindakan tambahan untuk mengubahnya. Selain itu, jika perubahan Anda mengharuskan untuk memproses ulang semua data dalam tabel DynamoDB Anda, Anda harus mengambil langkah-langkah untuk memulai ekspor baru.

Note

Dalam semua opsi ini, Anda mungkin masih mengalami masalah jika tabel DynamoDB Anda memiliki konflik tipe dengan template pemetaan yang Anda tentukan. Pastikan Anda

mengaktifkan antrean surat mati (DLQ) (bahkan dalam pengembangan). Ini membuatnya lebih mudah untuk memahami apa yang mungkin salah dengan catatan yang menyebabkan konflik ketika sedang diindeks ke indeks Anda. OpenSearch

Topik

- [Cara kerjanya](#)
- [Hapus indeks Anda dan setel ulang alur \(opsi alur-sentris\)](#)
- [Buat ulang indeks Anda dan setel ulang alur \(opsi indeks-sentris\)](#)
- [Buat indeks dan sink baru \(opsi online\)](#)
- [Praktik terbaik untuk menghindari dan men-debug konflik tipe](#)

Cara kerjanya

Berikut gambaran umum singkat dari tindakan yang dilakukan saat menangani perubahan yang menyebabkan gangguan pada indeks Anda. Lihat step-by-step prosedur di bagian berikut.

- Hentikan dan mulai alur: Opsi ini mengatur ulang status alur, dan alur akan dimulai ulang dengan ekspor penuh baru. Ini bersifat non-destruktif, sehingga tidak menghapus indeks Anda atau data apa pun di DynamoDB. Jika Anda tidak membuat indeks baru sebelum melakukan ini, Anda mungkin melihat banyak kesalahan dari konflik versi karena ekspor mencoba memasukkan dokumen yang lebih lama dari `_version` saat ini ke dalam indeks. Anda dapat mengabaikan kesalahan ini dengan aman. Anda tidak akan ditagih untuk alur saat alur tersebut dihentikan.
- Perbarui alur: Opsi ini memperbarui konfigurasi dalam alur dengan pendekatan [biru/hijau](#), tanpa kehilangan status apa pun. Jika Anda membuat perubahan signifikan pada alur Anda (seperti menambahkan rute, indeks, atau kunci baru ke indeks yang sudah ada), Anda mungkin perlu melakukan reset penuh pada alur dan membuat ulang indeks Anda. Opsi ini tidak melakukan ekspor penuh.
- Hapus dan buat ulang indeks: Opsi ini menghapus data dan pengaturan pemetaan pada indeks Anda. Anda harus melakukan ini sebelum membuat perubahan yang menyebabkan gangguan pada pemetaan Anda. Ini akan merusak aplikasi apa pun yang mengandalkan indeks hingga indeks dibuat ulang dan disinkronkan. Menghapus indeks tidak memulai ekspor baru. Anda harus menghapus indeks Anda hanya setelah Anda memperbarui alur Anda. Jika tidak, indeks Anda mungkin dibuat ulang sebelum Anda memperbarui pengaturan.

Hapus indeks Anda dan setel ulang alur (opsi alur-sentris)

Metode ini sering menjadi pilihan tercepat jika Anda masih dalam pengembangan. Anda akan menghapus indeks Anda di OpenSearch Layanan, lalu [menghentikan dan memulai](#) pipeline Anda untuk memulai ekspor baru semua data Anda. Hal ini memastikan bahwa tidak ada konflik template pemetaan dengan indeks yang ada, dan tidak ada kehilangan data dari tabel yang diproses tidak lengkap.

1. Hentikan pipeline baik melalui AWS Management Console, atau dengan menggunakan operasi `StopPipelineAPI` dengan AWS CLI atau SDK.
2. [Perbarui konfigurasi alur Anda](#) dengan perubahan baru.
3. Hapus indeks Anda di OpenSearch Layanan, baik melalui panggilan REST API atau OpenSearch Dasbor Anda.
4. Mulai pipeline melalui konsol, atau dengan menggunakan operasi API `StartPipeline` dengan AWS CLI atau SDK.

Note

Tindakan ini memulai ekspor penuh baru, yang akan menimbulkan biaya tambahan.

5. Pantau masalah yang tidak terduga karena ekspor baru dihasilkan untuk membuat indeks baru.
6. Konfirmasikan bahwa indeks sesuai dengan harapan Anda di OpenSearch Layanan.

Setelah ekspor selesai dan melanjutkan pembacaan dari aliran, data tabel DynamoDB Anda sekarang akan tersedia dalam indeks.


Buat ulang indeks Anda dan setel ulang alur (opsi indeks-sentris)

Metode ini berfungsi dengan baik jika Anda perlu melakukan banyak iterasi pada desain indeks di OpenSearch Service sebelum melanjutkan pipeline dari DynamoDB. Ini dapat berguna untuk pengembangan ketika Anda ingin mengulangi dengan sangat cepat pada pola pencarian Anda, dan ingin menghindari menunggu ekspor baru selesai di antara setiap iterasi.

1. Hentikan pipeline baik melalui AWS Management Console, atau dengan memanggil operasi `StopPipelineAPI` dengan AWS CLI atau SDK.
2. Hapus dan buat ulang indeks Anda OpenSearch dengan template pemetaan yang ingin Anda gunakan. Anda dapat memasukkan beberapa data sampel secara manual untuk mengonfirmasi

bahwa pencarian Anda berfungsi sebagaimana dimaksud. Jika data sampel Anda mungkin bertentangan dengan data apa pun dari DynamoDB, pastikan untuk menghapusnya sebelum melanjutkan ke langkah berikutnya.

3. Jika Anda memiliki template pengindeksan di pipeline Anda, hapus atau ganti dengan yang sudah Anda buat di OpenSearch Service. Pastikan bahwa nama indeks Anda cocok dengan nama dalam alur.
4. Mulai alur baik melalui konsol, atau dengan memanggil operasi API `StartPipeline` dengan AWS CLI atau SDK.

 Note


Tindakan ini akan memulai ekspor penuh baru, yang akan menimbulkan biaya tambahan.

5. Pantau masalah yang tidak terduga karena ekspor baru dihasilkan untuk membuat indeks baru.

Setelah ekspor selesai dan melanjutkan pembacaan dari aliran, data tabel DynamoDB Anda sekarang akan tersedia dalam indeks.

Buat indeks dan sink baru (opsi online)

Metode ini berfungsi dengan baik jika Anda perlu memperbarui template pemetaan Anda tetapi saat ini menggunakan indeks Anda dalam produksi. Ini menciptakan indeks baru, yang mana Anda harus memindahkan aplikasi Anda setelah disinkronkan dan divalidasi.

 Note

Ini akan membuat konsumen lain di aliran. Ini bisa menjadi masalah jika Anda juga memiliki konsumen lain seperti AWS Lambda atau tabel global. Anda mungkin perlu menjeda pembaruan ke alur yang ada untuk membuat kapasitas memuat indeks baru.

1. [Buat alur baru](#) dengan pengaturan baru dan nama indeks yang berbeda.
2. Pantau indeks baru untuk masalah yang tidak terduga.
3. Tukar aplikasi ke indeks baru.
4. Hentikan dan hapus alur lama setelah memvalidasi bahwa semuanya berfungsi dengan benar.

Praktik terbaik untuk menghindari dan men-debug konflik tipe

- Selalu gunakan antrean surat mati (DLQ) untuk memudahkan debug ketika ada konflik tipe.
- Selalu gunakan template indeks dengan pemetaan dan setel `include_keys`. Sementara OpenSearch Layanan secara dinamis memetakan kunci baru, ini dapat menyebabkan masalah dengan perilaku yang tidak terduga (seperti mengharapkan sesuatu menjadi `aGeoPoint`, tetapi dibuat sebagai `string` atau `object`) atau kesalahan (seperti memiliki `number` yang merupakan campuran `long` dan `float` nilai).
- Jika Anda perlu menjaga indeks yang ada tetap berfungsi dalam produksi, Anda juga dapat mengganti salah satu [langkah penghapusan indeks](#) sebelumnya hanya dengan mengganti nama indeks Anda di file konfigurasi alur Anda. Ini menciptakan indeks baru. Aplikasi Anda kemudian perlu diperbarui untuk menunjuk ke indeks baru setelah selesai.
- Jika Anda memiliki masalah konversi tipe yang Anda perbaiki dengan prosesor, Anda dapat mengujinya dengan `UpdatePipeline`. Untuk melakukan ini, Anda harus berhenti dan memulai atau [memproses antrean surat mati Anda](#) untuk memperbaiki dokumen yang sebelumnya dilewati yang memiliki kesalahan.

Praktik terbaik integrasi dengan DynamoDB

Saat mengintegrasikan DynamoDB dengan layanan lain, Anda sebaiknya selalu mengikuti praktik terbaik dalam menggunakan setiap layanan individual. Namun, ada beberapa praktik terbaik untuk integrasi tertentu yang sebaiknya Anda pertimbangkan.

Topik

- [Membuat snapshot di DynamoDB](#)
- [Menangkap perubahan data di DynamoDB](#)
- [Integrasi DynamoDB Zero-ETL dengan Layanan OpenSearch](#)

Membuat snapshot di DynamoDB

- Umumnya, kami merekomendasikan penggunaan [ekspor ke Amazon S3](#) untuk membuat snapshot untuk replikasi awal. Cara ini hemat biaya, dan tidak akan menyaingi lalu lintas aplikasi Anda dalam hal throughput. Anda juga dapat mempertimbangkan cadangan dan memulihkan ke tabel baru yang diikuti dengan operasi pemindaian. Hal ini akan menghindari persaingan untuk throughput dengan aplikasi Anda, tetapi secara umum akan jauh lebih hemat biaya daripada ekspor.

- Selalu atur a `StartTime` saat melakukan ekspor. Hal ini memudahkan untuk menentukan dari mana Anda akan memulai pengambilan data perubahan (CDC).
- Saat menggunakan ekspor ke S3, tetapkan tindakan siklus hidup pada bucket S3. Biasanya, tindakan kedaluwarsa yang diatur pada 7 hari sudah aman, tetapi Anda harus mengikuti panduan apa pun yang mungkin dimiliki perusahaan Anda. Meskipun Anda secara eksplisit menghapus item Anda setelah dikonsumsi, tindakan ini dapat membantu menangkap masalah, yang membantu mengurangi biaya yang tidak perlu dan mencegah pelanggaran kebijakan.

Menangkap perubahan data di DynamoDB

- Jika Anda membutuhkan CDC yang mendekati waktu nyata, gunakan [DynamoDB Streams](#) atau Amazon Kinesis [Data Streams](#) (KDS). Ketika memutuskan layanan mana yang akan Anda gunakan, umumnya pertimbangkan mana yang paling mudah digunakan dengan layanan hilir. Jika Anda perlu menyediakan pemrosesan peristiwa dalam urutan pada tingkat kunci partisi, atau jika Anda memiliki item yang sangat besar, gunakan DynamoDB Streams.
- Jika Anda tidak memerlukan CDC mendekati waktu nyata, Anda dapat menggunakan [ekspor ke Amazon S3 dengan ekspor inkremental](#) untuk hanya mengekspor perubahan yang terjadi antara dua titik waktu.

Jika Anda menggunakan ekspor ke S3 untuk menghasilkan snapshot, hal ini bisa sangat membantu karena Anda dapat menggunakan kode yang serupa untuk memproses ekspor inkremental. Biasanya, ekspor ke S3 sedikit lebih murah dibanding opsi streaming sebelumnya, tetapi biaya biasanya bukan merupakan faktor utama dalam menentukan opsi yang akan digunakan.

- Anda umumnya hanya dapat memiliki dua konsumen simultan dari aliran DynamoDB. Pertimbangkan hal ini saat merencanakan strategi integrasi Anda.
- Jangan gunakan pemindaian untuk mendeteksi perubahan. Hal ini mungkin bekerja dalam skala kecil, tetapi menjadi tidak praktis dengan cepat.

Integrasi DynamoDB Zero-ETL dengan Layanan OpenSearch

DynamoDB memiliki integrasi DynamoDB [Zero-ETL dengan Amazon Service](#). OpenSearch Untuk informasi selengkapnya, lihat [plugin DynamoDB untuk Penyerapan dan praktik terbaik khusus OpenSearch untuk Layanan Amazon](#). OpenSearch

Konfigurasi

- Anda hanya perlu melakukan pencarian pada indeks data. Selalu gunakan templat pemetaan (`template_type: index_template` dan `template_content`) serta `include_keys` untuk mengimplementasikan hal ini.
- Pantau log Anda untuk kesalahan yang terkait dengan konflik tipe. OpenSearch Layanan mengharapkan semua nilai untuk kunci yang diberikan memiliki tipe yang sama. Pengecualian dihasilkan jika ada ketidakcocokan. Jika Anda menemukan salah satu kesalahan ini, Anda dapat menambahkan prosesor untuk mengetahui bahwa kunci yang diberikan selalu memiliki nilai yang sama.
- Secara umum, gunakan nilai metadata `primary_key` untuk nilai `document_id`. Dalam OpenSearch Layanan, ID dokumen setara dengan kunci utama di DynamoDB. Menggunakan kunci primer akan memudahkan dokumen Anda ditemukan dan memastikan bahwa pembaruan secara konsisten direplikasi tanpa konflik.

Anda dapat menggunakan fungsi pembantu `getMetadata` untuk mendapatkan kunci primer Anda (misalnya, `document_id: "${getMetadata('primary_key')}`"). Jika Anda menggunakan kunci primer komposit, fungsi pembantu akan menggabungkannya bersama-sama untuk Anda.

- Secara umum, gunakan nilai metadata `opensearch_action` untuk pengaturan `action`. Ini akan memastikan bahwa pembaruan direplikasi sedemikian rupa sehingga data di OpenSearch Layanan cocok dengan status terbaru di DynamoDB.

Anda dapat menggunakan fungsi pembantu `getMetadata` untuk mendapatkan kunci primer Anda (misalnya, `action: "${getMetadata('opensearch_action')}`"). Anda juga bisa mendapatkan tipe peristiwa `streaming dynamodb_event_name` untuk kasus penggunaan seperti pemfilteran. Namun, biasanya hal ini tidak digunakan untuk pengaturan `action`.

Observabilitas

- Selalu gunakan antrian huruf mati (DLQ) di OpenSearch wastafel Anda untuk menangani peristiwa yang jatuh. DynamoDB umumnya kurang terstruktur OpenSearch daripada Layanan, dan selalu mungkin terjadi sesuatu yang tidak terduga. Dengan antrian surat mati, Anda dapat memulihkan peristiwa individual, dan bahkan mengotomatiskan proses pemulihan. Hal ini akan membantu agar Anda tidak perlu membangun kembali seluruh indeks Anda.
- Selalu atur peringatan agar penundaan replikasi Anda tidak melebihi jumlah yang diharapkan. Biasanya aman untuk mengasumsikan satu menit tanpa peringatan yang terlalu berisik. Ini dapat

bervariasi tergantung pada seberapa runcing lalu lintas tulis Anda dan pengaturan OpenSearch Compute Unit (OCU) Anda di pipeline.

Jika penundaan replikasi Anda melebihi 24 jam, aliran Anda akan mulai kehilangan peristiwa, dan Anda akan mengalami masalah akurasi kecuali jika Anda melakukan pembangunan ulang indeks Anda dari awal.

Penskalaan

- Gunakan penskalaan otomatis untuk alur untuk membantu menaikkan atau menurunkan skala OCU agar sesuai dengan beban kerja.
- Untuk tabel throughput yang disediakan tanpa penskalaan otomatis, sebaiknya atur OCU berdasarkan unit kapasitas tulis (WCU) Anda dibagi 1000. Tetapkan minimum ke 1 OCU di bawah jumlah tersebut (tetapi setidaknya 1), dan atur maksimum ke setidaknya 1 OCU di atas jumlah tersebut.

- Rumus:

```
OCU_minimum = GREATEST((table_WCU / 1000) - 1, 1)
OCU_maximum = (table_WCU / 1000) + 1
```

- Contoh: Tabel Anda memiliki 25000 WCU yang disediakan. OCU alur Anda harus diatur dengan minimum 24 ($25000/1000 - 1$) dan maksimum setidaknya 26 ($25000/1000 + 1$).
- Untuk tabel throughput yang disediakan dengan penskalaan otomatis, sebaiknya atur OCU berdasarkan WCU minimum dan maksimum Anda, dibagi 1000. Atur minimum ke 1 OCU di bawah minimum dari DynamoDB, dan atur maksimum ke setidaknya 1 OCU di atas maksimum dari DynamoDB.

- Rumus:

```
OCU_minimum = GREATEST((table_minimum_WCU / 1000) - 1, 1)
OCU_maximum = (table_maximum_WCU / 1000) + 1
```

- Contoh: Tabel Anda memiliki kebijakan penskalaan otomatis dengan minimum 8000 dan maksimum 14000. OCU alur Anda harus diatur dengan minimum 7 ($8000/1000 - 1$) dan maksimum 15 ($14000/1000 + 1$).
- Untuk tabel throughput sesuai permintaan, sebaiknya atur OCU berdasarkan puncak dan penurunan tipikal Anda untuk unit permintaan tulis per detik. Anda mungkin perlu membuat rata-rata dalam jangka waktu yang lebih lama, tergantung pada agregasi yang tersedia untuk Anda.

Atur minimum ke 1 OCU di bawah minimum dari DynamoDB, dan atur maksimum ke setidaknya 1 OCU di atas maksimum dari DynamoDB.

- Rumus:

```
# 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
```

- Contoh: Tabel Anda memiliki penurunan rata-rata 300 unit permintaan tulis per detik dan puncak rata-rata 4300. OCU alur Anda harus diatur dengan minimum 1 ($300/1000 - 1$, tetapi setidaknya 1) dan maksimum 5 ($4300/1000 + 1$).
- Ikuti praktik terbaik dalam penskalaan indeks OpenSearch Layanan tujuan Anda. Jika penskalaan indeks Anda kurang, ini akan memperlambat konsumsi dari DynamoDB, dan dapat menyebabkan penundaan.

Note

[GREATEST](#) adalah fungsi SQL yang, jika diberikan sekumpulan argumen, mengembalikan argumen dengan nilai terbesar.

Layanan, akun, dan tabel kuota di Amazon DynamoDB

Bagian ini menjelaskan kuota saat ini, yang sebelumnya disebut sebagai batas, dalam Amazon DynamoDB. Kecuali ditentukan lain, masing-masing kuota berlaku untuk setiap Wilayah.

Topik

- [Throughput dan mode kapasitas baca/tulis](#)
- [Kapasitas Terpesan](#)
- [Kuota impor](#)
- [Wawasan Kontributor](#)
- [Tabel](#)
- [Tabel global](#)
- [Indeks sekunder](#)
- [Tombol partisi dan kunci urutan](#)
- [Peraturan penamaan](#)
- [Tipe Data](#)
- [Item](#)
- [Atribut](#)
- [Parameter ekspresi](#)
- [Transaksi DynamoDB](#)
- [DynamoDB Streams](#)
- [DynamoDB Accelerator \(DAX\)](#)
- [Batasan khusus API](#)
- [Enkripsi DynamoDB saat istirahat](#)
- [Ekspor tabel ke Amazon S3](#)
- [Pencadangan dan pemulihan](#)

Throughput dan mode kapasitas baca/tulis

Anda dapat beralih di antara mode kapasitas baca/tulis satu kali setiap 24 jam. Satu-satunya pengecualian untuk ini adalah jika Anda mengalihkan tabel mode yang disediakan ke mode sesuai permintaan: Anda dapat beralih kembali ke mode yang disediakan dalam periode 24 jam yang sama.

Ukuran unit kapasitas (untuk tabel yang disediakan)

Satu unit kapasitas baca = satu bacaan sangat konsisten per detik, atau dua bacaan akhir konsisten per detik, untuk sebuah item hingga ukuran 4 KB.

Satu unit kapasitas tulis = satu tulis per detik, untuk item berukuran hingga 1 KB.

Permintaan baca transaksional memerlukan dua unit kapasitas baca untuk melakukan satu pembacaan per detik untuk item hingga 4 KB.

Permintaan tulis transaksional memerlukan dua unit kapasitas tulis untuk melakukan satu penulisan per detik untuk item hingga 1 KB.

Ukuran unit permintaan (untuk tabel sesuai permintaan)

Satu unit kapasitas baca = satu bacaan sangat konsisten per detik, atau dua bacaan akhir konsisten per detik, untuk item hingga ukuran 4 KB.

Satu unit permintaan tulis = satu tulis per detik, untuk item hingga ukuran 1 KB.

Permintaan baca transaksional memerlukan dua unit permintaan baca untuk melakukan satu pembacaan per detik untuk item hingga 4 KB.

Permintaan tulis transaksional memerlukan dua unit permintaan tulis untuk melakukan satu penulisan per detik untuk item hingga 1 KB.

Kuota default throughput

AWS menempatkan beberapa kuota default pada throughput yang dapat disediakan dan dikonsumsi akun Anda dalam suatu Wilayah.

Kuota throughput baca tingkat akun dan throughput tulis tingkat akun berlaku di tingkat akun. Kuota tingkat akun ini berlaku untuk jumlah kapasitas throughput yang disediakan untuk semua tabel akun Anda dan indeks sekunder global di Wilayah tertentu. Semua throughput akun yang tersedia dapat disediakan untuk satu tabel atau beberapa tabel. Kuota ini hanya berlaku untuk tabel yang menggunakan mode kapasitas yang disediakan.

Kuota throughput baca tingkat tabel dan throughput tulis tingkat tabel berlaku berbeda pada tabel yang menggunakan mode kapasitas yang disediakan, dan tabel yang menggunakan mode kapasitas sesuai permintaan.

Untuk GSI dan tabel mode kapasitas yang disediakan, kuota adalah jumlah maksimum unit kapasitas baca dan tulis yang dapat disediakan untuk setiap tabel atau GSI-nya di Wilayah. Total tabel individual dan semua GSI-nya juga harus tetap berada di bawah kuota throughput baca dan tulis tingkat akun. Hal ini merupakan tambahan dari persyaratan bahwa total semua tabel yang disediakan dan GSI-nya harus tetap berada di bawah kuota throughput baca dan tulis tingkat akun.

Untuk tabel mode kapasitas dan GSI sesuai permintaan, kuota tingkat tabel adalah unit kapasitas baca dan tulis maksimum yang tersedia untuk tabel mana pun, atau GSI individual mana pun dalam tabel tersebut. Tidak ada kuota throughput baca dan tulis tingkat akun yang diterapkan pada tabel dalam mode sesuai permintaan.

Di bawah ini adalah kuota throughput yang berlaku di akun Anda, secara default.

	Sesuai Permintaan	Disediakan	Dapat Disesuaikan
Per table	40,000 read request units and 40,000 write request units	40,000 read capacity units and 40,000 write capacity units	Ya
Per account	Not applicable	80,000 read capacity units and 80,000 write capacity units	Ya
Minimum throughput for any table or global secondary index	Not applicable	1 read capacity unit and 1 write capacity unit	Ya

Anda dapat menggunakan [konsol Kuota Layanan](#), [AWS API](#), dan [AWS CLI](#) untuk meminta peningkatan kuota untuk kuota yang dapat disesuaikan bila diperlukan.

[Untuk kuota throughput tingkat akun, Anda dapat menggunakan konsol Service Quotas, konsol, API, dan AWS CLI untuk membuat CloudWatch alarm AWS dan diberi tahu secara otomatis saat penggunaan Anda saat ini mencapai persentase tertentu dari nilai kuota yang diterapkan.](#) [AWS CloudWatch](#) Menggunakan CloudWatch Anda juga dapat memantau

penggunaan Anda dengan melihat metrik `AccountProvisionedReadCapacityUnits` dan `AccountProvisionedWriteCapacityUnits` AWS penggunaan. Untuk mempelajari selengkapnya tentang metrik penggunaan, lihat [metrik penggunaan AWS](#).

Meningkatkan atau mengurangi throughput (untuk tabel yang disediakan)

Meningkatkan throughput yang disediakan

Anda dapat meningkatkan `ReadCapacityUnits` atau `WriteCapacityUnits` sesering yang diperlukan, menggunakan operasi `AWS Management Console` atau `UpdateTable`. Dalam satu panggilan, Anda dapat meningkatkan throughput yang disediakan untuk sebuah tabel, untuk indeks sekunder global apa pun pada tabel tersebut, atau untuk kombinasi semuanya. Pengaturan baru tidak berlaku sampai operasi `UpdateTable` selesai.

Anda tidak dapat melebihi kuota per akun saat Anda menambahkan kapasitas yang disediakan, dan DynamoDB tidak mengizinkan Anda meningkatkan kapasitas yang disediakan dengan sangat cepat. Selain pembatasan ini, Anda dapat meningkatkan kapasitas yang disediakan untuk tabel Anda sebanyak yang Anda perlukan. Untuk informasi selengkapnya tentang kuota per akun, lihat bagian sebelumnya, [Kuota default throughput](#).

Menurunkan throughput yang disediakan

Untuk setiap tabel dan indeks sekunder global dalam operasi `UpdateTable`, Anda dapat mengurangi `ReadCapacityUnits` atau `WriteCapacityUnits` (atau keduanya). Pengaturan baru tidak berlaku sampai operasi `UpdateTable` selesai.

Terdapat kuota default pada jumlah penurunan kapasitas yang tersedia yang dapat Anda lakukan pada tabel DynamoDB per hari. Satu hari ditentukan berdasarkan Waktu Universal Terkoordinasi (UTC). Pada hari tertentu, Anda dapat memulai dengan melakukan hingga empat penurunan dalam satu jam selama Anda belum melakukan penurunan lainnya pada hari tersebut. Selanjutnya, Anda dapat melakukan satu penurunan tambahan per jam (setiap 60 menit sekali). Ini secara efektif membawa jumlah maksimum penurunan dalam sehari menjadi 27 kali.

Anda dapat menggunakan [konsol Kuota Layanan](#), [AWS API](#), dan [AWS CLI](#) untuk meminta peningkatan kuota, bila diperlukan.

Important

Batas penurunan tabel dan indeks sekunder global dipisahkan, sehingga setiap indeks sekunder global untuk tabel tertentu memiliki batas penurunannya sendiri. Namun, jika satu

permintaan menurunkan throughput untuk tabel dan indeks sekunder global, permintaan tersebut akan ditolak jika salah satu permintaan tersebut melebihi batas saat ini. Permintaan tidak diproses sebagian.

Example

Dalam 4 jam pertama suatu hari, tabel dengan indeks sekunder global dapat dimodifikasi sebagai berikut:

- Kurangi `WriteCapacityUnits` atau `ReadCapacityUnits` tabel (atau keduanya) sebanyak empat kali.
- Kurangi `WriteCapacityUnits` atau `ReadCapacityUnits` (atau keduanya) dari indeks sekunder global sebanyak empat kali.

Pada akhir hari yang sama, tabel dan throughput indeks sekunder global berpotensi mengalami penurunan masing-masing sebanyak 27 kali lipat.

Kapasitas Terpesan

AWS menempatkan kuota default pada jumlah kapasitas cadangan aktif yang dapat dibeli akun Anda. Batas kuota merupakan kombinasi kapasitas terpesan untuk unit kapasitas tulis (WCU) dan unit kapasitas baca (RCU).

	Kapasitas terpesan aktif	Dapat Disesuaikan
Per akun	1.000.000 unit kapasitas yang disediakan (WCU _ RCU)	Ya

Jika Anda mencoba membeli lebih dari 1.000.000 unit kapasitas yang disediakan dalam satu pembelian, Anda akan menerima kesalahan untuk batas kuota layanan ini. Jika Anda memiliki kapasitas terpesan aktif dan mencoba membeli kapasitas terpesan tambahan yang akan menghasilkan lebih dari 1.000.000 unit kapasitas aktif yang disediakan, Anda akan menerima kesalahan untuk batas kuota layanan ini.

Jika Anda memerlukan kapasitas terpesan lebih dari 1.000.000 unit kapasitas yang disediakan, Anda dapat meminta penambahan kuota dengan mengirimkan permintaan ke tim [dukungan](#).

Kuota impor

Impor DynamoDB dari Amazon S3 dapat mendukung hingga 50 pekerjaan impor bersamaan dengan total ukuran objek sumber impor sebesar 15 TB sekaligus di wilayah us-east-1, us-west-2, dan eu-west-1. Di seluruh wilayah lainnya, didukung hingga 50 tugas impor bersamaan dengan ukuran total 1 TB. Setiap pekerjaan impor dapat mengambil hingga 50.000 objek Amazon S3 di semua wilayah. Untuk informasi selengkapnya tentang impor dan validasi, lihat [kuota format impor dan validasi](#).

Wawasan Kontributor

Saat mengaktifkan Customer Insights di tabel DynamoDB, Anda masih tunduk pada batasan aturan Contributor Insights. Untuk informasi selengkapnya, lihat [CloudWatch service quotas](#).

Tabel

Ukuran tabel

Tidak ada batas praktis pada ukuran tabel. Tabel tidak dibatasi dalam jumlah item atau jumlah byte.

Jumlah maksimum tabel per akun per wilayah

Untuk AWS akun apa pun, ada kuota awal 2.500 tabel per AWS Wilayah.

Jika Anda membutuhkan lebih dari 2.500 tabel untuk satu akun, silakan hubungi tim AWS akun Anda untuk menjelajahi peningkatan hingga maksimum 10.000 tabel. Untuk lebih dari 10.000, praktik terbaik yang disarankan adalah menyiapkan beberapa akun, yang masing-masing dapat melayani hingga 10.000 tabel.

Anda dapat menggunakan [konsol Kuota Layanan](#), [API AWS](#), dan [AWS CLI](#) untuk melihat nilai kuota default dan yang diterapkan untuk jumlah tabel maksimum di akun Anda, dan untuk meminta peningkatan kuota, bila diperlukan. Anda juga dapat meminta penambahan kuota dengan memotong tiket ke [dukungan AWS](#)

Menggunakan [konsol Service Quotas](#), [AWS API](#) dan [AWS CLI](#) Anda dapat membuat CloudWatch alarm untuk mendapatkan notifikasi secara otomatis ketika penggunaan Anda saat ini mencapai persentase tertentu dari kuota Anda saat ini. Menggunakan CloudWatch Anda juga dapat memantau penggunaan Anda dengan melihat metrik TableCount AWS penggunaan. Untuk mempelajari selengkapnya tentang metrik penggunaan, lihat [metrik penggunaan AWS](#).

Tabel global

AWS menempatkan beberapa kuota default pada throughput yang dapat Anda sediakan atau gunakan saat menggunakan tabel global.

	Sesuai Permintaan	Disediakan
Per table	40,000 read request units and 40,000 write request units	40,000 read capacity units and 40,000 write capacity units
Per table, per destination Region, per day	10 TB for all source tables to which a replica was added for this destination Region	10 TB for all source tables to which a replica was added for this destination Region

Operasi transaksional memberikan jaminan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) hanya di AWS Wilayah tempat penulisan dibuat awalnya. Transaksi tidak didukung di seluruh Wilayah dalam tabel global. Misalnya, Anda memiliki tabel global dengan replika di Wilayah AS Timur (Ohio) dan AS Barat (Oregon) dan Anda melakukan TransactWriteItems operasi di Wilayah AS Timur (Virginia N.). Dalam hal ini, Anda mungkin mengamati transaksi yang diselesaikan sebagian di Wilayah AS Barat (Oregon) saat perubahan direplikasi. Perubahan direplikasi ke Wilayah lain hanya setelah diterapkan di Wilayah sumber.

Note

Mungkin ada contoh di mana Anda perlu meminta kenaikan batas kuota melalui AWS Support Jika salah satu hal berikut ini berlaku untuk Anda, lihat <https://aws.amazon.com/support>:

- Jika Anda menambahkan replika untuk tabel yang dikonfigurasi untuk menggunakan lebih dari 40.000 unit kapasitas tulis (WCU), Anda harus meminta peningkatan kuota layanan untuk penambahan kuota WCU replika Anda.
- Jika Anda menambahkan replika atau replika ke satu Wilayah tujuan dalam jangka waktu 24 jam dengan total gabungan lebih besar dari 10 TB, Anda harus meminta peningkatan kuota layanan untuk menambah kuota pengisian ulang data replika.

- Jika Anda mengalami kesalahan seperti berikut:
 - Tidak dapat membuat replika tabel 'example_table' di wilayah 'example_region_A' karena melebihi batas akun Anda saat ini di wilayah 'example_region_B'.

Indeks sekunder

Indeks sekunder per tabel

Anda dapat menentukan maksimum 5 indeks sekunder lokal.

Terdapat kuota default sebanyak 20 indeks sekunder global per tabel. Anda dapat menggunakan [konsol Kuota Layanan](#), [API AWS](#), dan [AWS CLI](#) untuk memeriksa indeks sekunder global per tabel default dan kuota saat ini yang berlaku untuk akun Anda, dan untuk meminta peningkatan kuota, bila diperlukan. Anda juga dapat meminta penambahan kuota dengan memotong tiket ke <https://aws.amazon.com/support>.

Anda hanya dapat membuat atau menghapus satu indeks sekunder global per operasi `UpdateTable`.

Atribut Indeks Sekunder yang diproyeksikan per tabel

Anda dapat memproyeksikan total hingga 100 atribut ke semua indeks sekunder lokal dan global tabel. Ini hanya berlaku untuk atribut proyeksi yang ditentukan pengguna.

Dalam operasi `CreateTable`, jika Anda menentukan `ProjectionType` dari `INCLUDE`, jumlah total atribut yang ditentukan dalam `NonKeyAttributes`, yang dijumlahkan di semua indeks sekunder, tidak boleh melebihi 100. Jika Anda memproyeksikan nama atribut yang sama ke dalam dua indeks berbeda, ini dihitung sebagai dua atribut yang berbeda ketika menentukan total.

Batasan ini tidak berlaku untuk indeks sekunder dengan `ProjectionType` `KEYS_ONLY` atau `ALL`.

Tombol partisi dan kunci urutan

Panjang kunci partisi

Panjang minimum nilai kunci partisi adalah 1 byte. Panjang maksimum adalah 2048 byte.

Nilai kunci partisi

Tidak ada batasan praktis untuk jumlah nilai kunci partisi yang berbeda, untuk tabel atau indeks sekunder.

Panjang kunci urutan

Panjang minimum nilai kunci urutan adalah 1 byte. Panjang maksimum adalah 1024 byte.

Nilai kunci urutan

Secara umum, tidak ada batas praktis pada jumlah nilai kunci urutan berbeda per nilai kunci partisi.

Pengecualian ini adalah untuk tabel dengan indeks sekunder. Koleksi item adalah set item yang memiliki nilai atribut kunci partisi yang sama. Dalam indeks sekunder global, kumpulan item tidak bergantung pada tabel dasar (dan dapat memiliki atribut kunci partisi yang berbeda), tetapi dalam indeks sekunder lokal, tampilan yang diindeks ditempatkan di partisi yang sama dengan item dalam tabel dan berbagi atribut kunci partisi yang sama. Sebagai akibat dari lokalitas ini, ketika sebuah tabel memiliki satu atau lebih LSI, set item tidak dapat didistribusikan ke beberapa partisi.

Untuk tabel dengan satu atau lebih LSI, ukuran koleksi item tidak boleh melebihi 10 GB. Ini mencakup semua item tabel dasar dan semua tampilan LSI yang diproyeksikan yang memiliki nilai atribut kunci partisi yang sama. 10 GB adalah ukuran maksimum sebuah partisi. Untuk informasi selengkapnya, lihat [Batas ukuran koleksi barang](#).

Peraturan penamaan

Nama tabel dan nama Indeks Sekunder

Nama untuk tabel dan indeks sekunder harus memiliki panjang minimal 3 karakter, namun tidak lebih dari 255 karakter. Berikut ini adalah karakter yang diperbolehkan:

- A-Z
- a-z
- 0-9
- _ (garis bawah)
- - (tanda hubung)
- . (titik)

Biner

Panjang Biner dibatasi oleh ukuran item maksimum 400 KB.

Aplikasi yang bekerja dengan atribut Binari harus mengkodekan data dalam format base64 sebelum mengirimkannya ke DynamoDB. Setelah menerima data, DynamoDB mendekodekannya ke dalam array byte yang tidak ditandatangani dan menggunakannya sebagai panjang atribut.

Item

Ukuran item

Ukuran item maksimum di DynamoDB adalah 400 KB, yang mencakup panjang biner nama atribut (panjang UTF-8) dan panjang nilai atribut (sekali lagi panjang biner). Nama atribut dihitung terhadap batas ukuran.

Misalnya, pertimbangkan sebuah item dengan dua atribut: satu atribut bernama "warna baju" dengan nilai "R" dan atribut lainnya bernama "ukuran baju" dengan nilai "M". Ukuran total item tersebut adalah 23 byte.

Ukuran item untuk tabel dengan Indeks Sekunder Lokal

Untuk setiap indeks sekunder lokal pada tabel, terdapat batas 400 KB untuk total hal berikut:

- Ukuran data item dalam tabel.
- Ukuran entri terkait (termasuk nilai kunci dan atribut yang diproyeksikan) di semua indeks sekunder lokal.

Atribut

Pasangan atribut nama-nilai per item

Ukuran kumulatif atribut per item harus sesuai dengan ukuran item DynamoDB maksimum (400 KB).

Jumlah nilai dalam daftar, peta, atau set

Tidak ada batasan jumlah nilai dalam Daftar, Peta, atau Kumpulan, selama item yang berisi nilai tersebut sesuai dengan batas ukuran item 400 KB.

Nilai atribut

Nilai atribut String dan Binari kosong diperbolehkan, jika atribut tersebut tidak digunakan sebagai atribut kunci untuk tabel atau indeks. Nilai String dan Biner kosong diperbolehkan di dalam tipe Set, Daftar, dan Peta. Nilai atribut tidak boleh berupa Set kosong (Set String, Set Angka, atau Set Biner). Namun, Daftar dan Peta kosong diperbolehkan.

Kedalaman atribut bertingkat

DynamoDB mendukung atribut bertingkat hingga sedalam 32 tingkat.

Parameter ekspresi

Parameter ekspresi termasuk `ProjectionExpression`, `ConditionExpression`, `UpdateExpression`, dan `FilterExpression`.

Panjang

Panjang maksimum string ekspresi adalah 4 KB. Misalnya, ukuran `ConditionExpression a=b` adalah 3 byte.

Panjang maksimum nama atribut ekspresi tunggal atau nilai atribut ekspresi adalah 255 byte. Misalnya, `#name` adalah 5 byte; `:val` adalah 4 byte.

Panjang maksimum semua variabel substitusi dalam sebuah ekspresi adalah 2 MB. Ini adalah jumlah panjang semua `ExpressionAttributeNames` dan `ExpressionAttributeValues`.

Operator dan operan

Jumlah maksimum operator atau fungsi yang diperbolehkan dalam `UpdateExpression` adalah 300. Misalnya, `UpdateExpressionSET a = :val1 + :val2 + :val3` berisi dua "+" operator.

Jumlah maksimum operan untuk komparator IN adalah 100.

Kata yang dicadangkan

DynamoDB tidak mencegah Anda menggunakan nama yang bertentangan dengan kata khusus. (Untuk daftar lengkap, lihat [Disimpan kata-kata di DynamoDB.](#))

Namun, jika Anda menggunakan kata khusus dalam parameter ekspresi, Anda juga harus menentukan `ExpressionAttributeNames`. Untuk informasi selengkapnya, lihat [Nama atribut ekspresi di DynamoDB](#).

Transaksi DynamoDB

Operasi API transaksional DynamoDB memiliki batasan berikut:

- Transaksi tidak boleh berisi lebih dari 100 item unik.
- Sebuah transaksi tidak boleh berisi lebih dari 4 MB data.
- Tidak ada dua tindakan dalam suatu transaksi yang dapat bekerja terhadap item yang sama dalam tabel yang sama. Misalnya, Anda tidak dapat melakukan `ConditionCheck` dan `Update` item yang sama dalam satu transaksi.
- Transaksi tidak dapat beroperasi pada tabel di lebih dari satu AWS akun atau Wilayah.
- Operasi transaksional memberikan jaminan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) hanya di AWS Wilayah tempat penulisan dibuat awalnya. Transaksi tidak didukung di seluruh Wilayah dalam tabel global. Misalnya, Anda memiliki tabel global dengan replika di Wilayah AS Timur (Ohio) dan AS Barat (Oregon) dan Anda melakukan operasi `TransactWriteItems` di Wilayah AS Timur (Virginia Utara). Dalam hal ini, Anda mungkin mengamati transaksi yang diselesaikan sebagian di Wilayah AS Barat (Oregon) saat perubahan direplikasi. Perubahan direplikasi ke Wilayah lain hanya setelah diterapkan di Wilayah sumber.

DynamoDB Streams

Pembaca serpihan secara bersamaan di DynamoDB Streams

Untuk tabel Wilayah tunggal yang bukan tabel global, Anda dapat merancang hingga dua proses untuk membaca dari serpihan DynamoDB Streams yang sama secara bersamaan. Melebihi batas ini dapat mengakibatkan throttling permintaan. Untuk tabel global, kami menyarankan Anda membatasi jumlah pembaca simultan menjadi satu untuk menghindari throttling permintaan.

Kapasitas tulis maksimum untuk tabel dengan DynamoDB Streams diaktifkan

AWS menempatkan beberapa kuota default pada kapasitas tulis untuk tabel DynamoDB dengan DynamoDB Streams diaktifkan. Kuota default ini hanya berlaku untuk tabel dalam mode kapasitas

baca/tulis yang disediakan. Berikut ini adalah kuota throughput yang berlaku pada akun Anda secara default.

- Wilayah AS Timur (Virginia Utara), AS Timur (Ohio), AS Barat (California Utara), AS Barat (Oregon), Amerika Selatan (Sao Paulo), Eropa (Frankfurt), Eropa (Irlandia), Asia Pasifik (Tokyo), Asia Pasifik (Seoul), Asia Pasifik (Singapura), Asia Pasifik (Sydney), Tiongkok (Beijing):
 - Per tabel – 40.000 unit kapasitas tulis
- Semua Wilayah lainnya:
 - Per tabel – 10.000 unit kapasitas tulis

Anda dapat menggunakan [konsol Kuota Layanan](#), [API AWS](#), dan [AWS CLI](#) untuk memeriksa kapasitas tulis maksimum untuk tabel dengan DynamoDB Streams yang diaktifkan secara default dan kuota saat ini yang berlaku di akun Anda, dan untuk meminta peningkatan kuota, bila diperlukan. Anda juga dapat meminta penambahan kuota dengan memotong tiket ke [dukungan AWS](#).

Note

Kuota throughput yang disediakan juga berlaku untuk tabel DynamoDB dengan Aliran DynamoDB diaktifkan. Saat Anda meminta peningkatan kuota pada kapasitas tulis untuk tabel dengan Streams diaktifkan, pastikan Anda juga meminta peningkatan kapasitas throughput yang disediakan untuk tabel ini. Untuk informasi selengkapnya, lihat [Kuota Default Throughput](#). Kuota lain juga berlaku saat memproses Aliran DynamoDB dengan throughput lebih tinggi. Untuk informasi selengkapnya, lihat [Panduan referensi API Amazon DynamoDB Streams](#).

DynamoDB Accelerator (DAX)

AWS ketersediaan wilayah

Untuk daftar AWS Wilayah di mana DAX tersedia, lihat [DynamoDB Accelerator \(DAX\)](#) di Referensi Umum AWS

Simpul

Klaster DAX terdiri dari tepat satu simpul primer, dan antara nol hingga sepuluh simpul replika baca.

Jumlah total node (per AWS akun) tidak dapat melebihi 50 dalam satu AWS Wilayah.

Grup parameter

Anda dapat membuat hingga 20 grup parameter DAX per Wilayah.

Grup subnet

Anda dapat membuat hingga 50 grup subnet DAX per Wilayah.

Dalam grup subnet, Anda dapat menentukan hingga 20 subnet.

Batasan khusus API

CreateTable/UpdateTable/DeleteTable/PutResourcePolicy/DeleteResourcePolicy

Secara umum, Anda dapat memiliki hingga 500 [CreateTable](#),, [UpdateTableDeleteTablePutResourcePolicy](#), dan [DeleteResourcePolicy](#) permintaan yang berjalan secara bersamaan dalam kombinasi apa pun. Akibatnya, jumlah total tabel dalam status CREATING, UPDATING, atau DELETING tidak boleh melebihi 500.

Anda dapat mengirimkan hingga 2.500 permintaan per detik permintaan API bidang kontrol yang dapat diubah ([CreateTableDeleteTableUpdateTablePutResourcePolicy](#),, dan [DeleteResourcePolicy](#)) di seluruh grup tabel. Namun, [DeleteResourcePolicy](#) permintaan [PutResourcePolicy](#) dan memiliki batas individu yang lebih rendah. Untuk informasi selengkapnya, lihat detail kuota berikut untuk [PutResourcePolicy](#) dan [DeleteResourcePolicy](#).

[CreateTable](#) dan [PutResourcePolicy](#) permintaan yang mencakup kebijakan berbasis sumber daya akan dihitung sebagai dua permintaan tambahan untuk setiap KB kebijakan. Misalnya, [PutResourcePolicy](#) permintaan [CreateTable](#) atau dengan kebijakan ukuran 5 KB akan dihitung sebagai 11 permintaan. 1 untuk [CreateTable](#) permintaan dan 10 untuk kebijakan berbasis sumber daya (2 x 5 KB). Demikian pula, kebijakan ukuran 20 KB akan dihitung sebagai 41 permintaan. 1 untuk [CreateTable](#) permintaan dan 40 untuk kebijakan berbasis sumber daya (2 x 20 KB).

PutResourcePolicy

Anda dapat mengirimkan hingga 25 permintaan [PutResourcePolicy](#) API per detik di seluruh grup tabel. Setelah permintaan yang berhasil untuk tabel individual, tidak ada [PutResourcePolicy](#) permintaan baru yang didukung selama 15 detik berikutnya.

Ukuran maksimum yang didukung untuk dokumen kebijakan berbasis sumber daya adalah 20 KB. DynamoDB menghitung spasi putih saat menghitung ukuran kebijakan terhadap batas ini.

DeleteResourcePolicy

Anda dapat mengirimkan hingga 50 permintaan DeleteResourcePolicy API per detik di seluruh grup tabel. Setelah PutResourcePolicy permintaan yang berhasil untuk tabel individual, tidak ada DeleteResourcePolicy permintaan yang didukung selama 15 detik berikutnya.

BatchGetItem

Satu operasi BatchGetItem dapat mengambil maksimal 100 item. Ukuran total semua item yang diambil tidak boleh melebihi 16 MB.

BatchWriteItem

Satu operasi BatchWriteItem dapat berisi hingga 25 permintaan PutItematau DeleteItem. Ukuran total semua item yang ditulis tidak boleh melebihi 16 MB.

DescribeStream

Anda dapat menelepon dengan DescribeStream kecepatan maksimum 10 kali per detik.

DescribeTableReplicaAutoScaling

Metode DescribeTableReplicaAutoScaling hanya mendukung 10 permintaan per detik.

DescribeLimits

DescribeLimits harus dipanggil hanya secara berkala. Anda dapat mengalami kesalahan throttling jika Anda memanggilnya lebih dari sekali dalam satu menit.

DescribeContributorInsights/ListContributorInsights/UpdateContributorInsights

DescribeContributorInsights, ListContributorInsights, dan UpdateContributorInsights hanya boleh dipanggil secara berkala. DynamoDB mendukung hingga lima permintaan per detik untuk masing-masing API ini.

DescribeTable/ListTables/GetResourcePolicy

Anda dapat mengirimkan hingga 2.500 permintaan per detik dari kombinasi permintaan API bidang kontrol hanya-baca (`DescribeTableListTables`, dan `GetResourcePolicy`). `GetResourcePolicy` API memiliki batas individu yang lebih rendah yaitu 100 permintaan per detik.

Query

Hasil yang ditetapkan dari `Query` dibatasi hingga 1 MB per panggilan. Anda dapat menggunakan `LastEvaluatedKey` dari respons kueri untuk mengambil lebih banyak hasil.

Scan

Hasil yang ditetapkan dari `Scan` dibatasi hingga 1 MB per panggilan. Anda dapat menggunakan `LastEvaluatedKey` dari respons pemindaian untuk mengambil hasil lainnya.

UpdateKinesisStreamingDestination

Saat melakukan `UpdateKinesisStreamingDestination` operasi, Anda dapat mengatur `ApproximateCreationDateTimePrecision` ke nilai baru maksimal 3 kali dalam periode 24 jam.

UpdateTableReplicaAutoScaling

Metode `UpdateTableReplicaAutoScaling` hanya mendukung sepuluh permintaan per detik.

UpdateTableTimeToLive

Metode `UpdateTableTimeToLive` hanya mendukung satu permintaan untuk mengaktifkan atau menonaktifkan `Time to Live (TTL)` per tabel tertentu per jam. Perubahan ini memerlukan waktu hingga satu jam untuk diproses sepenuhnya. Setiap `UpdateTimeToLive` panggilan tambahan untuk tabel yang sama selama durasi satu jam ini menghasilkan a `ValidationException`.

Enkripsi DynamoDB saat istirahat

Anda dapat beralih antara kunci yang dikelola AWS dikelola pelanggan hingga empat kali, kapan saja per jendela 24 jam, berdasarkan per tabel, mulai dari saat tabel dibuat. Kunci milik AWS Jika tidak ada perubahan dalam enam jam terakhir, perubahan tambahan diperbolehkan. Hal ini secara efektif menjadikan jumlah maksimum perubahan dalam sehari menjadi delapan (empat perubahan dalam enam jam pertama, dan satu perubahan untuk masing-masing jendela enam jam berikutnya dalam sehari).

Anda dapat mengganti kunci enkripsi untuk menggunakan Kunci milik AWS sesering yang diperlukan, bahkan jika kuota di atas telah habis.

Ini adalah kuota kecuali jika Anda meminta jumlah yang lebih tinggi. Untuk meminta peningkatan kuota layanan, lihat <https://aws.amazon.com/support>.

Ekspor tabel ke Amazon S3

Ekspor penuh: hingga 300 tugas ekspor bersamaan, atau hingga total 100TB dari semua ekspor tabel dalam penerbangan, dapat diekspor. Kedua batas ini diperiksa sebelum ekspor dimasukkan ke dalam antrean.

Ekspor tambahan: hingga 300 pekerjaan bersamaan, atau ukuran tabel 100TB, dalam jendela periode ekspor antara minimum 15 menit dan maksimum 24 jam, dapat diekspor secara bersamaan.

Pencadangan dan pemulihan

Saat memulihkan melalui pencadangan sesuai permintaan DynamoDB, Anda dapat menjalankan hingga 50 pemulihan bersamaan dengan total 50 TB. Saat memulihkan AWS Backup, Anda dapat mengeksekusi hingga 50 mengembalikan total 25TB secara bersamaan. Untuk informasi selengkapnya tentang pencadangan, lihat [Menggunakan cadangan Sesuai Permintaan dan DynamoDB Permintaan](#).

Referensi API tingkat rendah

Parameter [Referensi API Amazon DynamoDB](#) berisi daftar lengkap operasi yang didukung oleh:

- [DynamoDB](#).
- [DynamoDB Streams](#).
- [DynamoDB Accelerator \(DAX\)](#).

Pemecahan Masalah Amazon DynamoDB

Topik berikut memberikan saran pemecahan masalah untuk kesalahan dan masalah yang mungkin Anda temui saat menggunakan Amazon DynamoDB. Jika Anda menemukan masalah yang tidak tercantum di sini, Anda dapat menggunakan tombol Umpan Balik di halaman ini untuk melaporkannya.

Untuk saran dan jawaban pemecahan masalah selengkapnya terhadap pertanyaan dukungan umum, kunjungi [Pusat Pengetahuan AWS](#).

Topik

- [Memecahkan masalah latensi di Amazon DynamoDB](#)
- [Memecahkan masalah pembatasan di Amazon DynamoDB](#)

Memecahkan masalah latensi di Amazon DynamoDB

Jika beban kerja Anda tampaknya mengalami latensi tinggi, Anda dapat menganalisis CloudWatch `SuccessfulRequestLatency` metrik, dan memeriksa latensi rata-rata untuk melihat apakah itu terkait dengan DynamoDB. Beberapa variabilitas dalam `SuccessfulRequestLatency` yang dilaporkan adalah normal, dan lonjakan yang terjadi sesekali (khususnya dalam statistik `Maximum`) tidak perlu dikhawatirkan. Namun, jika statistik `Average` menunjukkan peningkatan tajam dan terus berlanjut, Anda harus memeriksa Service Health Dashboard dan Personal Health Dashboard AWS Anda untuk informasi selengkapnya. Beberapa kemungkinan penyebabnya mencakup ukuran item di tabel Anda (item 1kb dan item 400kb akan bervariasi dalam latensi) atau ukuran kueri (10 item versus 100 item).

Jika perlu, pertimbangkan untuk membuka kasus dukungan dengan AWS Support, dan terus menilai opsi cadangan yang tersedia untuk aplikasi Anda (seperti evakuasi Wilayah jika Anda memiliki arsitektur multi-Wilayah) sesuai dengan runbook Anda. Anda harus mencatat ID permintaan untuk permintaan lambat untuk memberikan ID ini AWS Support ketika Anda membuka kasus dukungan.

Metrik `SuccessfulRequestLatency` hanya mengukur latensi yang bersifat internal pada layanan DynamoDB - aktivitas sisi klien dan waktu perjalanan jaringan tidak disertakan. Untuk mempelajari selengkapnya tentang latensi keseluruhan untuk panggilan dari klien Anda ke layanan DynamoDB, Anda dapat mengaktifkan pencatatan metrik latensi di AWS SDK Anda.

Note

Untuk sebagian besar operasi tunggal (operasi yang berlaku pada satu item dengan menentukan sepenuhnya nilai kunci primer), DynamoDB memberikan `AverageSuccessfulRequestLatency` milidetik satu digit. Nilai ini tidak termasuk overhead transport untuk kode pemanggil yang mengakses titik akhir DynamoDB. Untuk operasi data multi-item, latensi akan bervariasi berdasarkan faktor-faktor seperti ukuran set hasil, kompleksitas struktur data yang dikembalikan, dan ekspresi kondisi dan ekspresi filter apa pun yang diterapkan. Untuk operasi multi-item berulang pada kumpulan data yang sama dengan parameter yang sama, DynamoDB akan memberikan `AverageSuccessfulRequestLatency` yang sangat konsisten.

Pertimbangkan satu atau beberapa strategi berikut untuk mengurangi latensi:

- Sesuaikan batas waktu permintaan dan perilaku percobaan ulang: Jalur dari klien Anda ke DynamoDB melintasi banyak komponen, yang masing-masing dirancang dengan mempertimbangkan redundansi. Pikirkan tentang cakupan ketahanan jaringan, batas waktu paket TCP, dan arsitektur terdistribusi DynamoDB itu sendiri. Perilaku SDK default dirancang untuk menemukan keseimbangan yang tepat untuk sebagian besar aplikasi. Jika latensi terbaik adalah prioritas tertinggi Anda, Anda harus mempertimbangkan untuk menyesuaikan waktu tunggu permintaan default dan pengaturan coba lagi untuk SDK Anda agar dapat melacak dengan cermat latensi umum untuk permintaan yang berhasil diukur oleh klien Anda. Permintaan yang memakan waktu jauh lebih lama dari biasanya kecil kemungkinannya untuk berhasil - jika Anda gagal cepat (fail fast) dan membuat permintaan baru, kemungkinan besar permintaan tersebut akan mengambil jalur yang berbeda dan mungkin berhasil dengan cepat. Ingatlah bahwa ada kerugian jika bersikap terlalu agresif dalam situasi ini. Diskusi bermanfaat mengenai topik ini dapat ditemukan di [Menyetel pengaturan permintaan AWS Java SDK HTTP untuk aplikasi Amazon DynamoDB yang sadar latensi](#).
- Kurangi jarak antara klien dan titik akhir DynamoDB: Jika Anda memiliki pengguna yang tersebar secara global, pertimbangkan untuk menggunakan [Tabel global - Replikasi multi-Wilayah untuk DynamoDB](#). Dengan tabel global, Anda dapat menentukan Wilayah AWS tempat Anda ingin tabel tersedia. Membaca data dari replika tabel global lokal dapat mengurangi latensi bagi pengguna Anda secara signifikan. Selain itu, pertimbangkan untuk menggunakan [titik akhir gateway](#) DynamoDB untuk menjaga lalu lintas klien Anda tetap dalam VPC Anda.
- Gunakan caching: Jika lalu lintas Anda banyak dibaca, pertimbangkan untuk menggunakan layanan caching, seperti [Akselerasi dalam memori dengan DynamoDB Accelerator \(DAX\)](#). DAX

adalah cache dalam memori yang terkelola sepenuhnya dan memiliki ketersediaan tinggi untuk DynamoDB yang memberikan peningkatan performa hingga 10x, dari milidetik hingga mikrodetik, bahkan pada jutaan permintaan per detik.

- Gunakan kembali koneksi: Permintaan DynamoDB dibuat melalui sesi yang diautentikasi yang default ke HTTPS. Memulai koneksi membutuhkan waktu sehingga latensi permintaan pertama lebih tinggi dari biasanya. Permintaan melalui koneksi yang sudah diinisialisasi menghasilkan latensi rendah DynamoDB yang konsisten. Karena alasan ini, Anda mungkin ingin membuat permintaan `GetItem` "tetap hidup" setiap 30 detik jika tidak ada permintaan lain yang dibuat, untuk menghindari latensi dalam pembuatan koneksi baru.
- Gunakan pembacaan yang pada akhirnya konsisten: Jika aplikasi Anda tidak memerlukan bacaan sangat konsisten, pertimbangkan untuk menggunakan bacaan akhir konsisten secara default. Pembacaan akhir konsisten berbiaya lebih rendah dan juga kecil kemungkinannya mengalami peningkatan latensi sementara. Lihat informasi yang lebih lengkap di [Konsistensi baca](#).

Memecahkan masalah pembatasan di Amazon DynamoDB

Dalam arsitektur berorientasi layanan dan sistem terdistribusi, membatasi tingkat di mana panggilan API diproses oleh berbagai komponen layanan disebut throttling. Ini membantu menghaluskan lonjakan, mengontrol ketidakcocokan dalam throughput komponen, dan memungkinkan pemulihan yang lebih dapat diprediksi saat terjadi peristiwa operasional yang tidak terduga. DynamoDB dirancang untuk jenis arsitektur ini, dan klien DynamoDB memiliki percobaan ulang yang dibangun untuk permintaan yang dibatasi. Beberapa tingkat pembatasan belum tentu menjadi masalah bagi aplikasi Anda, tetapi pelambatan terus-menerus dari bagian yang sensitif terhadap latensi dari alur kerja data Anda dapat berdampak negatif pada pengalaman pengguna dan mengurangi efisiensi keseluruhan sistem.

Jika operasi baca atau tulis di tabel DynamoDB Anda sedang dibatasi (atau jika Anda mendapatkan `ProvisionedThroughputExceededException` ketika Anda melakukan operasi), pertimbangkan untuk menggunakan strategi berikut untuk membantu menyelesaikan masalah.

Topik

- [Pastikan tabel mode yang disediakan memiliki kapasitas yang memadai](#)
- [Pertimbangkan untuk beralih ke mode on-demand](#)
- [Mendistribusikan operasi baca dan tulis secara merata di antara kunci partisi](#)
- [Tingkatkan kuota throughput baca atau tulis tingkat tabel](#)

Pastikan tabel mode yang disediakan memiliki kapasitas yang memadai

DynamoDB melaporkan metrik tingkat menit ke CloudWatch. Metrik dihitung sebagai jumlah selama satu menit dan kemudian dirata-ratakan. Namun, batas tarif DynamoDB diterapkan per detik.

Misalnya, jika Anda menyediakan 60 unit kapasitas tulis untuk tabel Anda, Anda dapat melakukan 3600 unit tulis dalam satu menit, tetapi mencoba melakukan lebih dari 60 unit tulis dalam satu detik dapat mengakibatkan beberapa permintaan dibatasi.

Untuk mengatasi masalah ini, pastikan meja Anda memiliki kapasitas yang cukup untuk melayani lalu lintas Anda dan coba lagi permintaan yang dibatasi menggunakan [backoff eksponensial](#). Jika Anda menggunakan AWS SDK, maka logika ini diimplementasikan secara default. Untuk alasan ini, throttle mungkin terjadi, dan dicoba ulang berhasil pada percobaan ulang kedua atau ketiga, sementara tidak pernah menyebabkan kesalahan dalam kode Anda. Untuk memahami throttle saat terjadi, setiap flavor AWS-SDK memiliki cara untuk mendeteksi throttle. Misalnya, di Python periksa nomor di `kembaliResponseMetadata.RetryAttempts` bidang. Jika ini lebih besar dari nol, itu menunjukkan ada throttle. Strategi besar setelah mendeteksi ini adalah untuk log kunci partisi terlibat, karena mungkin pola kunci panas yang akan sulit untuk mendiagnosa dengan CloudWatch sendirian.

Juga, pertimbangkan untuk menggunakan [penskalaan otomatis](#) pada tabel yang disediakan untuk secara otomatis menyesuaikan kapasitas yang disediakan berdasarkan beban kerja. Anda juga harus mempertimbangkan memperbarui kebijakan penskalaan untuk memiliki minimum yang lebih tinggi atau pemanfaatan target yang lebih rendah, dan pastikan pengaturan maksimum tidak diatur terlalu rendah.

Pertimbangkan untuk beralih ke mode on-demand

Penskalaan otomatis Amazon DynamoDB menggunakan [Penskalaan Otomatis Aplikasi](#) layanan untuk secara dinamis menyesuaikan kapasitas throughput yang disediakan sebagai respons terhadap pola lalu lintas aktual. Penskalaan Otomatis Aplikasi memulai penskalaan hanya jika dua titik data berturut-turut untuk unit kapasitas yang dikonsumsi melebihi nilai pemanfaatan target yang dikonfigurasi dalam rentang satu menit. Aplikasi Auto Scaling secara otomatis menskalakan kapasitas yang disediakan hanya jika kapasitas yang dikonsumsi lebih tinggi dari pemanfaatan target selama dua menit berturut-turut. Juga, acara skala-down dimulai ketika 15 titik data berturut-turut untuk kapasitas yang dikonsumsi di CloudWatch lebih rendah dari pemanfaatan target.

Setelah Application Auto Scaling dimulai, sebuah `UpdateTable` Panggilan API dipanggil untuk memperbarui kapasitas yang disediakan untuk tabel atau indeks DynamoDB Anda. Penskalaan Otomatis Aplikasi memerlukan titik data berturut-turut dengan nilai pemanfaatan target yang lebih

tinggi untuk meningkatkan kapasitas yang disediakan dari tabel DynamoDB. Selama periode ini, setiap permintaan yang melebihi kapasitas yang disediakan dari tabel akan dibatasi. Karena itu, pertimbangkan untuk menggunakan [Mode sesuai permintaan](#) untuk pola lalu lintas yang sangat tidak terduga (meledak). Untuk informasi selengkapnya, lihat [Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB](#).

Mendistribusikan operasi baca dan tulis secara merata di antara kunci partisi

Di DynamoDB, kunci partisi dengan kardinalitas rendah dapat menghasilkan banyak permintaan yang menargetkan hanya beberapa partisi dan menghasilkan partisi panas. Partisi panas dapat menyebabkan pelambatan jika batas partisi 3000 RCU dan 1000 WCU per detik terlampaui.

Untuk menemukan item yang paling diakses dan dicekik di meja Anda, gunakan [AmazonCloudWatchWawasan Kontributor](#). AmazonCloudWatchContributor Insights adalah alat diagnostik yang menyediakan tampilan ringkas tren lalu lintas tabel DynamoDB Anda dan membantu Anda mengidentifikasi kunci partisi yang paling sering diakses. Dengan alat ini, Anda dapat terus memantau grafik untuk pola akses item tabel Anda. Partisi panas dapat menurunkan kinerja keseluruhan meja Anda. Untuk menghindari kinerja yang buruk ini, distribusikan operasi baca dan tulis serata mungkin dengan memilih kunci kardinalitas tinggi. Untuk informasi lebih lanjut, lihat [Merancang kunci partisi untuk mendistribusikan beban kerja Anda](#) dan [Memilih kunci partisi DynamoDB yang tepat](#).

Note

Menggunakan AmazonCloudWatchAlat Contributor Insights untuk DynamoDB dikenakan biaya tambahan. Untuk informasi selengkapnya, lihat [CloudWatch wawasan kontributor untuk penagihan DynamoDB](#).

Tingkatkan kuota throughput baca atau tulis tingkat tabel

Tingkat meja [membaca throughput dan kuota throughput tulis tingkat tabel](#) berlaku di tingkat akun di Wilayah mana pun. Kuota ini berlaku untuk tabel dengan mode kapasitas yang disediakan dan mode kapasitas sesuai permintaan. Secara default, kuota throughput yang ditempatkan di meja Anda adalah 40.000 unit permintaan baca dan 40.000 unit permintaan tulis. Jika lalu lintas ke tabel Anda melebihi kuota ini, maka tabel mungkin dibatasi. Untuk informasi lebih lanjut tentang cara mencegah hal ini terjadi, lihat [Memantau Amazon DynamoDB untuk kesadaran operasional](#).

Untuk mengatasi masalah ini, gunakan [Konsol Kuota Layanan](#) untuk meningkatkan kuota throughput baca atau tulis tingkat tabel untuk akun Anda.

Lampiran DynamoDB

Topik

- [Memecahkan masalah pembentukan koneksi SSL/TLS](#)
- [Contoh tabel dan data](#)
- [Membuat contoh tabel dan mengunggah data](#)
- [DynamoDB contoh aplikasi menggunakan AWS SDK for Python \(Boto\): Tic-tac-toe](#)
- [Mengekspor dan mengimpor data DynamoDB menggunakan AWS Data Pipeline](#)
- [Backend Penyimpanan Amazon DynamoDB untuk Titan](#)
- [Disimpan kata-kata di DynamoDB](#)
- [Parameter bersyarat lama](#)
- [Versi API tingkat rendah sebelumnya \(2011-12-05\)](#)
- [AWS Contoh SDK for Java 1.x](#)

Memecahkan masalah pembentukan koneksi SSL/TLS

Amazon DynamoDB sedang dalam proses memindahkan titik akhir ke sertifikat aman yang ditandatangani oleh Otoritas Sertifikat Amazon Trust Services (ATS), bukan Otoritas Sertifikat pihak ketiga. Pada bulan Desember 2017, kami meluncurkan Wilayah EU-WEST-3 (Paris) dengan sertifikat aman yang dikeluarkan oleh Amazon Trust Services. Semua wilayah baru yang diluncurkan setelah Desember 2017 memiliki titik akhir dengan sertifikat yang dikeluarkan oleh Amazon Trust Services. Panduan ini menunjukkan cara untuk memvalidasi dan memecahkan masalah koneksi SSL/TLS.

Menguji aplikasi atau layanan Anda

Sebagian besar AWS SDK dan Command Line Interfaces (CLI) mendukung Amazon Trust Services Certificate Authority. Jika Anda menggunakan versi AWS SDK untuk Python atau CLI yang dirilis sebelum 29 Oktober 2013, Anda harus meng-upgrade. .NET, Java, PHP, Go JavaScript, dan C++ SDK dan CLI tidak menggabungkan sertifikat apa pun, sertifikat mereka berasal dari sistem operasi yang mendasarinya. SDK Ruby telah menyertakan setidaknya salah satu dari CA yang diperlukan sejak 10 Juni 2015. Sebelum tanggal tersebut, SDK Ruby V2 tidak menyertakan sertifikat. Jika Anda menggunakan versi AWS SDK yang tidak didukung, kustom, atau modifikasi, atau jika Anda menggunakan toko kepercayaan khusus, Anda mungkin tidak memiliki dukungan yang diperlukan untuk Amazon Trust Services Certificate Authority.

Untuk memvalidasi akses ke titik akhir DynamoDB, Anda perlu mengembangkan pengujian yang mengakses API DynamoDB atau API DynamoDB Streams di wilayah EU-WEST-3 dan memvalidasi bahwa handshake TLS berhasil. Titik akhir tertentu yang akan Anda perlukan untuk mengakses pengujian tersebut adalah:

- DynamoDB: <https://dynamodb.eu-west-3.amazonaws.com>
- DynamoDB Streams: <https://streams.dynamodb.eu-west-3.amazonaws.com>

Jika aplikasi Anda tidak mendukung Otoritas Sertifikat Amazon Trust Services, Anda akan melihat salah satu kegagalan berikut:

- Kesalahan Negosiasi SSL/TLS
- Penundaan lama sebelum perangkat lunak Anda menerima kesalahan yang menunjukkan kegagalan negosiasi SSL/TLS. Waktu tunda bergantung pada strategi coba lagi dan konfigurasi waktu habis klien Anda.

Menguji browser klien Anda

Untuk memverifikasi bahwa browser Anda dapat terhubung ke Amazon DynamoDB, buka URL berikut: <https://dynamodb.eu-west-3.amazonaws.com>. Jika pengujian berhasil, Anda akan melihat pesan seperti ini:

```
healthy: dynamodb.eu-west-3.amazonaws.com
```

Jika pengujian tidak berhasil, kesalahan yang mirip dengan ini akan ditampilkan: <https://untrusted-root.badssl.com/>.

Memperbarui klien aplikasi perangkat lunak Anda

Aplikasi yang mengakses titik akhir DynamoDB atau API DynamoDB Streams (baik melalui browser atau secara terprogram) perlu memperbarui daftar CA tepercaya pada mesin klien jika aplikasi tersebut tidak mendukung salah satu CA berikut:

- Amazon Root CA 1
- Starfield Services Root Certificate Authority - G2
- Starfield Class 2 Certification Authority

Jika klien sudah mempercayai SALAH SATU dari tiga CA di atas, klien akan mempercayai sertifikat yang digunakan oleh DynamoDB dan tidak ada tindakan yang diperlukan. Namun, jika klien Anda belum mempercayai salah satu CA di atas, koneksi HTTPS ke API DynamoDB atau DynamoDB Streams akan gagal. Untuk informasi lebih lanjut, silakan kunjungi posting blog ini: <https://aws.amazon.com/blogs/security/how-to-prepare-for-aws-move-to-its-own-certificate-authority/>.

Memperbarui browser klien Anda

Anda dapat memperbarui paketan sertifikat di browser cukup dengan memperbarui browser Anda. Petunjuk tentang browser yang paling umum dapat ditemukan di situs web browser tersebut:

- 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-versi-#ie=lainnya>

Memperbarui paketan sertifikat Anda secara manual

Jika Anda tidak dapat mengakses API DynamoDB atau API DynamoDB Streams, Anda perlu memperbarui paketan sertifikat Anda. Untuk melakukan hal ini, Anda perlu mengimpor setidaknya salah satu dari CA yang dibutuhkan. Anda dapat menemukannya di <https://www.amazontrust.com/repository/>.

Sistem operasi dan bahasa pemrograman berikut mendukung sertifikat Amazon Trust Services:

- Versi Microsoft Windows yang sudah menginstal pembaruan Januari 2005 atau yang lebih baru, Windows Vista, Windows 7, Windows Server 2008, dan versi yang lebih baru.
- MacOS X 10.4 dengan Java untuk MacOS X 10.4 Release 5, MacOS X 10.5 dan versi yang lebih baru.
- Red Hat Enterprise Linux 5 (Maret 2007), Linux 6, dan Linux 7 serta CentOS 5, CentOS 6, dan CentOS 7
- Ubuntu 8.10
- Debian 5.0
- Amazon Linux (semua versi)

- Java 1.4.2_12, Java 5 update 2, dan semua versi yang lebih baru, termasuk Java 6, Java 7, dan Java 8

Jika Anda masih tidak dapat terhubung, silakan berkonsultasi dengan dokumentasi perangkat lunak Anda, Vendor OS atau hubungi AWS Support <https://aws.amazon.com/support> untuk bantuan lebih lanjut.

Contoh tabel dan data

Panduan Developer Amazon DynamoDB menggunakan tabel sampel untuk menggambarkan berbagai aspek DynamoDB.

Nama tabel	Kunci Primer
ProductCatalog	Kunci utama sederhana: <ul style="list-style-type: none"> • Id (Angka)
Forum	Kunci utama sederhana: <ul style="list-style-type: none"> • Name (String)
Benang	Kunci utama komposit: <ul style="list-style-type: none"> • ForumName (String) • Subject (String)
Balas	Kunci utama komposit: <ul style="list-style-type: none"> • Id (String) • ReplyDateTime (String)

Tabel Reply memiliki indeks sekunder global bernama PostedBy-Message-Index. Indeks ini akan memfasilitasi kueri pada dua atribut non-kunci dari tabel Balasan.

Nama Nama Nama Nama indeks Nama indeks	Kunci Primer
PostedBy-Pesan-Indeks	Kunci utama komposit:

Nama Nama Nama Nama indeks Nama indeks	Kunci Primer
	<ul style="list-style-type: none"> • PostedBy (String) • Message (String)

Untuk informasi selengkapnya tentang tabel ini, lihat [Langkah 2: Buat tabel](#) dan [Langkah 2: Tulis data ke tabel menggunakan konsol atau AWS CLI](#).

File File File File File File File File File

Topik

- [ProductCatalogdata sampel](#)
- [Data sampel forum](#)
- [Data sampel benang](#)
- [Balas data sampel](#)

Bagian berikut menunjukkan file data sampel yang digunakan untuk memuat tabel ProductCatalog, Forum, Thread dan Balas.

Setiap file data berisi beberapa elemen PutRequest, masing-masing berisi satu item. Elemen PutRequest ini digunakan sebagai input untuk operasi BatchWriteItem, menggunakan AWS Command Line Interface (AWS CLI).

Untuk informasi selengkapnya, lihat [Langkah 2: Tulis data ke tabel menggunakan konsol atau AWS CLI](#) di [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#).

ProductCatalogdata sampel

```
{
  "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"
    }
}
}
}
}
]
```

Data sampel forum

```
{
  "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"}
        }
    }
}
]
}

```

Data sampel benang

```

{
  "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"
          }
        ]
      }
    }
  ]
}

```

Balas data sampel

```

{
  "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"
    }
  }
}
]
```

Membuat contoh tabel dan mengunggah data

Topik

- [Membuat tabel contoh dan mengunggah data menggunakan AWS SDK for Java](#)
- [Membuat tabel contoh dan mengunggah data menggunakan AWS SDK for .NET](#)

Di [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#), buatlah tabel terlebih dahulu menggunakan konsol DynamoDB lalu gunakan AWS CLI untuk menambahkan data ke tabel. Lampiran ini menyediakan kode untuk membuat tabel dan menambahkan data secara terprogram.

Membuat tabel contoh dan mengunggah data menggunakan AWS SDK for Java

Contoh kode Java berikut ini membuat tabel dan mengunggah data ke tabel. Struktur tabel yang dihasilkan dan data ditampilkan dalam [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#). Untuk step-by-step instruksi untuk menjalankan kode ini menggunakan Eclipse, lihat [Contoh kode Java](#)

```
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 CreateTableLoadData {

    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());
    }
}
}
```

Membuat tabel contoh dan mengunggah data menggunakan AWS SDK for .NET

Contoh kode C# berikut ini membuat tabel dan mengunggah data ke tabel. Struktur tabel yang dihasilkan dan data ditampilkan dalam [Membuat tabel dan memuat data untuk contoh kode di DynamoDB](#). Untuk step-by-step petunjuk untuk menjalankan kode ini di Visual Studio, lihat [Contoh kode .NET](#).

```
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 CreateTableLoadData
```

```
{
    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);
    }
}
}
```

DynamoDB contoh aplikasi menggunakan AWS SDK for Python (Boto): Tic-tac-toe

Topik

- [Langkah 1: men-deploy dan menguji secara lokal](#)
- [Langkah 2: Periksa model data dan detail implementasi](#)
- [Langkah 3: men-deploy dalam Produksi menggunakan layanan DynamoDB](#)
- [Langkah 4: Bersihkan Sumber Daya](#)

Game Tic-Tac-Toe adalah contoh aplikasi web yang dibangun di Amazon DynamoDB. Aplikasi ini menggunakan AWS SDK for Python (Boto) untuk membuat panggilan DynamoDB yang diperlukan untuk menyimpan data game dalam tabel DynamoDB, dan Flask kerangka kerja web Python untuk menggambarkan Developeran end-to-end aplikasi di DynamoDB, termasuk cara pemodelan data. Hal ini juga menunjukkan praktik terbaik terkait pemodelan data di DynamoDB, termasuk tabel yang Anda buat untuk aplikasi game, kunci utama yang Anda tentukan, indeks tambahan yang Anda butuhkan berbasiskan kebutuhan kueri Anda, dan penggunaan atribut nilai bersambung.

Anda memainkan aplikasi Tic-Tac-Toe di web dengan langkah berikut:

1. Anda masuk ke halaman beranda aplikasi.
2. Anda kemudian mengundang pengguna lain untuk memainkan game sebagai lawan Anda.

Sampai pengguna lain menerima undangan Anda, status game tetap sebagai PENDING. Setelah lawan menerima undangan, status game berubah menjadi IN_PROGRESS.

3. Game dimulai setelah lawan masuk dan menerima undangan.
4. Aplikasi menyimpan semua gerakan dalam game dan informasi status dalam tabel DynamoDB.
5. Game berakhir dengan kemenangan atau imbang, yang menetapkan status game menjadi FINISHED.

Latihan pembuatan end-to-end aplikasi dijelaskan dalam langkah-langkah di bawah ini:

- [Langkah 1: men-deploy dan menguji secara lokal](#) Pada bagian ini, Anda mengunduh, men-deploy, dan menguji aplikasi pada komputer lokal Anda. Anda akan membuat tabel yang diperlukan dalam versi unduhan dari DynamoDB.
- [Langkah 2: Periksa model data dan detail implementasi](#) Bagian ini pertama-tama menjelaskan secara rinci model data, termasuk indeks dan penggunaan atribut nilai bersambung. Kemudian bagian tersebut menjelaskan cara aplikasi bekerja.
- [Langkah 3: men-deploy dalam Produksi menggunakan layanan DynamoDB](#) Bagian ini berfokus pada pertimbangan deployment dalam produksi. Pada langkah ini, Anda membuat tabel

menggunakan layanan Amazon DynamoDB dan men-deploy aplikasi menggunakan AWS Elastic Beanstalk. Bila Anda memiliki aplikasi dalam produksi, Anda juga memberikan izin yang sesuai sehingga aplikasi dapat mengakses tabel DynamoDB. Petunjuk di bagian ini memandu Anda melalui deployment end-to-end produksi.

- [Langkah 4: Bersihkan Sumber Daya](#) Bagian ini menyoroti area yang tidak tercakup dalam contoh ini. Bagian ini juga menyediakan langkah-langkah bagi Anda untuk menghapus sumber daya AWS yang Anda buat dalam langkah-langkah sebelumnya sehingga Anda menghindari timbulnya biaya apapun.

Langkah 1: men-deploy dan menguji secara lokal

Topik

- [1.1: Unduh dan instal paket yang diperlukan](#)
- [1.2: Uji Aplikasi Game](#)

Dalam langkah ini Anda mengunduh, men-deploy, dan menguji aplikasi game Tic-Tac-Toe pada komputer lokal Anda. Alih-alih menggunakan layanan web Amazon DynamoDB, Anda akan mengunduh DynamoDB ke komputer Anda, dan membuat tabel yang dibutuhkan di sana.

1.1: Unduh dan instal paket yang diperlukan

Anda akan memerlukan hal-hal berikut untuk menguji aplikasi ini secara lokal:

- Python
- Flask (kerangka kerja untuk Python)
- AWS SDK for Python (Boto)
- DynamoDB berjalan di komputer Anda
- Git

Untuk mendapatkan alat ini, lakukan hal berikut:

1. Instal Python. Untuk step-by-step petunjuknya, lihat [Download Python](#).

Aplikasi Tic-Tac-Toe telah diuji menggunakan Python versi 2.7.

2. Instal Flask dan AWS SDK for Python (Boto) menggunakan Python Package Installer (PIP):

- Instal PIP.

Untuk instruksi selengkapnya, lihat [Instal PIP](#). Pada halaman instalasi, pilih tautan `get-pip.py`, kemudian simpan file. Kemudian buka terminal perintah sebagai administrator, dan masukkan yang perintah selanjutnya pada prompt perintah.

```
python.exe get-pip.py
```

Di Linux, Anda tidak menentukan ekstensi `.exe`. Anda hanya menentukan `python get-pip.py`.

- Menggunakan PIP, instal paket Flask dan Boto menggunakan kode berikut.

```
pip install Flask
pip install boto
pip install configparser
```

3. Unduh dan jalankan DynamoDB di komputer Anda. Untuk petunjuk tentang cara melakukannya, lihat [Menyiapkan DynamoDB lokal \(versi yang dapat diunduh\)](#).
4. Unduh aplikasi Tic-Tac-Toe:
 - a. Instal Git. Untuk instruksi, lihat [Unduhan git](#).
 - b. Jalankan kode berikut untuk mengunduh aplikasi.

```
git clone https://github.com/awslabs/dynamodb-tictactoe-example-app.git
```

1.2: Uji Aplikasi Game

Untuk menguji aplikasi Tic-Tac-Toe, Anda perlu menjalankan DynamoDB secara lokal di komputer Anda.

Untuk menjalankan tic-tac-toe aplikasi

1. Mulai DynamoDB.
2. Mulai server web untuk aplikasi Tic-Tac-Toe.

Untuk melakukannya, buka terminal perintah, arahkan ke folder tempat Anda mengunduh aplikasi Tic-Tac-Toe, dan jalankan aplikasi secara lokal menggunakan kode berikut.

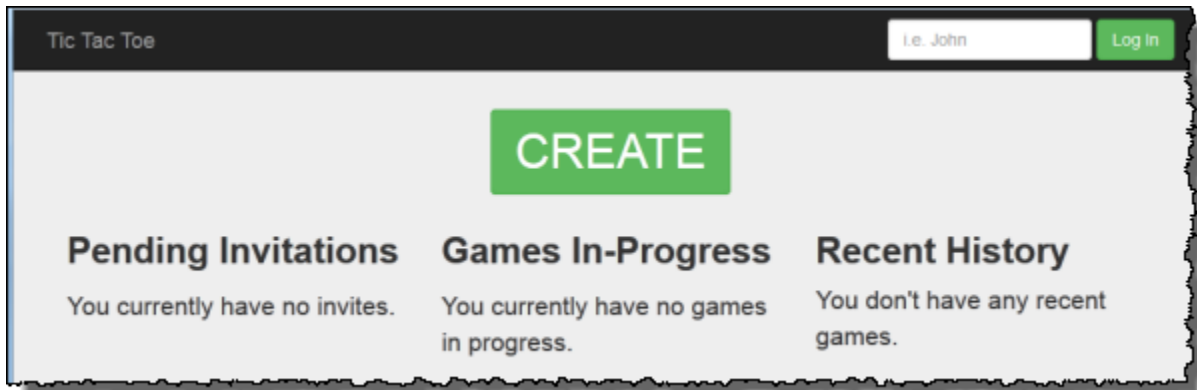
```
python.exe application.py --mode local --serverPort 5000 --port 8000
```

Di Linux, Anda tidak menentukan ekstensi `.exe`.

3. Buka peramban web Anda, dan masukkan perintah berikut ini.

```
http://localhost:5000/
```

peramban menampilkan beranda.

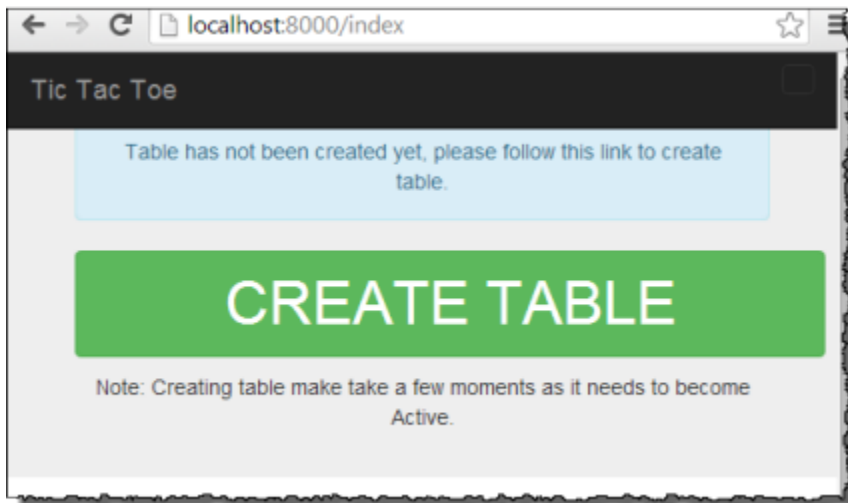


4. Masukkan **user1** di kotak Log in untuk masuk sebagai user1.

Note

Aplikasi contoh ini tidak melakukan autentikasi pengguna. ID pengguna hanya digunakan untuk mengidentifikasi pemain. Jika dua pemain masuk dengan alias yang sama, aplikasi bekerja seolah-olah Anda bermain di dua peramban yang berbeda.

5. Jika ini adalah pertama kalinya Anda bermain game, halaman meminta Anda untuk membuat tabel yang diperlukan (Games) di DynamoDB. Pilih BUAT TABEL.



6. Pilih CREATE untuk membuat tic-tac-toe game pertama.
7. Masukkan **user2** di kotak Pilih Lawan, dan pilih Buat Game!



Hal ini akan membuat game dengan cara menambahkan item dalam tabel Games. Ini menetapkan status game menjadi PENDING.

8. Buka jendela peramban lain, dan masukkan perintah berikut.

```
http://localhost:5000/
```

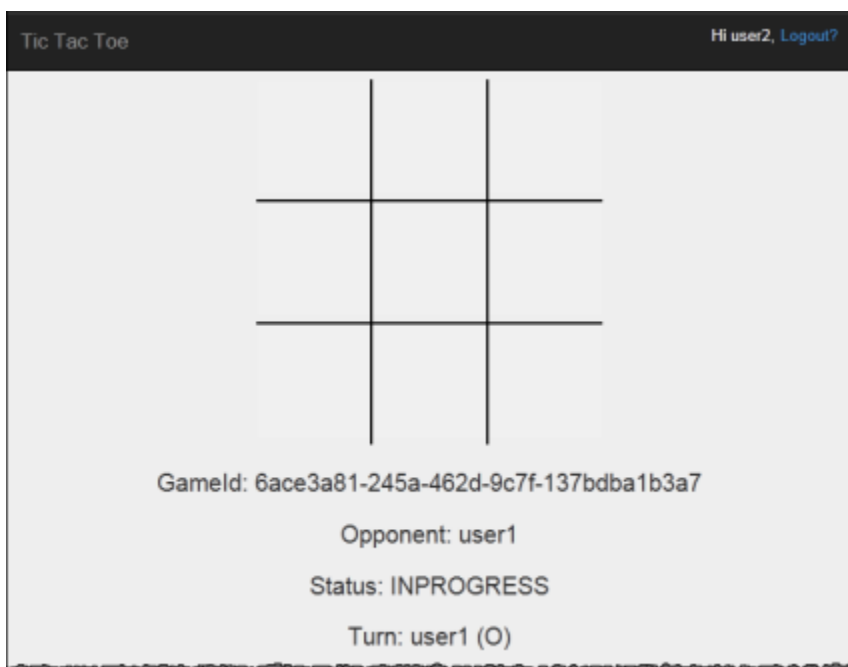
peramban mengirimkan informasi melalui cookie, sehingga Anda harus menggunakan mode penyamaran atau penjelajahan privat sehingga cookie Anda tidak ikut terbawa.

9. Masuk sebagai user2.

Halaman menampilkan undangan tertunda dari user1.



10. Pilih terima untuk menerima undangan.



Halaman game menampilkan tic-tac-toe kotak kosong. Halaman ini juga menunjukkan informasi game yang relevan seperti ID game, giliran pemain selanjutnya, dan status game.

11. Mainkan game.

Untuk setiap pengguna yang sedang melakukan gilirannya, layanan web mengirimkan permintaan ke DynamoDB untuk memperbarui item game secara kondisional di tabel Games. Sebagai contoh, kondisi memastikan bahwa langkah tersebut valid, kotak yang dipilih pengguna tersedia, dan bahwa itu adalah pergantian pengguna atau pemain selanjutnya yang membuat langkah. Untuk langkah yang valid, operasi update menambahkan atribut baru yang sesuai dengan pilihan di forum. Operasi

pembaruan juga menetapkan nilai atribut yang ada untuk pengguna yang dapat membuat langkah berikutnya.

Pada halaman game, aplikasi membuat JavaScript panggilan asynchronous setiap detik, hingga 5 menit, untuk memeriksa apakah status game di DynamoDB telah berubah. Jika sudah, aplikasi memperbarui halaman dengan informasi baru. Setelah 5 menit, aplikasi berhenti membuat permintaan, dan Anda perlu menyegarkan halaman untuk mendapatkan informasi terbaru.

Langkah 2: Periksa model data dan detail implementasi

Topik

- [2.1: Model data basic](#)
- [2.2: Aplikasi dalam tindakan \(kode panduan\)](#)

2.1: Model data basic

Aplikasi contoh ini menyoroti konsep model data DynamoDB berikut:

- Tabel Dalam DynamoDB, tabel adalah kumpulan item (yaitu, riwayat), dan setiap item adalah kumpulan pasangan nama-nilai yang disebut atribut.

Dalam contoh Tic-Tac-Toe ini, aplikasi menyimpan semua data game dalam sebuah tabel, Games. Aplikasi ini menciptakan satu item dalam tabel per game dan menyimpan semua data game sebagai atribut. tic-tac-toe Game dapat memiliki hingga sembilan gerakan. Karena tabel DynamoDB tidak memiliki skema dalam kasus di mana hanya kunci utama adalah atribut yang diperlukan, aplikasi dapat menyimpan berbagai jumlah atribut per item game.

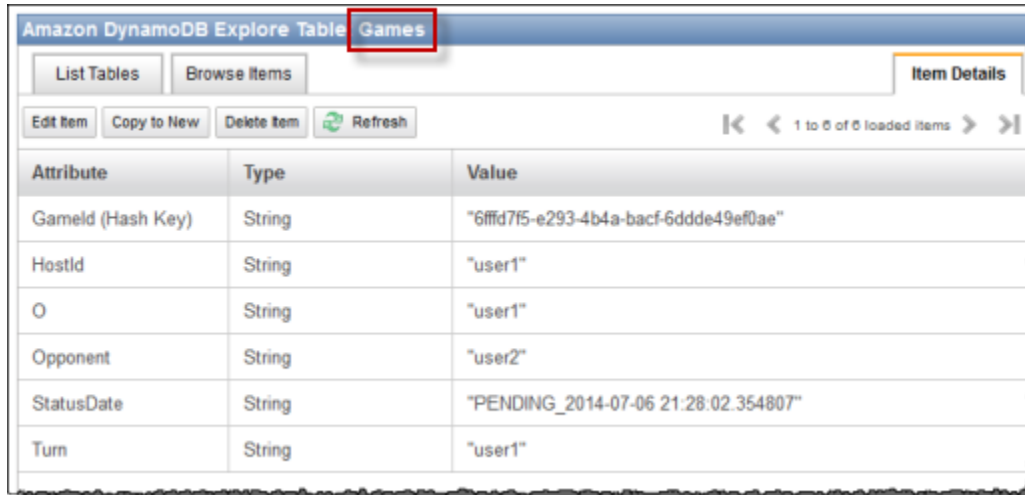
Tabel Games memiliki kunci primer sederhana yang terbuat dari satu atribut, GameId, tipe string. Aplikasi ini memberikan ID unik untuk setiap game. Untuk informasi selengkapnya tentang kunci utama DynamoDB, lihat [Kunci primer](#).

Ketika pengguna memulai tic-tac-toe game dengan mengundang pengguna lain untuk ikut bermain, aplikasi menciptakan item baru dalam Games tabel dengan atribut yang menyimpan metadata game, seperti berikut:

- HostId, pengguna yang memulai game.
- Opponent, pengguna yang diundang untuk ikut bermain.
- Pengguna yang gilirannya untuk bermain. Pengguna yang memulai langkah awal di game.

- Pengguna yang menggunakan simbol O di forum. Pengguna yang memulai game menggunakan simbol O.

Di samping itu, aplikasi membuat atribut bersambung `StatusDate`, menandai status game awal sebagai `PENDING`. Screenshot berikut menunjukkan contoh item seperti yang muncul di konsol DynamoDB:



Attribute	Type	Value
Gamelid (Hash Key)	String	"6fffd7f5-e293-4b4a-bacf-6ddde49ef0ae"
HostId	String	"user1"
O	String	"user1"
Opponent	String	"user2"
StatusDate	String	"PENDING_2014-07-06 21:28:02.354807"
Turn	String	"user1"

Seiring game berlangsung, aplikasi menambahkan satu atribut ke tabel untuk setiap gerakan game. Nama atribut adalah posisi forum, misalnya `TopLeft` atau `BottomRight`. Sebagai contoh, sebuah langkah mungkin memiliki atribut `TopLeft` dengan nilai `O`, sebuah atribut `TopRight` dengan nilai `O`, dan atribut `BottomRight` dengan nilai `X`. Nilai atribut adalah baik `O` atau `X`, tergantung pada pengguna mana yang bergerak. Contohnya, pertimbangkan forum berikut ini.



You Tie		
O	X	O
O	O	X
X	O	X

Gamelid: 5ca60639-bef9-4c03-83e9-bb7abe4debca
 Opponent: user2
 Status: FINISHED
 Turn: N/A

- Atribut nilai bersambung adalah atribut `StatusDate` yang menggambarkan atribut nilai bersambung. Dalam pendekatan ini, alih-alih menciptakan atribut terpisah untuk menyimpan status

game (PENDING, IN_PROGRESS, dan FINISHED) dan tanggal (ketika langkah terakhir dibuat), Anda menggabungkan mereka sebagai atribut tunggal, misalnya IN_PROGRESS_2014-04-30 10:20:32.

Aplikasi kemudian menggunakan atribut StatusDate dalam menciptakan indeks sekunder dengan menentukan StatusDate sebagai kunci untuk indeks. Manfaat menggunakan atribut StatusDate nilai bersambung selanjutnya diilustrasikan dalam indeks yang akan dibahas selanjutnya.

- Indeks sekunder global Anda dapat menggunakan kunci utama tabel, GameId, untuk secara efisien meng-kueri tabel untuk menemukan item game. Untuk kueri tabel pada atribut selain atribut kunci utama, DynamoDB mendukung pembuatan indeks sekunder. Dalam aplikasi contoh ini, Anda membangun dua indeks sekunder berikut:

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
This table has no local secondary indexes.										
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 -indeks. Indeks ini memiliki HostId sebagai kunci partisi dan StatusDate sebagai kunci lain. Anda dapat menggunakan indeks ini untuk kueri pada HostId, misalnya untuk menemukan game yang diselenggarakan oleh pengguna tertentu.
- OpponentId-StatusDate -indeks. Indeks ini memiliki OpponentId sebagai kunci partisi dan StatusDate sebagai kunci lain. Anda dapat menggunakan indeks ini untuk kueri pada Opponent, misalnya untuk menemukan game dimana pengguna tertentu adalah lawan.

Indeks ini disebut indeks sekunder global karena kunci partisi dalam indeks ini tidak sama dengan kunci partisi (GameId), digunakan dalam kunci primer dari tabel.

Perhatikan bahwa kedua indeks menentukan StatusDate sebagai kunci lain. Melakukan hal di bawah ini memungkinkan hal berikut:

- Anda dapat kueri menggunakan operator perbandingan BEGINS_WITH. Misalnya, Anda dapat menemukan semua game dengan atribut IN_PROGRESS yang diselenggarakan oleh pengguna

tertentu. Dalam hal ini, operator `BEGINS_WITH` memeriksa nilai `StatusDate` yang dimulai dengan `IN_PROGRESS`.

- DynamoDB menyimpan item dalam indeks dengan urutan yang diurutkan, dengan nilai kunci. Jadi jika semua awalan status adalah sama (sebagai contoh, `IN_PROGRESS`), format ISO yang digunakan untuk bagian tanggal akan memiliki item yang diurutkan dari yang paling lawas hingga yang paling baru. Pendekatan ini memungkinkan permintaan tertentu untuk dilakukan secara efisien, misalnya berikut ini:
 - Ambil hingga 10 dari game `IN_PROGRESS` yang diselenggarakan oleh pengguna yang login. Untuk kueri ini, Anda menentukan indeks `HostId-StatusDate-index`.
 - Ambil hingga 10 dari game `IN_PROGRESS` di mana pengguna login adalah lawan. Untuk kueri ini, Anda menentukan indeks `OpponentId-StatusDate-index`.

Untuk informasi selengkapnya tentang indeks sekunder global, lihat [Meningkatkan akses data dengan indeks sekunder](#).

2.2: Aplikasi dalam tindakan (kode panduan)

Aplikasi ini memiliki dua halaman utama:

- Halaman Beranda Halaman ini menyediakan pengguna login sederhana, BUAT untuk membuat tic-tac-toe game baru, daftar game yang sedang berlangsung, BUAT untuk membuat game baru, daftar game yang sedang berlangsung, dan undangan game yang sedang berlangsung.

Halaman beranda tidak melakukan penyegaran secara otomatis; Anda harus menyegarkan halaman untuk menyegarkan daftar.

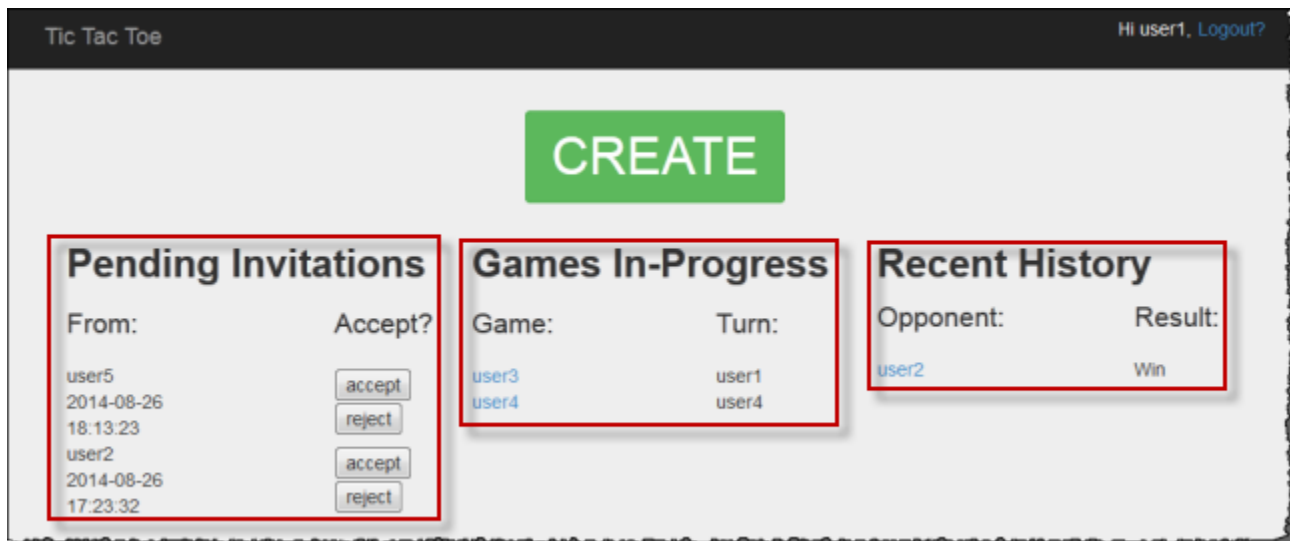
- Halaman game Halaman game Halaman ini menampilkan tic-tac-toe grid tempat pengguna bermain.

Aplikasi memperbarui halaman game secara otomatis setiap detik. JavaScript Di peramban Anda memanggil server web Python setiap detik untuk meng-kueri tabel Games apakah item game dalam tabel telah berubah. Jika mereka memiliki, JavaScript memicu penyegaran halaman sehingga pengguna melihat forum telah diperbarui.

Mari kita lihat secara rinci cara aplikasi bekerja.

Halaman beranda

Setelah pengguna log in, aplikasi menampilkan tiga daftar informasi berikut.



- Undangan Daftar ini menampilkan hingga 10 undangan terbaru dari orang lain yang menunggu penerimaan oleh pengguna yang masuk. Dalam screenshot sebelumnya, user1 memiliki undangan dari user5 dan user2 tertunda.
- Game yang sedang berlangsung Daftar ini menampilkan hingga 10 game terbaru yang sedang berlangsung. Ini adalah game yang dimainkan pengguna secara aktif, yang mempunyai status IN_PROGRESS. Dalam screenshot, user1 secara aktif memainkan tic-tac-toe game dengan user3 dan user4.
- Riwayat terbaru Daftar ini menampilkan hingga 10 game terbaru yang ditelah diselesaikan pengguna, yang memiliki status FINISHED. Dalam game yang ditampilkan dengan screenshot, user1 sebelumnya telah bermain dengan user2. Untuk setiap game yang telah selesai, daftar menunjukkan hasil game.

Dalam kode, `index` fungsi (di `application.py`) membuat tiga panggilan berikut untuk mengambil informasi status game:

```
inviteGames      = controller.getGameInvites(session["username"])
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames    = controller.getGamesWithStatus(session["username"], "FINISHED")
```

Setiap panggilan ini mengembalikan daftar item dari DynamoDB yang dibungkus oleh objek Game. Sangat mudah untuk mengekstrak data dari objek-objek ini dalam tampilan. Fungsi indeks melewati daftar objek ini menjadi tampilan untuk membuat HTML.

```
return render_template("index.html",
```

```
user=session["username"],
invites=inviteGames,
inprogress=inProgressGames,
finished=finishedGames)
```

Aplikasi Tic-Tac-Toe mendefinisikan kelas utama Game untuk menyimpan data game yang diambil dari DynamoDB. Fungsi-fungsi ini mengembalikan daftar objek Game yang memungkinkan Anda untuk mengisolasi sisa aplikasi dari kode yang berkaitan dengan item Amazon DynamoDB. Dengan demikian, fungsi-fungsi ini membantu Anda memisahkan kode aplikasi Anda dari detail lapisan penyimpanan data.

Pola arsitektur yang dijelaskan di sini juga disebut sebagai pola UI model-view-controller (MVC). Dalam hal ini, instans objek Game (mewakili data) adalah model, dan halaman HTML adalah tampilan. Kontroler dibagi menjadi dua file. File `application.py` memiliki logika pengendali untuk kerangka kerja Flask, dan logika bisnis terisolasi dalam file `gameController.py`. Artinya, aplikasi menyimpan segala sesuatu yang ada hubungannya dengan DynamoDB SDK dalam file tersendiri di folder `dynamodb`.

Mari kita meninjau tiga fungsi dan cara mereka meng-kueri tabel Games menggunakan indeks sekunder global untuk mengambil data yang relevan.

Menggunakan `getGameInvites` untuk mendapatkan daftar undangan game yang tertunda

Fungsi `getGameInvites` mengambil daftar 10 undangan tertunda terbaru. Game ini telah dibuat oleh pengguna, namun lawan belum menerima undangan game. Untuk game-game ini, statusnya tetap PENDING sampai lawan menerima undangan. Jika lawan menolak undangan, aplikasi akan menghapus item yang sesuai dari tabel.

Fungsi menentukan kueri sebagai berikut:

- Fungsi menentukan indeks `OpponentId-StatusDate-index` untuk digunakan dengan nilai kunci indeks berikut dan operator perbandingan:
 - Kunci partisi adalah `OpponentId` dan mengambil kunci indeks *user ID*.
 - Kunci semacam ini adalah `StatusDate` dan mengambil operator perbandingan dan nilai kunci indeks `beginswith="PENDING_"`.

Anda menggunakan indeks `OpponentId-StatusDate-index` untuk mengambil game di mana pengguna yang log in adalah yang diundang, yaitu pengguna yang log in adalah lawan.

- Kueri membatasi hasilnya untuk 10 item.

```
gameInvitesIndex = self.cm.getGamesTable().query(
    Opponent__eq=user,
    StatusDate__beginswith="PENDING_",
    index="OpponentId-StatusDate-index",
    limit=10)
```

Dalam indeks, untuk setiap OpponentId (kunci partisi) DynamoDB menyimpan item yang diurutkan berdasarkan (kunci) StatusDate. Oleh karena itu, game yang dikembalikan oleh kueri akan menjadi 10 game terbaru.

Menggunakan getGamesWith Status untuk mendapatkan daftar game dengan status tertentu

Setelah lawan menerima undangan game, status game berubah menjadi IN_PROGRESS. Setelah game selesai, status berubah menjadi FINISHED.

Pertanyaan untuk menemukan game yang sedang berlangsung atau selesai adalah sama kecuali untuk nilai status yang berbeda. Oleh karena itu, aplikasi mendefinisikan getGamesWithStatus fungsi, yang mengambil nilai status sebagai parameter.

```
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

Bagian berikut membahas game yang sedang berlangsung, namun deskripsi yang sama juga berlaku untuk game yang sudah selesai.

Daftar game yang sedang berlangsung untuk pengguna tertentu meliputi hal berikut ini:

- Game yang sedang berlangsung yang diselenggarakan oleh pengguna
- Game yang sedang berlangsung dimana pengguna adalah lawan

Fungsi getGamesWithStatus menjalankan dua kueri berikut, setiap kali menggunakan indeks sekunder yang sesuai.

- Fungsi mengkueri tabel Games menggunakan indeks HostId-StatusDate-index. Untuk indeks, kueri menentukan nilai kunci primer baik kunci partisi (HostId) dan nilai (StatusDate) kunci, bersama dengan operator perbandingan.

```
hostGamesInProgress = self.cm.getGamesTable ().query(HostId__eq=user,
```

```
StatusDate__beginswith=status,  
index="HostId-StatusDate-index",  
limit=10)
```

Perhatikan sintaks Python untuk operator perbandingan:

- `HostId__eq=user` menentukan operator perbandingan kesetaraan.
- `StatusDate__beginswith=status` menentukan operator perbandingan `BEGINS_WITH`.
- Fungsi meng-kueri tabel Games menggunakan indeks `OpponentId-StatusDate-index`.

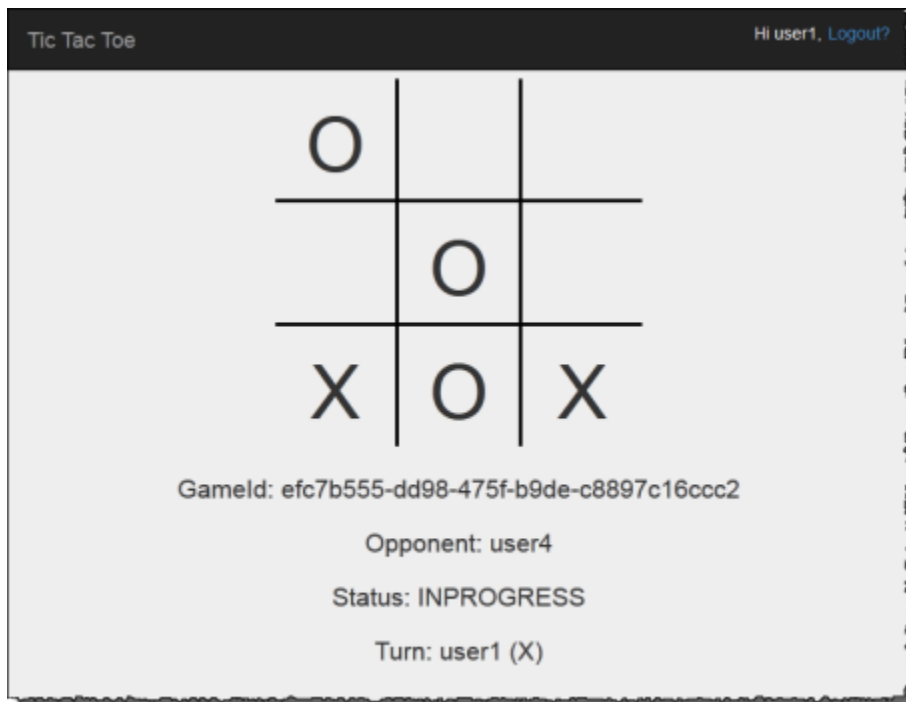
```
oppGamesInProgress = self.cm.getGamesTable().query(Opponent__eq=user,  
                                                    StatusDate__beginswith=status,  
                                                    index="OpponentId-StatusDate-index",  
                                                    limit=10)
```

- Fungsi kemudian menggabungkan dua daftar, dan untuk 0 sampai 10 item pertama menciptakan daftar objek Game dan mengembalikan daftar ke fungsi panggilan (yaitu, indeks).

```
games = self.mergeQueries(hostGamesInProgress,  
                          oppGamesInProgress)  
return games
```

Halaman permainan

Halaman game adalah tempat pengguna memainkan tic-tac-toe game. Ini menunjukkan grid game bersama dengan informasi yang relevan terkait game. Screenshot berikut menunjukkan contoh game yang sedang dimainkan:



Aplikasi menampilkan halaman game dalam situasi-situasi berikut:

- Pengguna membuat game mengundang pengguna lain untuk ikut bermain.

Dalam hal ini, halaman menunjukkan pengguna sebagai host dan status game sebagai PENDING, menunggu lawan untuk menerima undangan.

- Pengguna menerima salah satu undangan yang tertunda di beranda.

Dalam hal ini, halaman menunjukkan pengguna sebagai lawan dan status game sebagai IN_PROGRESS.

Pemilihan pengguna di papan menghasilkan permintaan POST formulir untuk aplikasi. Artinya, Flask memanggil fungsi `selectSquare` (di `application.py`) dengan data bentuk HTML. Fungsi ini, pada gilirannya, memanggil fungsi `updateBoardAndTurn` (di `gameController.py`) untuk memperbarui item game sebagai berikut:

- Fungsi menambahkan atribut baru khusus untuk gerakan.
- Fungsi memperbarui nilai atribut Turn bagi pengguna yang punya giliran selanjutnya.

```
controller.updateBoardAndTurn(item, value, session["username"])
```

Fungsi mengembalikan BETUL jika pembaruan item berhasil; jika tidak, fungsi akan mengembalikan SALAH. Perhatikan hal berikut terkait fungsi `updateBoardAndTurn`:

- Fungsi memanggil fungsi `update_item` SDK for Python dalam membuat satu set pembaruan terbatas untuk item yang sudah ada. Fungsi peta untuk operasi `UpdateItem` di DynamoDB. Untuk informasi selengkapnya, lihat [UpdateItem](#).

Note

Perbedaan antara operasi `UpdateItem` dan `PutItem` adalah bahwa `PutItem` menggantikan seluruh item. Untuk informasi selengkapnya, lihat [PutItem](#).

Untuk panggilan `update_item`, kode mengidentifikasi berikut:

- Kunci utama tabel Games (yaitu, `ItemId`).

```
key = { "GameId" : { "S" : gameId } }
```

- Atribut baru untuk ditambahkan, khusus untuk gerakan pengguna saat ini, dan nilainya (misalnya, `TopLeft="X"`).

```
attributeUpdates = {  
  position : {  
    "Action" : "PUT",  
    "Value" : { "S" : representation }  
  }  
}
```

- Kondisi yang harus BETUL agar pembaruan dapat diterapkan:
 - Game harus sedang berjalan atau berlangsung. Yaitu, nilai atribut `StatusDate` harus dimulai dengan `IN_PROGRESS`.
 - Giliran saat ini harus mengubah pengguna yang valid seperti yang ditentukan oleh atribut `Turn`.
 - Kotak yang dipilih pengguna harus tersedia. Artinya, atribut yang sesuai dengan kotak tidak boleh ada.

```
expectations = {"StatusDate" : {"AttributeValueList": [{"S" : "IN_PROGRESS_"}],  
  "ComparisonOperator": "BEGINS_WITH"},  
  "Turn" : {"Value" : {"S" : current_player}},
```

```
position : {"Exists" : False}}
```

Sekarang fungsi memanggil `update_item` untuk memperbarui item.

```
self.cm.db.update_item("Games", key=key,
    attribute_updates=attributeUpdates,
    expected=expectations)
```

Setelah fungsi kembali, fungsi `selectSquare` memanggil pengalihan, seperti yang ditunjukkan dalam contoh berikut.

```
redirect("/game="+gameId)
```

Panggilan ini menyebabkan peramban untuk disegarkan. Sebagai bagian dari penyegaran ini, aplikasi memeriksa apakah game telah berakhir dengan kemenangan atau imbang. Jika game sudah selesai, aplikasi akan memperbarui item game.

Langkah 3: men-deploy dalam Produksi menggunakan layanan DynamoDB

Topik

- [3.1: Buat IAM role untuk Amazon EC2](#)
- [3.2: Buat tabel game di Amazon DynamoDB](#)
- [3.3: paket dan men-deploy kode tic-tac-toe aplikasi](#)
- [3.4: SiapkanAWS Elastic Beanstalk Lingkungan](#)

Di bagian sebelumnya, Anda men-deploy dan menguji aplikasi Tic-Tac-Toe secara lokal pada komputer Anda menggunakan DynamoDB lokal. Sekarang, Anda men-deploy aplikasi dalam produksi sebagai berikut:

- men-deploy aplikasi menggunakanAWS Elastic Beanstalk, easy-to-use layanan untuk men-deploy dan menskalakan aplikasi dan layanan web. Untuk informasi selengkapnya, lihat [men-deploy aplikasi labu keAWS Elastic Beanstalk](#).

Elastic Beanstalk meluncurkan satu atau lebih instans Amazon Elastic Compute Cloud (Amazon EC2), yang Anda konfigurasi melalui Elastic Beanstalk, di mana aplikasi Tic-Tac-Toe Anda akan berjalan.

- Menggunakan layanan Amazon DynamoDB, membuat tabel Games yang ada AWS bukan secara lokal di komputer Anda.

Selain itu, Anda juga harus mengkonfigurasi izin. Sumber daya AWS yang Anda buat, seperti tabel Games di DynamoDB, bersifat privat secara default. Hanya pemilik sumber daya, yaitu akun AWS yang membuat tabel Games, yang dapat mengakses tabel ini. Dengan demikian, secara default aplikasi Tic-Tac-Toe tidak dapat memperbarui tabel Games.

Untuk memberikan izin yang diperlukan, Anda membuat AWS Identity and Access Management (IAM) role dan memberikan izin peran ini untuk mengakses tabel Games. Instans Amazon EC2 Anda pertama-tama akan mengasumsikan peran ini. Sebagai tanggapan, AWS mengembalikan mandat keamanan sementara yang dapat digunakan instans Amazon EC2 untuk memperbarui tabel Games mewakili aplikasi Tic-Tac-Toe. Bila Anda mengkonfigurasi aplikasi Elastic Beanstalk, Anda menentukan IAM role yang dapat diasumsikan oleh instans atau instans Amazon EC2. Untuk informasi selengkapnya tentang IAM role, lihat [IAM role untuk amazon EC2](#) di Panduan Pengguna Amazon EC2 untuk Instance Linux.

Note

Sebelum Anda membuat instans Amazon EC2 untuk aplikasi Tic-Tac-Toe, Anda harus terlebih dahulu memutuskan Wilayah AWS mana yang Anda inginkan untuk dibuatkan instans oleh Elastic Beanstalk. Setelah Anda membuat aplikasi Elastic Beanstalk, Anda memberikan nama wilayah yang sama dan titik akhir dalam file konfigurasi. Aplikasi Tic-Tac-Toe menggunakan informasi dalam file ini untuk membuat tabel Games dan mengirim permintaan berikutnya dalam Wilayah AWS khusus. Kedua tabel DynamoDB Games dan instans Amazon EC2 yang diluncurkan oleh Elastic Beanstalk harus ada di Wilayah yang sama. Untuk daftar Wilayah yang tersedia, lihat [Amazon DynamoDB](#) di Referensi Umum Amazon Web.

Singkatnya, Anda melakukan hal berikut untuk men-deploy aplikasi Tic-Tac-Toe dalam produksi:

1. Buat IAM role menggunakan layanan IAM. Anda melampirkan kebijakan untuk peran ini yang memberikan izin untuk tindakan DynamoDB dalam mengakses tabel Games.
2. paket kode aplikasi Tic-Tac-Toe dan file konfigurasi, dan buat file `.zip`. Anda gunakan file `.zip` ini untuk memberikan kode aplikasi Tic-Tac-Toe kepada Elastic Beanstalk untuk ditempatkan pada

server Anda. Untuk informasi selengkapnya tentang cara membuat paket, lihat [Membuat paket sumber aplikasi](#) di PanduanAWS Elastic Beanstalk Developer.

Dalam file konfigurasi (`beanstalk.config`), Anda memberikan informasi Wilayah dan titik akhir AWS. Aplikasi Tic-Tac-Toe menggunakan informasi ini untuk menentukan Wilayah DynamoDB untuk diajak berkomunikasi.

3. Mengatur lingkungan Elastic Beanstalk. Elastic Beanstalk meluncurkan instans Amazon EC2 atau instans dan men-deploy paket aplikasi Tic-Tac-Toe Anda pada mereka. Setelah lingkungan Elastic Beanstalk siap, Anda memberikan nama file konfigurasi dengan menambahkan variabel lingkungan `CONFIG_FILE`.
4. Buat tabel DynamoDB. Dengan menggunakan layanan Amazon DynamoDB, buat tabel Games di AWS bukan secara lokal di komputer Anda. Ingat, tabel ini memiliki kunci primer sederhana yang terbuat dari kurici partisi `GameId` dari tipe string.
5. Uji game dalam produksi.

3.1: Buat IAM role untuk Amazon EC2

Menciptakan IAM role Amazon EC2 jenis memungkinkan instans Amazon EC2 yang menjalankan aplikasi Tic-Tac-Toe untuk menganggap peran yang benar dan membuat permintaan aplikasi untuk mengakses tabel Games. Saat membuat peran, pilih opsi Kebijakan Kustom dan salin serta tempel kebijakan berikut.

```
{
  "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/*"  
    ]  
}  
]  
}
```

Untuk instruksi selanjutnya, lihat [Membuat role untuk AWS layanan \(AWS Management Console\)](#) di Panduan Pengguna IAM.

3.2: Buat tabel game di Amazon DynamoDB

Tabel Games di DynamoDB menyimpan data game. Jika tabel tidak ada, aplikasi yang akan membuat tabel untuk Anda. Dalam hal ini, biarkan aplikasi membuat tabel Games.

3.3: paket dan men-deploy kode tic-tac-toe aplikasi

Jika anda mengikuti langkah-langkah contoh ini, maka Anda sudah mengunduh aplikasi Tic-Tac-Toe. Jika belum, unduh aplikasi dan ekstrak semua file ke folder di komputer lokal Anda. Untuk instruksi selengkapnya, lihat [Langkah 1: men-deploy dan menguji secara lokal](#).

Setelah Anda mengekstrak semua file, Anda akan memiliki folder code. Untuk menyerahkan folder ini ke Elastic Beanstalk, paket isi folder ini sebagai file `.zip`. Pertama, tambahkan file konfigurasi ke folder tersebut. Aplikasi Anda menggunakan Wilayah dan titik akhir informasi untuk membuat tabel DynamoDB di Wilayah tertentu dan membuat permintaan operasi tabel berikutnya menggunakan titik akhir yang ditentukan.

1. Beralih ke folder tempat Anda mengunduh aplikasi Tic-Tac-Toe.
2. Dalam folder root aplikasi, buat file teks bernama `beanstalk.config` dengan konten berikut.

```
[dynamodb]  
region=<AWS region>  
endpoint=<DynamoDB endpoint>
```

Misalnya, Anda dapat menjalankan perintah berikut.

```
[dynamodb]  
region=us-west-2  
endpoint=dynamodb.us-west-2.amazonaws.com
```

Untuk daftar Wilayah yang tersedia, lihat [Amazon DynamoDB](#) di Amazon Web Services General Reference.

Important

Wilayah yang ditentukan dalam file konfigurasi adalah lokasi di mana aplikasi Tic-Tac-Toe membuat tabel Games di DynamoDB. Anda harus membuat aplikasi Elastic Beanstalk yang dibahas pada bagian berikutnya di Wilayah yang sama.

Note

Ketika Anda membuat aplikasi Elastic Beanstalk Anda, Anda meminta untuk meluncurkan lingkungan di mana Anda dapat memilih jenis lingkungan. Untuk menguji aplikasi contoh Tic-Tac-Toe, Anda dapat memilih jenis lingkungan Instans Tunggal, lewati hal berikut, dan lanjutkan ke langkah berikutnya.

Namun, jenis lingkungan Penyeimbangan beban, penskalaan otomatis menyediakan lingkungan yang sangat tersedia dan dapat diskalakan, sesuatu yang harus Anda pertimbangkan ketika Anda membuat dan men-deploy aplikasi lain. Jika Anda memilih jenis lingkungan ini, Anda juga perlu untuk menghasilkan UUID dan menambahkannya ke file konfigurasi, seperti yang ditunjukkan berikut.

```
[dynamodb]
region=us-west-2
endpoint=dynamodb.us-west-2.amazonaws.com
[flask]
secret_key= 284e784d-1a25-4a19-92bf-8eeb7a9example
```

Dalam komunikasi klien server, ketika server mengirimkan respon, demi keamanan server mengirimkan cookie yang ditandai lalu klien mengirimkan kembali ke server dalam permintaan berikutnya. Ketika hanya ada satu server, server dapat secara lokal menghasilkan kunci enkripsi saat server dimulai. Ketika ada banyak server, semua server perlu tahu kunci enkripsi yang sama; jika tidak, server tersebut tidak akan dapat membaca cookie yang ditetapkan oleh server peer. Dengan menambahkan `secret_key` ke file konfigurasi, Anda memberitahu semua server untuk menggunakan kunci enkripsi ini.

3. Buka isi dari folder root aplikasi (yang mencakup file `beanstalk.config`), misalnya, `TicTacToe.zip`.
4. Unggah file `.zip` ke bucket Amazon Simple Storage Service (Amazon S3). Pada bagian berikutnya, Anda menyediakan file `.zip` ini ke Elastic Beanstalk untuk mengunggah di server atau di beberapa server.

Untuk instruksi tentang cara mengunggah ke bucket Amazon S3, lihat [Buat bucket](#) dan [Menambahkan objek ke bucket](#) di Panduan Pengguna Amazon Simple Storage Service.

3.4: Siapkan AWS Elastic Beanstalk Lingkungan

Pada langkah ini, Anda membuat aplikasi Elastic Beanstalk, yang merupakan kumpulan komponen termasuk lingkungan. Untuk contoh ini, Anda meluncurkan satu instans Amazon EC2 untuk men-deploy dan menjalankan aplikasi Tic-Tac-Toe Anda.

1. Masukkan URL kustom berikut untuk mengatur konsol Elastic Beanstalk dalam mengatur lingkungan.

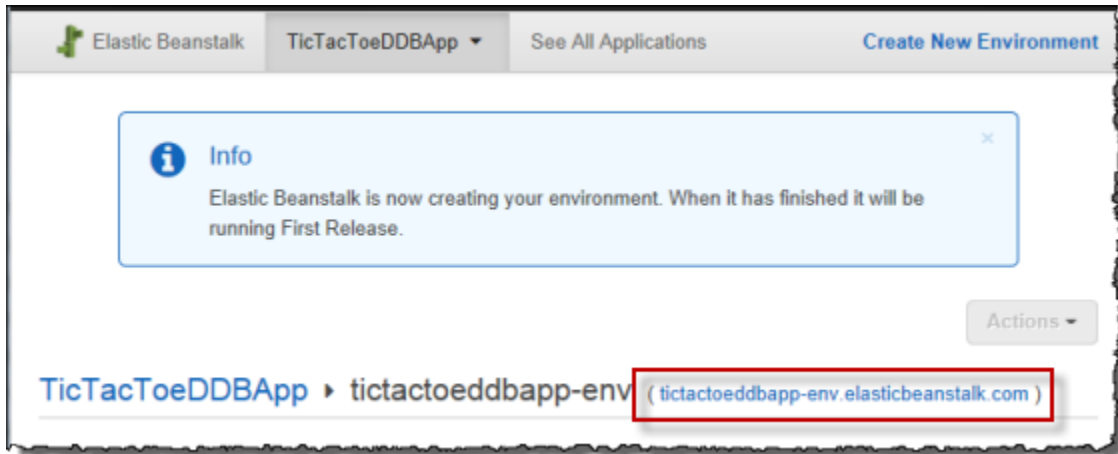
```
https://console.aws.amazon.com/elasticbeanstalk/?region=<AWS-Region>#/
newApplication
?applicationName=TicTacToe<your-name>
&solutionStackName=Python
&sourceBundleUrl=https://s3.amazonaws.com/<bucket-name>/TicTacToe.zip
&environmentType=SingleInstance
&instanceType=t1.micro
```

Untuk informasi selengkapnya tentang URL khusus, lihat [Membangun URL Peluncuran Sekarang](#) di AWS Elastic Beanstalk Panduan Developer. Untuk URL, perhatikan hal berikut:

- Anda harus menyediakan nama Wilayah AWS (nama yang sama dengan yang Anda berikan dalam file konfigurasi), nama dan nama objek bucket Amazon S3.
- Untuk pengujian, URL meminta jenis `SingleInstance` lingkungan, dan `t1.micro` sebagai jenis instans.
- Nama aplikasi harus unik. Dengan demikian, di URL sebelumnya, kami sarankan Anda menambahkan nama Anda ke `applicationName`.

Melakukan hal ini akan membuka konsol Elastic Beanstalk. Dalam beberapa kasus, Anda mungkin perlu masuk.

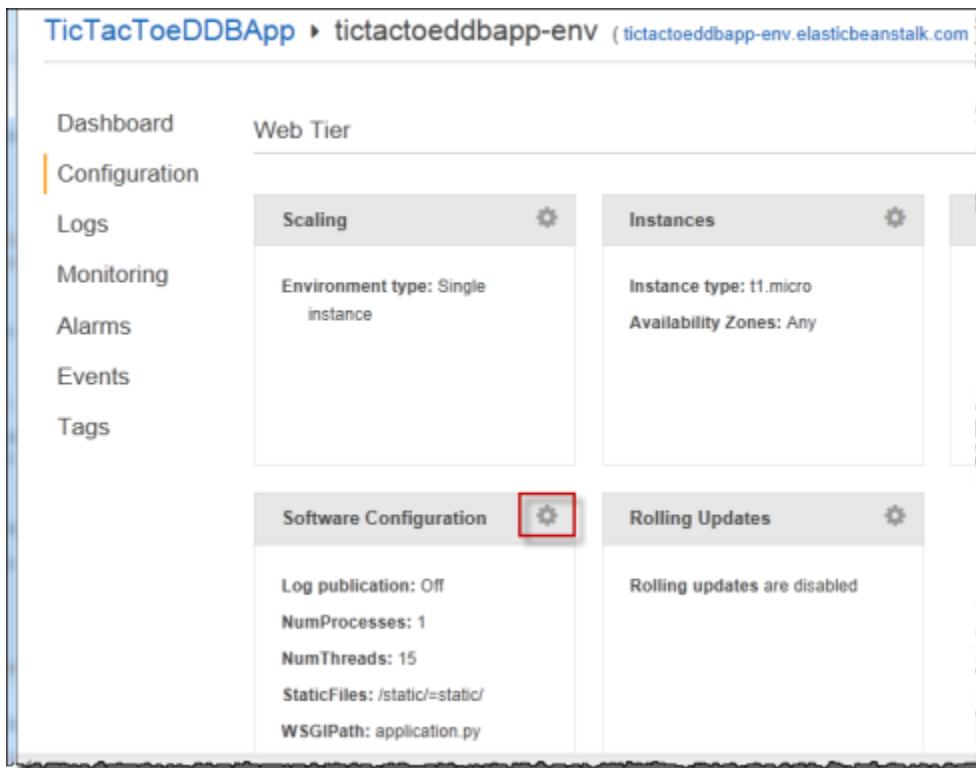
2. Di konsol Elastic Beanstalk, pilih Tinjau dan Luncurkan, lalu pilih Luncurkan.
3. Perhatikan URL untuk referensi di masa mendatang. URL ini membuka halaman aplikasi Tic-Tac-Toe Anda.



4. Konfigurasi aplikasi Tic-Tac-Toe supaya aplikasi tahu lokasi file konfigurasi.

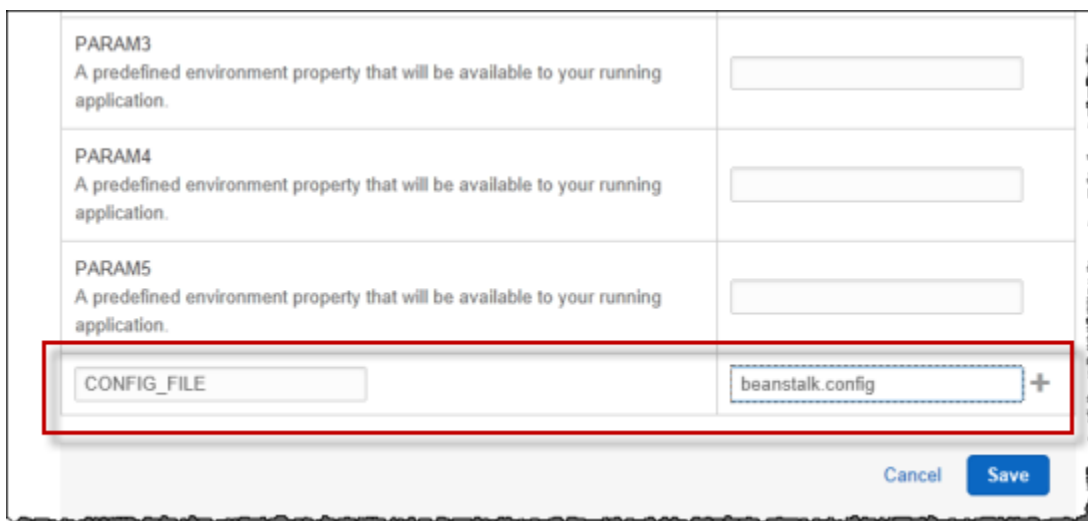
Setelah Elastic Beanstalk membuat aplikasi, pilih Konfigurasi.

- a. Pilih ikon roda gigi di sebelah Konfigurasi Perangkat Lunak, seperti yang ditunjukkan dalam gambar berikut.



- b. Di akhir bagian Properti Lingkungan, masukkan **CONFIG_FILE** dan nilai **beanstalk.config**, lalu pilih Simpan.

Mungkin diperlukan waktu beberapa menit agar pembaruan lingkungan selesai.

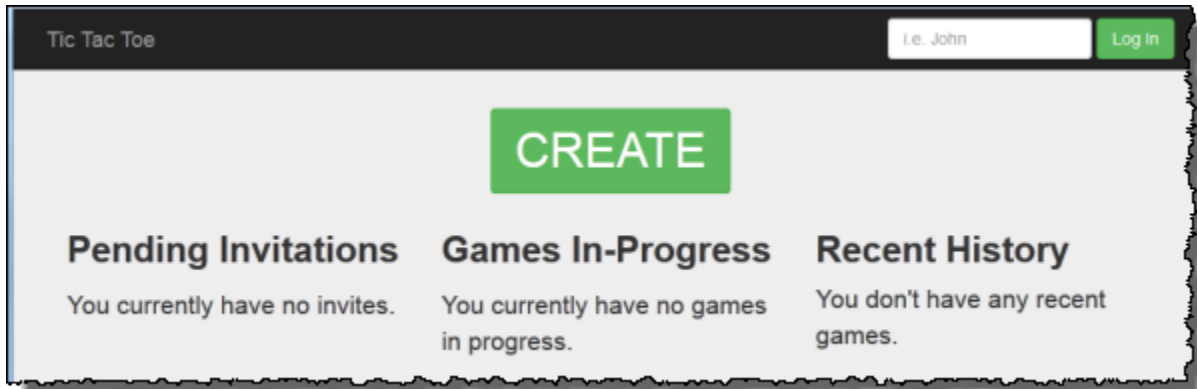


Setelah pembaruan selesai, Anda bisa memainkan game.

5. Di peramban, masukkan URL yang Anda salin di langkah sebelumnya, seperti yang ditunjukkan dalam contoh berikut.

```
http://<pen-name>.elasticbeanstalk.com
```

Melakukan hal ini akan membuka halaman beranda aplikasi.



6. Masuk sebagai testuser1, dan pilih BUAT untuk memulai tic-tac-toe game baru.
7. Masukkan **testuser2** di kotak Pilih Lawan.



8. Buka jendela peramban lain.

Pastikan bahwa Anda menghapus semua cookie di jendela peramban Anda sehingga Anda tidak akan masuk sebagai pengguna yang sama.

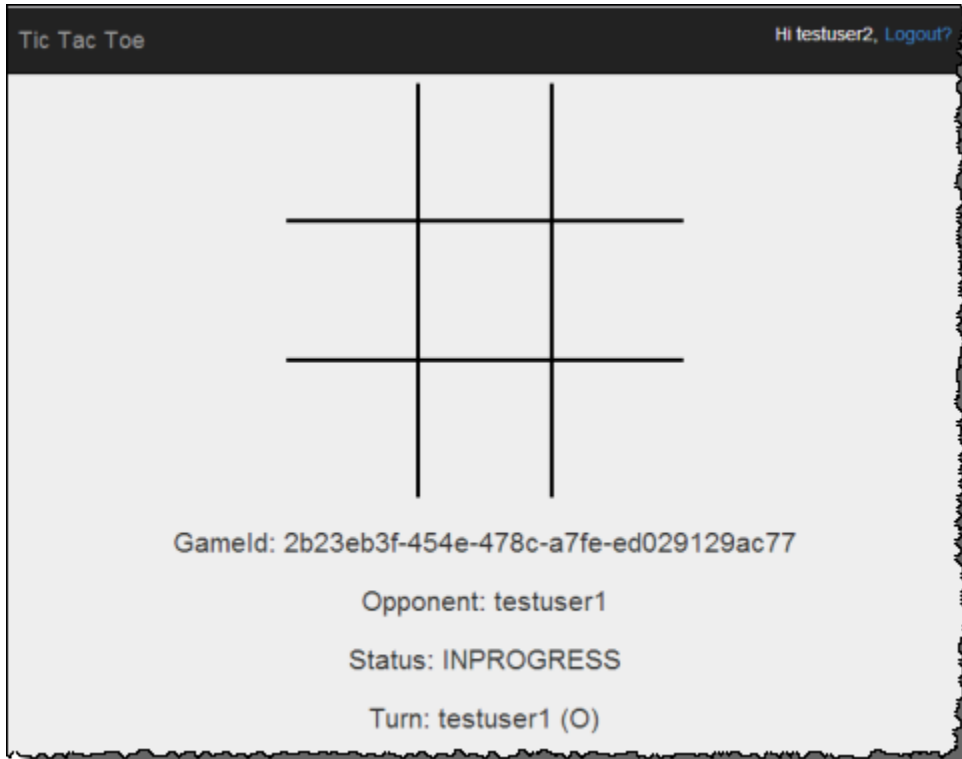
9. Masukkan URL yang sama untuk membuka halaman beranda aplikasi, seperti yang ditunjukkan dalam contoh berikut.

```
http://<env-name>.elasticbeanstalk.com
```

10. Masuk sebagai testuser2.
11. Untuk undangan dari testuser1 dalam daftar undangan yang tertunda, pilih terima.



12. Sekarang halaman game muncul.



Kedua testuser1 dan testuser2 bisa bermain game. Untuk setiap langkah, aplikasi menyimpan gerakan dalam item yang sesuai dengan yang ada di tabel Games.

Langkah 4: Bersihkan Sumber Daya

Sekarang Anda telah menyelesaikan deployment dan pengujian aplikasi Tic-Tac-Toe. Aplikasi ini mencakup Developeran aplikasi end-to-end web di Amazon DynamoDB, kecuali untuk autentikasi pengguna. Aplikasi ini menggunakan informasi login di halaman beranda hanya untuk menambahkan nama pemain saat membuat game. Dalam aplikasi produksi, Anda akan menambahkan kode yang diperlukan untuk melakukan login pengguna dan autentikasi.

Jika Anda telah selesai melakukan pengujian, Anda dapat menghapus sumber daya yang Anda buat untuk menguji aplikasi Tic-Tac-Toe untuk menghindari timbulnya biaya.

Untuk menghapus sumber daya yang Anda buat

1. Hapus tabel Games yang Anda buat di DynamoDB.
2. akhiri lingkungan Elastic Beanstalk untuk membebaskan instans Amazon EC2.
3. Hapus IAM role yang Anda buat.
4. Hapus objek yang Anda buat di Amazon S3.

Mengekspor dan mengimpor data DynamoDB menggunakan AWS Data Pipeline

Anda dapat menggunakan AWS Data Pipeline untuk mengekspor data dari tabel DynamoDB ke file di bucket Amazon S3. Anda juga dapat menggunakan konsol untuk mengimpor data dari Amazon S3 ke tabel DynamoDB, di wilayah AWS yang sama atau berbeda.

Note

Kini Konsol DynamoDB mendukung impor dari Amazon S3 dan ekspor ke Amazon S3 secara native. Aliran ini tidak kompatibel dengan aliran AWS Data Pipeline impor. Untuk informasi selengkapnya, lihat [Mengimpor dari Amazon S3](#), [Mengekspor dari Amazon S3](#), dan postingan blog [Export Amazon DynamoDB table data to your data lake in Amazon S3](#).

Kemampuan untuk mengekspor dan mengimpor data berguna dalam banyak skenario. Sebagai contoh, Anda ingin mempertahankan suatu set acuan data untuk tujuan pengujian. Anda dapat memasukkan data acuan ke tabel DynamoDB dan mengekspornya ke Amazon S3. Kemudian,

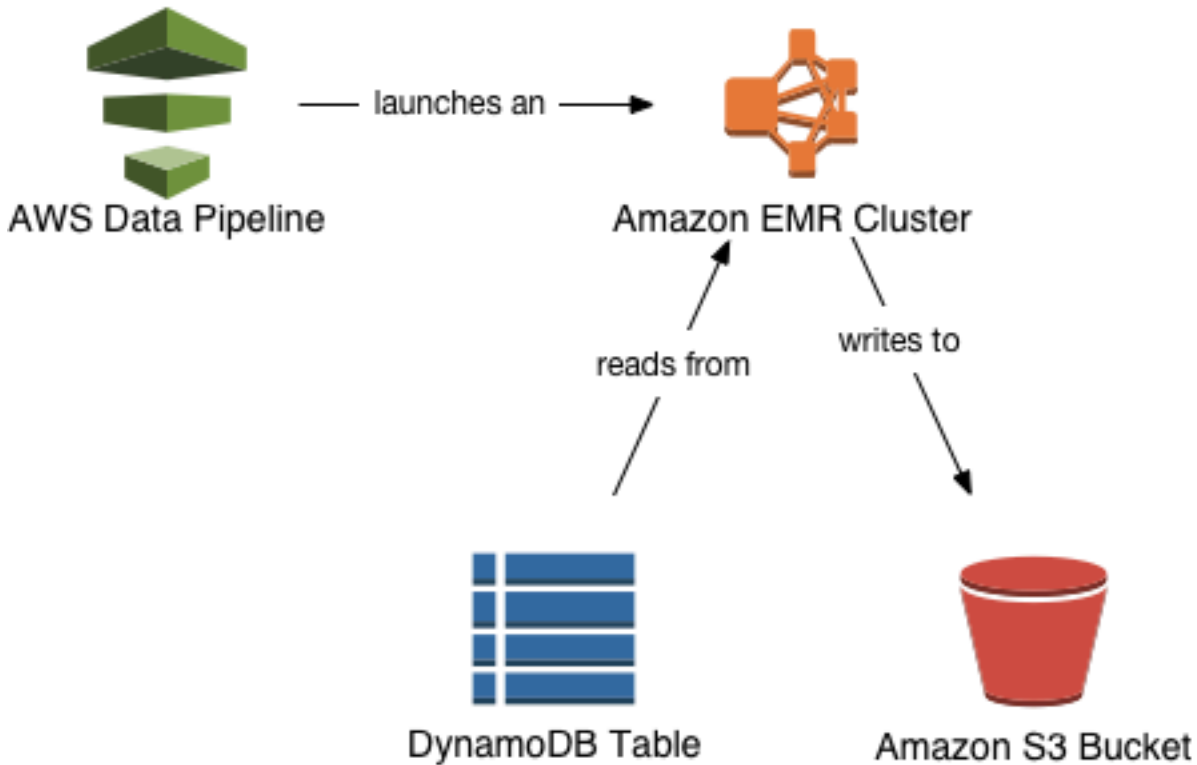
setelah menjalankan aplikasi yang memodifikasi data pengujian, Anda dapat "mereset" set data dengan mengimpor acuannya dari Amazon S3 kembali ke tabel DynamoDB. Contoh lain mencakup penghapusan data secara tidak sengaja, bahkan operasi `DeleteTable` yang tidak disengaja. Dalam kasus ini, Anda dapat memulihkan data dari file ekspor sebelumnya di Amazon S3. Anda bahkan dapat menyalin data dari tabel DynamoDB dalam satu wilayah AWS, menyimpan data di Amazon S3, lalu mengimpor data dari Amazon S3 ke tabel DynamoDB yang sama di wilayah kedua. Aplikasi di wilayah kedua kemudian dapat mengakses titik akhir DynamoDB terdekat dan bekerja menggunakan salinan data sendiri dengan latensi jaringan yang berkurang.

Important

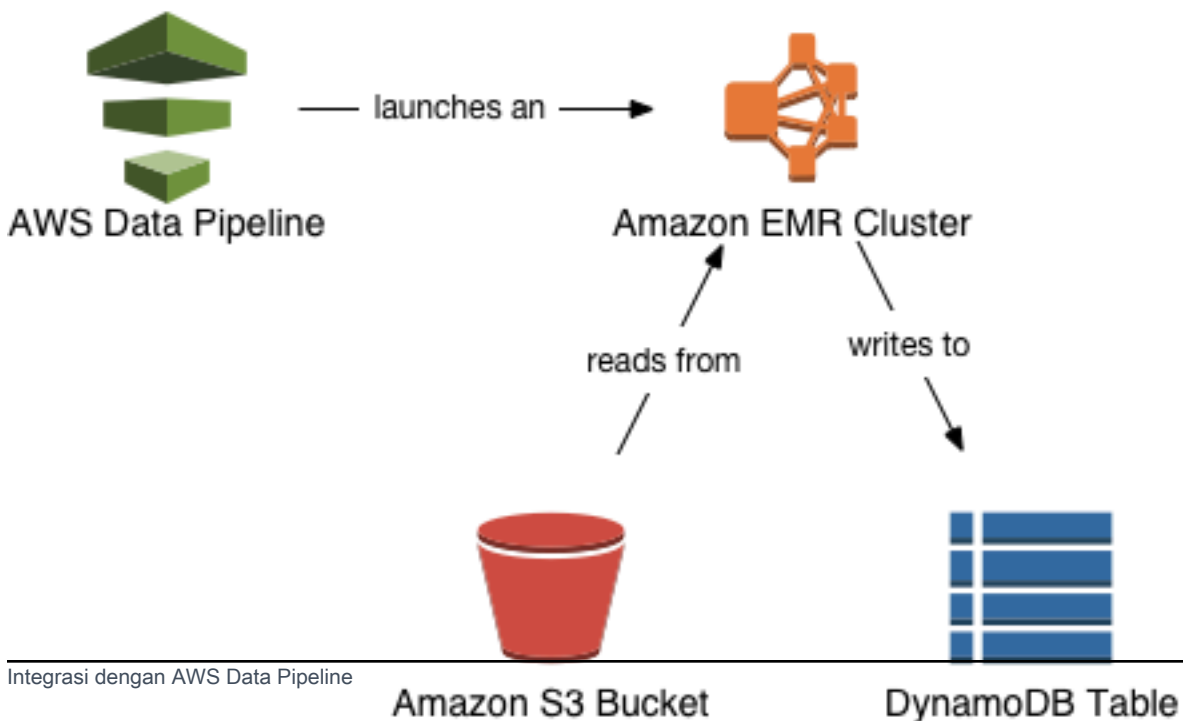
Pencadangan dan Pemulihan DynamoDB adalah fitur yang dikelola sepenuhnya. Anda dapat mencadangkan tabel berisi mulai dari beberapa megabyte hingga ratusan terabyte data tanpa memengaruhi performa dan ketersediaan aplikasi produksi Anda. Anda dapat memulihkan tabel Anda dengan satu klik dalam AWS Management Console atau satu panggilan API. Kami sangat menyarankan agar Anda menggunakan fitur backup dan restore asli DynamoDB daripada menggunakan AWS Data Pipeline Untuk informasi selengkapnya, lihat [Menggunakan cadangan Sesuai Permintaan dan DynamoDB Permintaan](#).

Diagram berikut menunjukkan gambaran umum mengekspor dan mengimpor data DynamoDB menggunakan AWS Data Pipeline.

Exporting Data from DynamoDB to Amazon S3



Importing Data from Amazon S3 to DynamoDB



Untuk mengekspor tabel DynamoDB, Anda menggunakan AWS Data Pipeline konsol untuk membuat pipeline baru. Pipeline meluncurkan kluster Amazon EMR untuk melakukan ekspor yang sebenarnya. Amazon EMR membaca data dari DynamoDB dan menulis data ke file ekspor di bucket Amazon S3.

Proses untuk impor sama, kecuali bahwa data dibaca dari bucket Amazon S3 dan ditulis ke tabel DynamoDB.

Important

Saat Anda mengekspor atau mengimpor data DynamoDB, Anda akan dikenakan biaya tambahan untuk layanan AWS dasar yang digunakan:

- AWS Data Pipeline— mengelola alur kerja impor/ekspor untuk Anda.
- Amazon S3— berisi data yang Anda ekspor dari DynamoDB atau impor ke DynamoDB.
- Amazon EMR— menjalankan kluster Hadoop terkelola untuk melakukan pembacaan dan penulisan antara DynamoDB ke Amazon S3. Konfigurasi kluster adalah satu simpul pemimpin instans `m3.xlarge` dan satu simpul inti instans `m3.xlarge`.

Untuk informasi selengkapnya, lihat [Harga AWS Data Pipeline](#), [Harga Amazon EMR](#), dan [Harga Amazon S3](#).

Prasyarat untuk ekspor dan impor data

Saat Anda menggunakan AWS Data Pipeline untuk mengekspor dan mengimpor data, Anda harus menentukan tindakan yang diizinkan untuk dilakukan pipeline, dan sumber daya mana yang dapat dikonsumsi pipeline. Tindakan dan sumber daya yang diizinkan didefinisikan menggunakan peran AWS Identity and Access Management (IAM).

Anda juga dapat mengontrol akses dengan membuat kebijakan IAM dan melampirkannya ke pengguna, peran, atau grup. Kebijakan ini memungkinkan Anda menentukan pengguna yang diizinkan untuk mengimpor dan mengekspor data DynamoDB Anda.

Important

Pengguna membutuhkan akses terprogram jika mereka ingin berinteraksi dengan AWS luar. AWS Management Console Cara untuk memberikan akses terprogram tergantung pada jenis pengguna yang mengakses AWS.

Untuk memberi pengguna akses terprogram, pilih salah satu opsi berikut.

Pengguna mana yang membutuhkan akses terprogram?	Untuk	Oleh
Identitas tenaga kerja (Pengguna yang dikelola di Pusat Identitas IAM)	Gunakan kredensial sementara untuk menandatangani permintaan terprogram ke AWS CLI, AWS SDK, atau API. AWS	<p>Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan.</p> <ul style="list-style-type: none"> • Untuk AWS CLI, lihat Mengkonfigurasi yang akan AWS CLI digunakan AWS IAM Identity Center dalam Panduan AWS Command Line Interface Pengguna. • Untuk AWS SDK, alat, dan AWS API, lihat otentikasi Pusat Identitas IAM di Panduan Referensi AWS SDK dan Alat.
IAM	Gunakan kredensial sementara untuk menandatangani permintaan terprogram ke AWS CLI, AWS SDK, atau API. AWS	Mengikuti petunjuk dalam Menggunakan kredensial sementara dengan AWS sumber daya di Panduan Pengguna IAM.


Pengguna mana yang membutuhkan akses terprogram?	Untuk	Oleh
IAM	(Tidak direkomendasikan) Gunakan kredensial jangka panjang untuk menandatangani permintaan terprogram ke AWS CLI, AWS SDK, atau API. AWS	<p>Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan.</p> <ul style="list-style-type: none"> • Untuk mengetahui AWS CLI, lihat Mengautentikasi menggunakan kredensial pengguna IAM di Panduan Pengguna AWS Command Line Interface • Untuk AWS SDK dan alat bantu, lihat Mengautentikasi menggunakan kredensial jangka panjang di Panduan Referensi AWS SDK dan Alat. • Untuk AWS API, lihat Mengelola kunci akses untuk pengguna IAM di Panduan Pengguna IAM.

Membuat Peran IAM untuk AWS Data Pipeline

Untuk menggunakan AWS Data Pipeline, peran IAM berikut harus ada di AWS akun Anda:

- `DataPipelineDefaultRole`— tindakan yang dapat diambil oleh pipeline Anda atas nama Anda.
- `DataPipelineDefaultResourceRole`— AWS sumber daya yang akan disediakan pipa atas nama Anda. Untuk mengekspor dan mengimpor data DynamoDB, sumber daya tersebut mencakup kluster Amazon EMR dan instans Amazon EC2 yang terkait dengan kluster tersebut.

Jika Anda belum pernah menggunakan AWS Data Pipeline sebelumnya, Anda harus membuat `DataPipelineDefaultRole` dan `DataPipelineDefaultResourceRole` diri Anda sendiri. Setelah Anda membuat peran ini, Anda dapat menggunakannya kapan saja untuk mengekspor atau mengimpor data DynamoDB.

 Note

Jika sebelumnya Anda telah menggunakan AWS Data Pipeline konsol untuk membuat pipeline, maka `DataPipelineDefaultRole` dan `DataPipelineDefaultResourceRole` dibuat untuk Anda pada saat itu. Tidak ada tindakan lebih lanjut yang diperlukan; Anda dapat melewati bagian ini dan mulai membuat pipeline menggunakan konsol DynamoDB. Untuk informasi selengkapnya, lihat [Mengekspor data dari DynamoDB ke Amazon S3](#) dan [Mengimpor data dari Amazon S3 ke DynamoDB](#).

1. Masuk ke AWS Management Console dan buka konsol IAM di <https://console.aws.amazon.com/iam/>.
2. Dari Dasbor Konsol IAM, klik Peran.
3. Klik Buat Peran dan lakukan hal berikut:
 - a. Di entitas tepercaya Layanan AWS , pilih Data Pipeline.
 - b. Di panel Pilih kasus penggunaan Anda, pilih Data Pipeline lalu pilih Berikutnya: Izin.
 - c. Perhatikan bahwa kebijakan `AWSDataPipelineRole` dilampirkan secara otomatis. Pilih Berikutnya: Tinjauan.
 - d. Di bidang Nama peran, masukkan `DataPipelineDefaultRole` sebagai nama peran dan pilih Buat peran.
4. Klik Buat Peran dan lakukan hal berikut:
 - a. Di entitas tepercaya Layanan AWS , pilih Data Pipeline.
 - b. Di panel Pilih kasus penggunaan Anda, pilih Peran EC2 untuk Data Pipeline lalu pilih Berikutnya: Izin.
 - c. Perhatikan bahwa kebijakan `AmazonEC2RoleForDataPipelineRole` dilampirkan secara otomatis. Pilih Berikutnya: Tinjauan.
 - d. Di bidang Nama peran, masukkan `DataPipelineDefaultResourceRole` sebagai nama peran dan pilih Buat peran.

Setelah membuat peran ini, Anda dapat mulai membuat pipeline menggunakan konsol DynamoDB. Untuk informasi selengkapnya, lihat [Mengekspor data dari DynamoDB ke Amazon S3](#) dan [Mengimpor data dari Amazon S3 ke DynamoDB](#).

Memberikan izin kepada pengguna dan grup untuk melakukan tugas ekspor dan impor menggunakan AWS Identity and Access Management

Jika Anda ingin mengizinkan pengguna, peran atau grup lain untuk mengekspor dan mengimpor data tabel DynamoDB Anda, Anda dapat membuat kebijakan IAM dan melampirkannya kepada pengguna atau grup yang Anda tentukan. Kebijakan hanya berisi izin yang diperlukan untuk melakukan tugas-tugas tersebut.

Memberi akses penuh

Prosedur berikut menjelaskan cara melampirkan kebijakan AWS `AmazonDynamoDBFullAccess`, `AWSDataPipeline_FullAccess` dan kebijakan sebaris EMR Amazon ke pengguna. Kebijakan terkelola ini menyediakan akses penuh ke AWS Data Pipeline dan ke sumber daya DynamoDB, serta digunakan dengan kebijakan inline Amazon EMR, memungkinkan pengguna untuk melakukan tindakan yang dijelaskan dalam dokumentasi ini.

Note

Untuk membatasi cakupan izin yang disarankan, kebijakan inline di atas memberlakukan penggunaan tanda `dynamodbdatapipeline`. Jika ingin memanfaatkan dokumentasi tanpa batasan ini, Anda dapat menghapus bagian `Condition` dari kebijakan yang disarankan.

1. Masuk ke AWS Management Console dan buka konsol IAM di <https://console.aws.amazon.com/iam/>.
2. Dari Dasbor Konsol IAM, klik Pengguna dan pilih pengguna yang ingin Anda ubah.
3. Di tab Izin, klik Tambah Kebijakan.
4. Di panel Lampirkan izin, klik Lampirkan kebijakan yang ada secara langsung.
5. Pilih `AmazonDynamoDBFullAccess` dan `AWSDataPipeline_FullAccess` dan klik Berikutnya: Tinjauan.
6. Klik Tambahkan izin.
7. Kembali ke tab Izin, klik Tambahkan kebijakan inline.

- Di halaman Buat kebijakan, klik tab JSON.
- Tempelkan konten di bawah ini.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMR",
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:DescribeStep",
        "elasticmapreduce:DescribeCluster",
        "elasticmapreduce:RunJobFlow",
        "elasticmapreduce:TerminateJobFlows"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "elasticmapreduce:RequestTag/dynamodbdatapipeline": "false"
        }
      }
    }
  ]
}
```

- Klik Tinjau kebijakan.
- Ketikkan `EMRforDynamoDBDataPipeline` pada bidang nama.
- Klik Buat kebijakan.

Note

Anda dapat menerapkan prosedur serupa untuk melampirkan kebijakan yang dikelola ini kepada peran atau grup, bukan pengguna.

Membatasi akses ke tabel DynamoDB tertentu

Jika Anda ingin membatasi akses sehingga pengguna hanya dapat mengekspor atau mengimpor subset tabel Anda, Anda harus membuat dokumen kebijakan IAM yang disesuaikan. Anda dapat menggunakan proses yang dijelaskan di [Memberi akses penuh](#) sebagai titik awal untuk kebijakan

husus Anda, lalu memodifikasi kebijakan sehingga pengguna hanya dapat bekerja dengan tabel yang Anda tentukan.

Sebagai contoh, anggaplah bahwa Anda ingin mengizinkan seorang pengguna untuk hanya mengekspor dan mengimpor tabel Forum, Utas, dan Balas. Prosedur ini menjelaskan cara membuat kebijakan khusus sehingga pengguna dapat bekerja dengan tabel tersebut, tetapi tidak dengan tabel yang lain.

1. Masuk ke AWS Management Console dan buka konsol IAM di <https://console.aws.amazon.com/iam/>.
2. Dari Dasbor Konsol IAM, klik Kebijakan, lalu klik Buat Kebijakan.
3. Di panel Create Policy, buka Copy an AWS Managed Policy dan klik Select.
4. Di panel Salin Kebijakan AWS Terkelola, buka AmazonDynamoDBFullAccess dan klik Pilih.
5. Di panel Tinjau Kebijakan, lakukan hal berikut:
 - a. Tinjau Nama Kebijakan dan Deskripsi yang dihasilkan secara otomatis. Jika ingin, Anda dapat mengubah nilai tersebut.
 - b. Di kotak teks Dokumen Kebijakan, edit kebijakan untuk membatasi akses ke tabel tertentu. Secara default, kebijakan mengizinkan semua tindakan DynamoDB pada semua tabel Anda:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:DeleteAlarms",
        "cloudwatch:DescribeAlarmHistory",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:DescribeAlarmsForMetric",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:ListMetrics",
        "cloudwatch:PutMetricAlarm",
        "dynamodb:*",
        "sns:CreateTopic",
        "sns>DeleteTopic",
        "sns:ListSubscriptions",
        "sns:ListSubscriptionsByTopic",
        "sns:ListTopics",
        "sns:Subscribe",
        "sns:Unsubscribe"
      ]
    }
  ]
}
```

```
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Sid": "DDBConsole"
  },
  ...remainder of document omitted...
```

Untuk membatasi kebijakan, hapus baris berikut terlebih dahulu:

```
"dynamodb:*",
```

Berikutnya, buat Action baru yang memungkinkan akses hanya ke tabel Forum, Utas, dan Balas:

```
{
  "Action": [
    "dynamodb:*"
  ],
  "Effect": "Allow",
  "Resource": [
    "arn:aws:dynamodb:us-west-2:123456789012:table/Forum",
    "arn:aws:dynamodb:us-west-2:123456789012:table/Thread",
    "arn:aws:dynamodb:us-west-2:123456789012:table/Reply"
  ]
},
```

Note

Ganti `us-west-2` dengan wilayah tempat tabel DynamoDB Anda berada. Ganti `123456789012` dengan nomor AWS akun Anda.

Terakhir, tambahkan Action ke dokumen kebijakan:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
```

```
        "dynamodb:*"
    ],
    "Effect": "Allow",
    "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/Forum",
        "arn:aws:dynamodb:us-west-2:123456789012:table/Thread",
        "arn:aws:dynamodb:us-west-2:123456789012:table/Reply"
    ]
},
{
    "Action": [
        "cloudwatch:DeleteAlarms",
        "cloudwatch:DescribeAlarmHistory",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:DescribeAlarmsForMetric",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:ListMetrics",
        "cloudwatch:PutMetricAlarm",
        "sns:CreateTopic",
        "sns>DeleteTopic",
        "sns:ListSubscriptions",
        "sns:ListSubscriptionsByTopic",
        "sns:ListTopics",
        "sns:Subscribe",
        "sns:Unsubscribe"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Sid": "DDBConsole"
},
```

...remainder of document omitted...

6. Jika pengaturan kebijakan sudah sesuai keinginan Anda, klik **Buat Kebijakan**.

Setelah membuat kebijakan, Anda dapat melampirkannya ke pengguna.

1. Dari Dasbor Konsol IAM, klik **Pengguna** dan pilih pengguna yang ingin Anda ubah.
2. Di tab **Izin**, klik **Lampirkan Kebijakan**.
3. Di panel **Lampirkan Kebijakan**, pilih nama kebijakan Anda dan klik **Lampirkan Kebijakan**.

Note

Anda dapat menggunakan prosedur serupa untuk melampirkan kebijakan kepada peran atau grup, bukan pengguna.

Mengekspor data dari DynamoDB ke Amazon S3

Bagian ini menjelaskan cara mengekspor data dari satu atau beberapa tabel DynamoDB ke bucket Amazon S3. Anda perlu membuat bucket Amazon S3 sebelum dapat melakukan ekspor.

⚠ Important

Jika Anda belum pernah menggunakan AWS Data Pipeline sebelumnya, Anda perlu mengatur dua peran IAM sebelum mengikuti prosedur ini. Untuk informasi selengkapnya, lihat [Membuat Peran IAM untuk AWS Data Pipeline](#).

1. Masuk ke AWS Management Console dan buka AWS Data Pipeline konsol di <https://console.aws.amazon.com/datapipeline/>.
2. Jika Anda belum memiliki jaringan pipa di AWS wilayah saat ini, pilih Mulai sekarang.

Jika Anda sudah memiliki setidaknya satu pipeline, pilih Buat pipeline baru.
3. Pada halaman Buat Pipeline, lakukan hal berikut:
 - a. Di bidang Nama, ketikkan nama untuk pipeline Anda. Misalnya: MyDynamoDBExportPipeline.
 - b. Untuk parameter Sumber, pilih Buat menggunakan templat. Dari daftar templat drop-down, pilih Ekspor tabel DynamoDB ke S3.
 - c. Di bidang Nama tabel DynamoDB sumber, ketik nama tabel DynamoDB yang ingin Anda ekspor.
 - d. Di kotak teks Folder S3 Output, masukkan URI Amazon S3 tempat file ekspor akan ditulis. Misalnya: s3://mybucket/exports

Format URI ini adalah s3://*bucketname*/*folder* dengan:

- *bucketname* adalah nama bucket Amazon S3 Anda.

- `folder` adalah nama folder di dalam bucket tersebut. Jika tidak ada, folder akan dibuat secara otomatis. Jika Anda tidak menentukan nama untuk folder tersebut, nama akan diberikan dalam bentuk `s3://bucketname/region/tablename`.
- e. Di kotak teks Lokasi S3 untuk log, masukkan URI Amazon S3 tempat file log untuk ekspor akan ditulis. Misalnya: `s3://mybucket/logs/`

Format URI untuk Folder Log S3 sama seperti untuk Folder S3 Output. URI harus diselesaikan ke folder; file log tidak dapat ditulis ke tingkat atas bucket S3.

4. Tambahkan tanda dengan Kunci `dynamodbdatapipeline` dan Nilai `true`.
5. Jika pengaturan sudah sesuai keinginan Anda, klik Aktifkan.

Pipeline Anda sekarang akan dibuat; proses ini dapat memakan waktu beberapa menit. Anda dapat memantau kemajuan di AWS Data Pipeline konsol.

Setelah ekspor selesai, Anda dapat membuka [Konsol Amazon S3](#) untuk melihat file ekspor. Nama file output adalah nilai pengidentifikasi tanpa ekstensi, seperti contoh ini: `ae10f955-fb2f-4790-9b11-fbfea01a871e_000000`. Format internal file ini dijelaskan di [Struktur File](#) di Panduan AWS Data Pipeline Pengembang.

Mengimpor data dari Amazon S3 ke DynamoDB

Bagian ini mengasumsikan bahwa Anda telah mengekspor data dari tabel DynamoDB dan file ekspor telah ditulis ke bucket Amazon S3 Anda. Format internal file ini dijelaskan di [Struktur File](#) di Panduan AWS Data Pipeline Pengembang. Perhatikan bahwa ini adalah satu-satunya format file yang DynamoDB dapat mengimpor menggunakan. AWS Data Pipeline

Kami akan menggunakan istilah tabel sumber untuk tabel asal data diekspor dan tabel tujuan untuk tabel yang akan menerima data yang diimpor. Anda dapat mengimpor data dari file ekspor di Amazon S3, dengan ketentuan bahwa semua hal di bawah ini terpenuhi:

- Tabel tujuan sudah ada. (Proses impor tidak akan membuat tabel untuk Anda).
- Tabel tujuan memiliki skema kunci yang sama seperti tabel sumber.

Tabel tujuan tidak harus kosong. Namun, proses impor akan menggantikan setiap item data dalam tabel yang memiliki kunci yang sama seperti item dalam file ekspor. Misalnya, Anda memiliki tabel Pelanggan dengan kunci `CustomerId`, dan hanya ada tiga item dalam tabel (`CustomerId1`, `2`, dan

3). Jika file ekspor Anda juga berisi item data untuk CustomerID 1, 2, dan 3, item dalam tabel tujuan akan diganti dengan item dari file ekspor. Jika file ekspor juga berisi item data untuk CustomerId4, maka item itu akan ditambahkan ke tabel.

Tabel tujuan bisa berada di AWS wilayah yang berbeda. Misalnya, Anda memiliki tabel Pelanggan di wilayah AS Barat (Oregon) dan mengekspor datanya ke Amazon S3. Kemudian, Anda dapat mengimpor data tersebut ke tabel Pelanggan yang sama di wilayah Eropa (Irlandia). Hal ini disebut dengan ekspor dan impor lintas wilayah. Untuk daftar lengkap wilayah AWS, lihat [Wilayah dan titik akhir](#) di Referensi Umum AWS.

Perhatikan bahwa AWS Management Console memungkinkan Anda mengekspor beberapa tabel sumber sekaligus. Namun, Anda hanya dapat mengimpor satu tabel pada satu waktu.

1. Masuk ke AWS Management Console dan buka AWS Data Pipeline konsol di <https://console.aws.amazon.com/datapipeline/>.
2. (Opsional) Jika Anda ingin melakukan impor lintas wilayah, buka sudut kanan atas jendela dan pilih wilayah tujuan.
3. Pilih Buat pipeline baru.
4. Pada halaman Buat Pipeline, lakukan hal berikut:
 - a. Di bidang Nama, ketikkan nama untuk pipeline Anda. Misalnya: `MyDynamoDBImportPipeline`.
 - b. Untuk parameter Sumber, pilih Buat menggunakan templat. Dari daftar templat drop-down, pilih Impor data cadangan DynamoDB dari S3.
 - c. Di kotak teks Folder S3 Input, masukkan URI Amazon S3 tempat file ekspor dapat ditemukan. Misalnya: `s3://mybucket/exports`

Format URI ini adalah `s3://bucketname/folder` dengan:

- `bucketname` adalah nama bucket Amazon S3 Anda.
- `folder` adalah nama folder yang berisi file ekspor.

Pekerjaan impor akan menemukan file di lokasi Amazon S3 yang ditentukan. Format internal file ini dijelaskan di [Verifikasi file ekspor data](#) dalam Panduan Developer AWS Data Pipeline

- d. Di bidang Nama tabel DynamoDB target, ketik nama tabel DynamoDB tujuan Anda ingin mengimpor data.

- e. Di kotak teks Lokasi S3 untuk log, masukkan URI Amazon S3 tempat file log untuk impor akan ditulis. Misalnya: `s3://mybucket/logs/`

Format URI untuk Folder Log S3 sama seperti untuk Folder S3 Output. URI harus diselesaikan ke folder; file log tidak dapat ditulis ke tingkat atas bucket S3.

- f. Tambahkan tanda dengan Kunci `dynamodbdatapipeline` dan Nilai `true`.

5. Jika pengaturan sudah sesuai keinginan Anda, klik Aktifkan.

Pipeline Anda sekarang akan dibuat; proses ini dapat memakan waktu beberapa menit. Pekerjaan impor akan segera dimulai setelah pipeline dibuat.

Memecahkan masalah

Bagian ini membahas beberapa mode kegagalan dasar dan pemecahan masalah untuk ekspor DynamoDB.

Jika terjadi kesalahan selama ekspor atau impor, status pipeline di konsol AWS Data Pipeline akan ditampilkan sebagai ERROR. Jika ini terjadi, klik nama pipeline yang gagal untuk membuka halaman detailnya. Halaman ini akan menampilkan detail tentang semua langkah dalam pipeline dan statusnya masing-masing. Secara khusus, memeriksa setiap jejak tumpukan eksekusi yang Anda lihat.

Terakhir, buka bucket Amazon S3 Anda dan cari file log ekspor atau impor yang ditulis di bucket.

Berikut adalah beberapa masalah umum yang dapat menyebabkan kegagalan pipeline serta tindakan korektif. Untuk mendiagnosis pipeline Anda, bandingkan kesalahan yang Anda lihat dengan masalah yang tercatat di bawah ini.

- Untuk impor, pastikan bahwa tabel tujuan sudah ada dan memiliki skema kunci yang sama seperti tabel sumber. Syarat ini harus dipenuhi, atau impor akan gagal.
- Pastikan bahwa pipeline memiliki tanda `dynamodbdatapipeline`; jika tidak, panggilan API Amazon EMR tidak akan berhasil.
- Pastikan bahwa bucket Amazon S3 yang Anda tentukan telah dibuat dan Anda telah membaca dan menulis izin pada bucket tersebut.
- Pipeline mungkin telah melampaui batas waktu eksekusinya. (Anda menetapkan parameter ini ketika membuat pipeline.) Misalnya, Anda mungkin telah menetapkan batas waktu eksekusi selama 1 jam, tetapi pekerjaan ekspor mungkin memerlukan waktu lebih lama dari ini. Coba untuk

menghapus lalu membuat kembali pipeline, tetapi kali ini dengan interval waktu eksekusi yang lebih lama.

- Perbarui file manifes jika Anda memulihkan dari bucket Amazon S3 yang bukan merupakan bucket asli yang digunakan dalam ekspor (berisi salinan ekspor).
- Anda mungkin tidak memiliki izin yang benar untuk melakukan ekspor atau impor. Untuk informasi selengkapnya, lihat [Prasyarat untuk ekspor dan impor data](#).
- Anda mungkin telah mencapai kuota sumber daya di AWS akun Anda, seperti jumlah maksimum instans Amazon EC2 atau jumlah saluran pipa maksimum. AWS Data Pipeline Untuk informasi selengkapnya, termasuk cara meminta peningkatan kuota, lihat [Kuota layanan AWS](#) di Referensi Umum AWS.

Note

Untuk detail selengkapnya tentang pemecahan masalah pipeline, buka [Pemecahan Masalah](#) dalam Panduan Developer AWS Data Pipeline .

Template yang telah ditetapkan untuk AWS Data Pipeline dan DynamoDB

Jika Anda ingin pemahaman yang lebih dalam tentang cara AWS Data Pipeline kerja, kami sarankan Anda berkonsultasi dengan Panduan AWS Data Pipeline Pengembang. Panduan ini berisi step-by-step tutorial untuk membuat dan bekerja dengan saluran pipa; Anda dapat menggunakan tutorial ini sebagai titik awal untuk membuat saluran pipa Anda sendiri. Sebaiknya Anda membaca tutorial AWS Data Pipeline yang menjelaskan langkah-langkah yang diperlukan untuk membuat pipeline impor dan ekspor yang dapat disesuaikan untuk kebutuhan Anda. Lihat [Tutorial: Impor dan ekspor Amazon DynamoDB menggunakan AWS Data Pipeline](#) dalam Panduan Developer AWS Data Pipeline .

AWS Data Pipeline menawarkan beberapa template untuk membuat pipeline; template berikut ini relevan dengan DynamoDB.

Mengekspor data antara DynamoDB dan Amazon S3

Note

DynamoDB Console sekarang mendukung aliran Ekspor ke Amazon S3 sendiri, namun tidak kompatibel dengan aliran impor. AWS Data Pipeline Untuk informasi selengkapnya, lihat

[Ekspor data DynamoDB ke Amazon S3: cara kerjanya](#) dan postingan blog [Export Amazon DynamoDB table data to your data lake in Amazon S3, no code writing required.](#)

AWS Data Pipeline Konsol menyediakan dua template yang telah ditentukan untuk mengekspor data antara DynamoDB dan Amazon S3. Untuk informasi selengkapnya tentang templat ini, lihat bagian berikut dalam Panduan Developer AWS Data Pipeline :

- [Ekspor DynamoDB ke Amazon S3](#)
- [Ekspor Amazon S3 ke DynamoDB](#)

Backend Penyimpanan Amazon DynamoDB untuk Titan

Backend Penyimpanan DynamoDB untuk proyek Titan telah digantikan oleh Backend Penyimpanan Amazon DynamoDB untuk JanusGraph, yang tersedia di [GitHub](#).

Untuk petunjuk terbaru pada Backend Penyimpanan DynamoDB untuk JanusGraph, lihat file [Readme.md](#).

Disimpan kata-kata di DynamoDB

Kata kunci berikut ini disimpan untuk digunakan oleh DynamoDB. Jangan gunakan istilah-istilah ini sebagai nama atribut dalam pernyataan. Daftar ini tidak peka terhadap huruf besar/kecil.

Jika Anda harus menulis pernyataan yang berisi nama atribut yang bertentangan dengan istilah yang disimpan DynamoDB, Anda dapat menentukan nama atribut pernyataan untuk digunakan sebagai pengganti istilah yang disimpan. Untuk informasi selengkapnya, lihat [Nama atribut ekspresi di DynamoDB](#).

```
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
SPECIFICITYTYPE
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

Parameter bersyarat lama

Bagian ini membandingkan parameter bersyarat legasi dengan parameter ekspresi di DynamoDB.

⚠ Important

Kami menyarankan Anda menggunakan parameter ekspresi baru alih-alih parameter lama ini bila memungkinkan. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#).

Selain itu, DynamoDB tidak mengizinkan pencampuran parameter kondisional lama dan parameter ekspresi dalam satu panggilan. Sebagai contoh, memanggil operasi Query dengan `AttributesToGet` dan `ConditionExpression` akan menimbulkan kesalahan.

Tabel berikut menunjukkan operasi DynamoDB API yang masih mendukung parameter lama ini, dan parameter ekspresi mana yang akan digunakan sebagai gantinya. Tabel ini dapat membantu jika Anda akan memperbarui aplikasi Anda agar aplikasi tersebut menggunakan parameter ekspresi sebagai gantinya.

Jika Anda menggunakan operasi API ini...	Dengan parameter warisan ini...	Gunakan parameter ekspresi ini sebagai gantinya
BatchGetItem	AttributesToGet	ProjectionExpression
DeleteItem	Expected	ConditionExpression
GetItem	AttributesToGet	ProjectionExpression
PutItem	Expected	ConditionExpression
Query	AttributesToGet	ProjectionExpression
	KeyConditions	KeyConditionExpression
	QueryFilter	FilterExpression
Scan	AttributesToGet	ProjectionExpression
	ScanFilter	FilterExpression
UpdateItem	AttributeUpdates	UpdateExpression

Jika Anda menggunakan operasi API ini...	Dengan parameter warisan ini...	Gunakan parameter ekspresi ini sebagai gantinya
	Expected	ConditionExpression

Bagian berikut menyediakan informasi lebih lanjut tentang parameter bersyarat legasi.

Topik

- [AttributesToGet \(warisan\)](#)
- [AttributeUpdates \(warisan\)](#)
- [ConditionalOperator \(warisan\)](#)
- [Diharapkan \(warisan\)](#)
- [KeyConditions \(warisan\)](#)
- [QueryFilter \(warisan\)](#)
- [ScanFilter \(warisan\)](#)
- [Kondisi penulisan dengan parameter lama](#)

AttributesToGet (warisan)

Note

Kami menyarankan Anda menggunakan parameter ekspresi baru alih-alih parameter lama ini bila memungkinkan. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#). Untuk informasi spesifik tentang parameter baru menggantikan yang satu ini, [gunakan ProjectionExpression sebagai gantinya](#).

Parameter kondisional warisan `AttributesToGet` adalah array dari satu atau lebih atribut untuk mengambil dari DynamoDB. Jika tidak ada nama atribut yang disediakan, maka semua atribut akan dikembalikan. Jika salah satu atribut yang diminta tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil.

`AttributesToGet` memungkinkan Anda untuk mengambil atribut dari tipe Daftar atau Peta; namun, tidak dapat mengambil elemen individu dalam Daftar atau Peta.

Perhatikan bahwa `AttributesToGet` tidak berpengaruh pada konsumsi throughput yang disediakan. DynamoDB menentukan unit kapasitas yang digunakan berdasarkan ukuran item, bukan jumlah data yang dikembalikan ke aplikasi.

Gunakan `ProjectionExpression` sebagai gantinya - Contoh

Misalkan Anda ingin mengambil item dari tabel Musik, tapi Anda hanya ingin mengembalikan beberapa atribut. Anda dapat menggunakan permintaan `GetItem` dengan parameter `AttributesToGet`, seperti dalam contoh AWS CLI ini:

```
aws dynamodb get-item \  
  --table-name Music \  
  --attributes-to-get '["Artist", "Genre"]' \  
  --key '{  
    "Artist": {"S": "No One You Know"},  
    "SongTitle": {"S": "Call Me Today"}  
  }'
```

Anda dapat menggunakan sebagai `ProjectionExpression` gantinya:

```
aws dynamodb get-item \  
  --table-name Music \  
  --projection-expression "Artist, Genre" \  
  --key '{  
    "Artist": {"S": "No One You Know"},  
    "SongTitle": {"S": "Call Me Today"}  
  }'
```

AttributeUpdates (warisan)

Note

Kami menyarankan Anda menggunakan parameter ekspresi baru alih-alih parameter lama ini bila memungkinkan. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#). Untuk informasi spesifik tentang parameter baru menggantikan yang satu ini, [gunakan UpdateExpression sebagai gantinya..](#)

Dalam sebuah `UpdateItem` operasi, parameter bersyarat lama `AttributeUpdates` berisi nama-nama atribut yang akan dimodifikasi, tindakan yang harus dilakukan pada masing-masing, dan nilai

baru untuk masing-masing. Jika Anda memperbarui atribut yang merupakan atribut kunci indeks untuk setiap indeks pada tabel tersebut, tipe atribut harus sesuai dengan tipe kunci indeks yang didefinisikan dalam `AttributesDefinition` pada deskripsi tabel. Anda dapat menggunakan `UpdateItem` untuk memperbarui setiap atribut non-kunci.

Nilai atribut tidak boleh nol. Atribut tipe biner dan string harus memiliki panjang lebih dari nol. Set tipe atribut tidak boleh kosong. Permintaan dengan nilai kosong akan ditolak dengan pengecualian `ValidationException`.

Setiap elemen `AttributeUpdates` terdiri dari nama atribut untuk memodifikasi, beserta hal berikut:

- `Value` - Nilai baru, jika berlaku, untuk atribut ini.
- `Action` - Sebuah nilai yang menentukan cara melakukan pembaruan. Tindakan ini hanya berlaku untuk atribut yang ada yang tipe data adalah Angka atau suatu set; jangan gunakan `ADD` untuk tipe data lainnya.

Jika item dengan kunci utama tertentu ditemukan dalam tabel, nilai berikut melakukan tindakan berikut:

- `PUT` - Menambahkan atribut tertentu untuk item. Jika sudah ada, atribut akan digantikan oleh nilai baru.
- `DELETE` - Menghapus atribut dan nilainya, jika tidak ada nilai yang ditentukan untuk `DELETE`. Tipe data dari nilai yang ditentukan harus sesuai dengan tipe data dari nilai yang ada.

Jika suatu set nilai ditentukan, maka nilai-nilai tersebut dikurangi dari set lama. Sebagai contoh, jika nilai atribut adalah himpunan `[a, b, c]` dan tindakan `DELETE` menentukan `[a, c]`, maka nilai atribut akhir adalah `[b]`. Menentukan satu set kosong adalah kesalahan.

- `ADD` - Menambahkan nilai yang ditentukan untuk item, jika atribut belum ada. Jika atribut tidak ada, maka perilaku `ADD` tergantung pada tipe data atribut:
 - Jika atribut yang ada adalah angka, dan jika `Value` juga berupa angka, maka `Value` secara matematis ditambahkan ke atribut yang ada. Jika `Value` adalah angka negatif, nilai tersebut dikurangi dari atribut yang ada.

Note

Jika Anda menggunakan `ADD` untuk menambah atau mengurangi nilai angka untuk item yang tidak ada sebelum pembaruan, DynamoDB menggunakan 0 sebagai nilai awal.

Sama halnya jika Anda menggunakan ADD untuk item yang ada untuk menambah atau mengurangi nilai atribut yang tidak ada sebelum pembaruan, DynamoDB menggunakan 0 sebagai nilai awal. Misalnya, anggap item yang ingin Anda perbarui tidak memiliki atribut bernama itemcount, tapi Anda memutuskan untuk tetap ADD angka 3 ke atribut ini. DynamoDB akan membuat atribut itemcount, mengatur nilai awal untuk 0, dan akhirnya menambahkan 3 ke nilai tersebut. Hasilnya akan menjadi atribut itemcount baru, dengan nilai 3.

- Jika tipe data yang ada adalah suatu set, dan jika Value juga suatu set, maka Value ditambahkan ke set yang ada. Misalnya, jika nilai atribut adalah himpunan [1, 2], dan tindakan ADD menentukan [3], maka nilai atribut akhir adalah [1, 2, 3]. Terjadi kesalahan jika tindakan ADD ditentukan untuk atribut set dan tipe atribut yang ditentukan tidak cocok dengan tipe set yang ada.

Kedua set harus memiliki tipe data primitif yang sama. Sebagai contoh, jika tipe data yang ada adalah suatu set string, Value juga harus berupa suatu set string.

Jika item dengan kunci tertentu ditemukan dalam tabel, nilai berikut melakukan tindakan berikut:

- PUT - Penyebab DynamoDB membuat item baru dengan kunci utama yang ditentukan, lalu menambahkan atribut.
- DELETE - Tidak ada yang terjadi, karena atribut tidak dapat dihapus dari item yang tidak ada. Operasi berhasil, tapi DynamoDB tidak membuat item baru.
- ADD - Menyebabkan DynamoDB membuat item dengan kunci utama yang disediakan dan angka (atau set angka) untuk nilai atribut. Satu-satunya tipe data yang diperbolehkan adalah Angka dan Set Angka.

Jika Anda memberikan atribut yang merupakan bagian dari kunci indeks, maka tipe data untuk atribut tersebut harus sesuai dengan skema dalam definisi atribut tabel.

Gunakan UpdateExpression sebagai gantinya - Contoh

Misalkan Anda ingin memodifikasi item dalam tabel Musik. Anda dapat menggunakan permintaan UpdateItem dengan parameter AttributeUpdates, seperti dalam contoh AWS CLI ini:

```
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"}
    }
}'
```

Anda dapat menggunakan sebagai UpdateExpression gantinya:

```
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"}
}'
```

Untuk informasi selengkapnya tentang memperbarui atribut, lihat [Memperbarui item dalam tabel DynamoDB](#).

ConditionalOperator (warisan)

Note

Kami menyarankan Anda menggunakan parameter ekspresi baru alih-alih parameter lama ini bila memungkinkan. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#).

Parameter bersyarat lama ConditionalOperator adalah operator logis yang digunakan untuk menerapkan kondisi dalam Expected, QueryFilter atau ScanFilter peta:

- AND - Jika semua syarat bernilai true, maka seluruh peta bernilai true.
- ATAU - Jika setidaknya salah satu kondisi dievaluasi menjadi benar, maka seluruh peta mengevaluasi menjadi benar.

Jika Anda menghilangkan `ConditionalOperator`, `AND` adalah pengaturan default.

Operasi akan berhasil hanya jika seluruh peta bernilai `true`.

Note

Parameter ini tidak mendukung atribut tipe Daftar atau Peta.

Diharapkan (warisan)

Note

Kami menyarankan Anda menggunakan parameter ekspresi baru alih-alih parameter lama ini bila memungkinkan. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#). Untuk informasi spesifik tentang parameter baru menggantikan yang satu ini, [gunakan ConditionExpression sebagai gantinya](#).

Parameter bersyarat lama `Expected` adalah blok bersyarat untuk operasi. `UpdateItem Expected` adalah peta pasangan atribut/kondisi. Setiap elemen peta terdiri dari nama atribut, operator perbandingan, dan satu atau beberapa nilai. DynamoDB membandingkan atribut dengan nilai(-nilai) yang Anda berikan, menggunakan operator perbandingan. Untuk setiap elemen `Expected`, hasil evaluasi adalah `true` atau `false`.

Jika Anda menentukan lebih dari satu elemen dalam peta `Expected`, maka secara default semua syarat harus bernilai `true`. Dengan kata lain, semua syarat dibuat menjadi `AND`. (Anda dapat menggunakan parameter `ConditionalOperator` untuk syarat `OR` sebagai gantinya. Jika Anda melakukan ini, maka setidaknya salah satu syarat harus bernilai `true`, bukan semuanya.)

Jika peta `Expected` bernilai `true`, maka operasi bersyarat berhasil; jika tidak, gagal.

`Expected` berisi hal berikut ini:

- `AttributeValueList` - Satu atau beberapa nilai untuk mengevaluasi atribut yang disediakan. Jumlah nilai dalam daftar tergantung pada `ComparisonOperator` yang sedang digunakan.

Untuk tipe Angka, perbandingan nilainya bersifat numerik.

Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada Unicode dengan UTF-8 encoding biner. Sebagai contoh, a lebih besar dari A, dan a lebih besar dari B.

Untuk jenis Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner.

- `ComparisonOperator` - Sebuah pembandingan untuk mengevaluasi atribut dalam `AttributeValueList`. Saat melakukan perbandingan, DynamoDB menggunakan pembacaan yang sangat konsisten.

Tersedia operator perbandingan berikut ini:

`EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

Berikut adalah deskripsi setiap operator perbandingan.

- `EQ`: Sama. `EQ` didukung untuk semua tipe data, termasuk daftar dan peta.

`AttributeValueList` hanya dapat berisi satu tipe elemen `AttributeValue` dari tipe String, Angka, Biner, Set String, Set Angka, atau Set Biner. Jika item berisi elemen `AttributeValue` dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Sebagai contoh, `{"S": "6"}` tidak sama dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sama dengan `{"NS": ["6", "2", "1"]}`.

- `NE`: Tidak sama. `NE` didukung untuk semua tipe data, termasuk daftar dan peta.

`AttributeValueList` hanya dapat berisi satu `AttributeValue` dari tipe String, Angka, Biner, Set String, Set Angka, atau Set Biner. Jika item berisi `AttributeValue` dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Misalnya, `{"S": "6"}` tidak sama dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sama dengan `{"NS": ["6", "2", "1"]}`.

- `LE` : Kurang dari atau sama dengan.

`AttributeValueList` hanya dapat berisi satu elemen `AttributeValue` dari tipe String, Angka, atau Biner (bukan tipe set). Jika item berisi elemen `AttributeValue` dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Misalnya, `{"S": "6"}` tidak sama dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.

- LT : Kurang dari.

`AttributeValueList` hanya dapat berisi satu `AttributeValue` dari tipe `String`, `Angka`, atau `Biner` (bukan tipe `set`). Jika item berisi elemen `AttributeValue` dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Misalnya, `{"S": "6"}` tidak sama dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.

- GE : Lebih besar dari atau sama dengan.

`AttributeValueList` hanya dapat berisi satu elemen `AttributeValue` dari tipe `String`, `Angka`, atau `Biner` (bukan tipe `set`). Jika item berisi elemen `AttributeValue` dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Misalnya, `{"S": "6"}` tidak sama dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.

- GT : Lebih besar dari.

`AttributeValueList` hanya dapat berisi satu elemen `AttributeValue` dari tipe `String`, `Angka`, atau `Biner` (bukan tipe `set`). Jika item berisi elemen `AttributeValue` dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Misalnya, `{"S": "6"}` tidak sama dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.

- NOT_NULL: Atribut ada. NOT_NULL didukung untuk semua tipe data, termasuk daftar dan peta.

Note

Operator ini menguji keberadaan atribut, bukan tipe data. Jika tipe data atribut "a" adalah `null`, dan Anda mengevaluasinya menggunakan NOT_NULL, hasilnya adalah Boolean `true`. Hasil ini karena atribut "a" ada; tipe data tidak relevan dengan operator perbandingan NOT_NULL.

- NULL: Atribut tidak ada. NULL didukung untuk semua tipe data, termasuk daftar dan peta.

Note

Operator ini menguji ketiadaan atribut, bukan tipe data. Jika tipe data atribut "a" adalah `null`, dan Anda mengevaluasinya menggunakan NULL, hasilnya adalah Boolean `false`. Hal ini karena ada atribut "a"; tipe data tidak relevan dengan operator perbandingan NULL.

- **CONTAINS** : Memeriksa urutan, atau nilai dalam suatu set.

`AttributeValueList` hanya dapat berisi satu elemen `AttributeValue` dari tipe `String`, `Angka`, atau `Biner` (bukan tipe set). Jika atribut target perbandingan adalah dari tipe `String`, maka cek operator untuk pertandingan substring. Jika atribut target perbandingan adalah dari tipe `Biner`, maka operator mencari urutan dari target yang cocok dengan input. Jika atribut target perbandingan adalah suatu set ("`SS`", "`NS`", atau "`BS`"), maka operator bernilai `true` jika menemukan yang sama persis dengan anggota set tersebut.

`CONTAINS` didukung untuk daftar: Saat mengevaluasi "`a CONTAINS b`", "`a`" dapat menjadi suatu daftar; namun, "`b`" tidak dapat menjadi set, peta, atau daftar.

- **NOT_CONTAINS** : Memeriksa tidak adanya urutan, atau tidak adanya nilai dalam suatu set.

`AttributeValueList` hanya dapat berisi satu elemen `AttributeValue` dari tipe `String`, `Angka`, atau `Biner` (bukan tipe set). Jika atribut target perbandingan adalah `String`, maka operator memeriksa tidak adanya kecocokan substring. Jika atribut target perbandingan adalah `Biner`, maka operator mencari tidak adanya urutan pada target yang cocok dengan input. Jika atribut target perbandingan adalah suatu set ("`SS`", "`NS`", atau "`BS`"), maka operator bernilai `true` jika `does not` menemukan yang sama persis dengan anggota set tersebut.

`NOT_COINTAINS` didukung untuk daftar: Ketika mengevaluasi "`a NOT CONTAINS b`", "`a`" dapat menjadi daftar; namun, "`b`" tidak dapat menjadi set, peta, atau daftar.

- **BEGINS_WITH** : Memeriksa awalan.

`AttributeValueList` hanya dapat berisi satu `AttributeValue` dari tipe `String` atau `Biner` (bukan `Angka` atau tipe set). Atribut target perbandingan harus dari tipe `String` atau `Biner` (bukan `Angka` atau set).

- **IN** : Pemeriksaan elemen yang cocok dalam dua set.

`AttributeValueList` dapat berisi satu atau beberapa elemen `AttributeValue` dari tipe `String`, `Angka`, atau `Biner` (bukan tipe set). Atribut ini dibandingkan dengan atribut tipe set yang ada dari suatu item. Jika setiap elemen dari set input ada dalam atribut item, ekspresi bernilai `true`.

- **BETWEEN**: Lebih besar dari atau sama dengan nilai pertama, dan kurang dari atau sama dengan nilai kedua.

`AttributeValueList` harus berisi dua elemen `AttributeValue` dari tipe yang sama, baik `String`, `Angka`, atau `Biner` (bukan tipe set). Suatu atribut target cocok jika nilai target lebih besar

dari, atau sama dengan, elemen pertama dan kurang dari, atau sama dengan, elemen kedua. Jika item berisi elemen `AttributeValue` dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Misalnya, `{"S": "6"}` tidak sebanding dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`

Parameter berikut dapat digunakan sebagai pengganti `AttributeValueList` dan `ComparisonOperator`:

- `Value` - Nilai untuk DynamoDB guna membandingkan dengan atribut.
- `Exists` - Nilai Boolean yang menyebabkan DynamoDB mengevaluasi nilai sebelum mencoba operasi bersyarat:
 - Jika `Exists` adalah `true`, DynamoDB akan memeriksa untuk melihat apakah nilai atribut sudah ada dalam tabel. Jika ditemukan, maka kondisinya dievaluasi menjadi benar; jika tidak, kondisinya akan salah.
 - Jika `Exists` adalah `false`, DynamoDB mengasumsikan bahwa nilai atribut tidak ada dalam tabel. Jika pada dasarnya nilai tidak ada, maka asumsi tersebut valid dan syarat bernilai `true`. Jika nilainya ditemukan, terlepas dari asumsi bahwa nilai itu tidak ada, syaratnya bernilai salah.

Perhatikan bahwa nilai default untuk `Exists` adalah `true`.

Parameter `Value` dan `Exists` tidak kompatibel dengan `AttributeValueList` dan `ComparisonOperator`. Perhatikan bahwa jika Anda menggunakan kedua set parameter sekaligus, DynamoDB akan mengembalikan pengecualian `ValidationException`.

Note

Parameter ini tidak mendukung atribut tipe Daftar atau Peta.

Gunakan `ConditionExpression` sebagai gantinya - Contoh

Misalkan Anda ingin memodifikasi item dalam tabel Musik, tetapi hanya jika syarat tertentu benar. Anda dapat menggunakan permintaan `UpdateItem` dengan parameter `Expected`, seperti dalam contoh AWS CLI ini:

```
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"} ]
  }
}'

```

Anda dapat menggunakan sebagai ConditionExpression gantinya:

```

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 (warisan)

Note

Kami menyarankan Anda menggunakan parameter ekspresi baru alih-alih parameter lama ini bila memungkinkan. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#). Untuk informasi spesifik tentang parameter baru menggantikan yang satu ini, [gunakan KeyConditionExpression sebagai gantinya..](#)

Parameter bersyarat lama `KeyConditions` berisi kriteria seleksi untuk operasi. Query Untuk kueri pada tabel, Anda dapat memiliki syarat hanya pada tabel atribut kunci utama. Anda harus memberikan nama kunci partisi dan nilai sebagai syarat EQ. Anda dapat memberikan syarat kedua secara opsional, dengan merujuk pada kunci pengurutan.

 Note

Jika Anda tidak memberikan syarat kunci pengurutan, semua item yang cocok dengan kunci partisi akan diambil. Jika ada, `FilterExpression` atau `QueryFilter` akan diterapkan setelah item diambil.

Untuk kueri pada indeks, Anda dapat memiliki syarat hanya pada atribut kunci indeks. Anda harus memberikan nama kunci partisi indeks dan nilai sebagai syarat EQ. Anda dapat memberikan syarat kedua secara opsional, dengan merujuk pada kunci pengurutan indeks.

Setiap elemen `KeyConditions` terdiri dari nama atribut untuk membandingkan, beserta hal berikut:

- `AttributeValueList` - Satu atau beberapa nilai untuk mengevaluasi atribut yang disediakan. Jumlah nilai dalam daftar tergantung pada `ComparisonOperator` yang sedang digunakan.

Untuk tipe Angka, perbandingan nilainya bersifat numerik.

Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada Unicode dengan UTF-8 encoding biner. Sebagai contoh, a lebih besar dari A, dan a lebih besar dari B.

Untuk Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner.

- `ComparisonOperator` - Sebuah pembanding untuk mengevaluasi atribut. Misalnya: sama, lebih besar dari, dan kurang dari.

Untuk `KeyConditions`, hanya operator perbandingan berikut yang didukung:

EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN

Berikut adalah deskripsi operator perbandingan ini.

- EQ : Sama dengan.

`AttributeValueList` hanya dapat berisi satu `AttributeValue` dari tipe `String`, `Angka`, atau `Biner` (bukan tipe `set`). Jika item berisi elemen `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, `{"S": "6"}` tidak sama dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sama dengan `{"NS": ["6", "2", "1"]}`.

- `LE` : Kurang dari atau sama dengan.

`AttributeValueList` hanya dapat berisi satu elemen `AttributeValue` dari tipe `String`, `Angka`, atau `Biner` (bukan tipe `set`). Jika item berisi elemen `AttributeValue` dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Misalnya, `{"S": "6"}` tidak sama dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.

- `LT` : Kurang dari.

`AttributeValueList` hanya dapat berisi satu `AttributeValue` dari tipe `String`, `Angka`, atau `Biner` (bukan tipe `set`). Jika item berisi elemen `AttributeValue` dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Misalnya, `{"S": "6"}` tidak sama dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.

- `GE` : Lebih besar dari atau sama dengan.

`AttributeValueList` hanya dapat berisi satu elemen `AttributeValue` dari tipe `String`, `Angka`, atau `Biner` (bukan tipe `set`). Jika item berisi elemen `AttributeValue` dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Misalnya, `{"S": "6"}` tidak sama dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.

- `GT` : Lebih besar dari.

`AttributeValueList` hanya dapat berisi satu elemen `AttributeValue` dari tipe `String`, `Angka`, atau `Biner` (bukan tipe `set`). Jika item berisi elemen `AttributeValue` dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Misalnya, `{"S": "6"}` tidak sama dengan `{"N": "6"}`. Selain itu, `{"N": "6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.

- `BEGINS_WITH` : Memeriksa awalan.

`AttributeValueList` hanya dapat berisi satu `AttributeValue` dari tipe `String` atau `Biner` (bukan `Angka` atau tipe `set`). Atribut target perbandingan harus dari tipe `String` atau `Biner` (bukan `Angka` atau `set`).

- BETWEEN: Lebih besar dari atau sama dengan nilai pertama, dan kurang dari atau sama dengan nilai kedua.

AttributeValueList harus berisi dua elemen AttributeValue dari tipe yang sama, baik String, Angka, atau Biner (bukan tipe set). Suatu atribut target cocok jika nilai target lebih besar dari, atau sama dengan, elemen pertama dan kurang dari, atau sama dengan, elemen kedua. Jika item berisi elemen AttributeValue dengan tipe berbeda dari yang disediakan dalam permintaan, nilai tidak cocok. Misalnya, {"S": "6"} tidak sebanding dengan {"N": "6"}. Selain itu, {"N": "6"} tidak sebanding dengan {"NS": ["6", "2", "1"]}

Gunakan KeyConditionExpression sebagai gantinya - Contoh

Misalkan Anda ingin mengambil beberapa item dengan kunci partisi yang sama dari tabel Musik. Anda dapat menggunakan permintaan Query dengan parameter KeyConditions, seperti dalam contoh AWS CLI ini:

```
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"} ]  
    }  
  }'
```

Anda dapat menggunakan sebagai KeyConditionExpression gantinya:

```
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 (warisan)

Note

Kami menyarankan Anda menggunakan parameter ekspresi baru alih-alih parameter lama ini bila memungkinkan. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#). Untuk informasi spesifik tentang parameter baru menggantikan yang satu ini, [gunakan FilterExpression sebagai gantinya](#).

Dalam Query operasi, parameter bersyarat lama `QueryFilter` adalah kondisi yang mengevaluasi hasil kueri setelah item dibaca dan hanya mengembalikan nilai yang diinginkan.

Parameter ini tidak mendukung atribut tipe Daftar atau Peta.

Note

`QueryFilter` diterapkan setelah item dibaca; proses penyaringan tidak menggunakan unit kapasitas baca tambahan.

Jika Anda memberikan lebih dari satu syarat di peta `QueryFilter`, maka secara default semua syarat harus bernilai true. Dengan kata lain, semua syarat dibuat menjadi AND. (Anda dapat menggunakan parameter [ConditionalOperator \(warisan\)](#) untuk syarat OR sebagai gantinya. Jika Anda melakukan ini, maka setidaknya salah satu syarat harus bernilai true, bukan semuanya.)

Perhatikan bahwa `QueryFilter` tidak mengizinkan atribut kunci. Anda tidak dapat menentukan syarat filter pada kunci partisi atau kunci pengurutan.

Setiap elemen `QueryFilter` terdiri dari nama atribut untuk membandingkan, beserta hal berikut:

- `AttributeValueList` - Satu atau beberapa nilai untuk mengevaluasi atribut yang disediakan. Jumlah nilai dalam daftar tergantung pada operator yang ditentukan dalam `ComparisonOperator`.

Untuk tipe Angka, perbandingan nilainya bersifat numerik.

Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada encoding biner UTF-8. Sebagai contoh, a lebih besar dari A, dan a lebih besar dari B.

Untuk jenis Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner.

Untuk informasi tentang menentukan tipe data di JSON, lihat [API tingkat rendah DynamoDB](#).

- `ComparisonOperator` - Sebuah pembanding untuk mengevaluasi atribut. Misalnya: sama, lebih besar dari, dan kurang dari.

Tersedia operator perbandingan berikut ini:

EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN

Gunakan FilterExpression sebagai gantinya - Contoh

Misalkan Anda ingin mengajukan kueri tabel Musik dan menerapkan syarat untuk item yang cocok. Anda dapat menggunakan permintaan Query dengan parameter `QueryFilter`, seperti dalam contoh AWS CLI ini:

```
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"} ]
    }
  }'
```

Anda dapat menggunakan sebagai `FilterExpression` gantinya:

```
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 (warisan)

Note

Kami menyarankan Anda menggunakan parameter ekspresi baru alih-alih parameter lama ini bila memungkinkan. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#). Untuk informasi spesifik tentang parameter baru menggantikan yang satu ini, [gunakan FilterExpression sebagai gantinya..](#)

Dalam Scan operasi, parameter bersyarat lama `ScanFilter` adalah kondisi yang mengevaluasi hasil pemindaian dan hanya mengembalikan nilai yang diinginkan.

Note

Parameter ini tidak mendukung atribut tipe Daftar atau Peta.

Jika Anda menentukan lebih dari satu syarat di peta `ScanFilter`, maka secara default semua syarat harus bernilai true. Dengan kata lain, semua syarat dibuat menjadi AND. (Anda dapat menggunakan parameter [ConditionalOperator \(warisan\)](#) untuk syarat OR sebagai gantinya. Jika Anda melakukan ini, maka setidaknya salah satu syarat harus bernilai true, bukan semuanya.)

Setiap elemen `ScanFilter` terdiri dari nama atribut untuk membandingkan, beserta hal berikut:

- `AttributeValueList` - Satu atau beberapa nilai untuk mengevaluasi atribut yang disediakan. Jumlah nilai dalam daftar tergantung pada operator yang ditentukan dalam `ComparisonOperator`.

Untuk tipe Angka, perbandingan nilainya bersifat numerik.

Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada encoding biner UTF-8. Sebagai contoh, a lebih besar dari A, dan a lebih besar dari B.

Untuk Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner.

Untuk informasi tentang menentukan tipe data di JSON, lihat [API tingkat rendah DynamoDB](#).

- `ComparisonOperator` - Sebuah pembandingan untuk mengevaluasi atribut. Misalnya: sama, lebih besar dari, dan kurang dari.

Tersedia operator perbandingan berikut ini:

EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
BEGINS_WITH | IN | BETWEEN

Gunakan FilterExpression sebagai gantinya - Contoh

Misalkan Anda ingin memindai tabel Musik dan menerapkan syarat untuk item yang cocok. Anda dapat menggunakan permintaan Scan dengan parameter `ScanFilter`, seperti dalam contoh AWS CLI ini:

```
aws dynamodb scan \  
  --table-name Music \  
  --scan-filter '{  
    "Genre":{  
      "AttributeValueList":[ {"S":"Rock"} ],  
      "ComparisonOperator": "EQ"  
    }  
  }'
```

Anda dapat menggunakan sebagai `FilterExpression` gantinya:

```
aws dynamodb scan \  
  --table-name Music \  
  --filter-expression 'Genre = :g' \  
  --expression-attribute-values '{  
    ":g": {"S":"Rock"}  
  }'
```

Kondisi penulisan dengan parameter lama

Note

Kami menyarankan Anda menggunakan parameter ekspresi baru alih-alih parameter lama ini bila memungkinkan. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#).

Bagian berikut menjelaskan cara menulis syarat untuk digunakan dengan parameter legasi, seperti `Expected`, `QueryFilter`, dan `ScanFilter`.

Note

Aplikasi baru harus menggunakan parameter ekspresi sebagai gantinya. Untuk informasi selengkapnya, lihat [Menggunakan ekspresi di DynamoDB](#).

Kondisi sederhana

Dengan nilai atribut, Anda dapat menulis syarat untuk perbandingan terhadap atribut tabel. Syarat selalu bernilai `true` atau `false`, dan terdiri dari:

- `ComparisonOperator` – lebih besar dari, kurang dari, sama dengan, dan sebagainya.
- `AttributeValueList` (opsional) – nilai atribut untuk dibandingkan. Tergantung pada `ComparisonOperator` yang sedang digunakan, `AttributeValueList` mungkin berisi satu, dua, atau lebih nilai; atau mungkin tidak ada sama sekali.

Bagian berikut menjelaskan berbagai operator perbandingan, beserta contoh cara menggunakannya dalam syarat.

Operator perbandingan tanpa nilai atribut

- `NOT_NULL` - benar jika atribut ada.
- `NULL` - benar jika atribut tidak ada.

Gunakan operator ini untuk memeriksa apakah atribut ada, atau tidak ada. Karena tidak ada nilai untuk dibandingkan, jangan tentukan `AttributeValueList`.

Contoh

Ekspresi berikut bernilai true jika ada atribut Dimensi.

```
...
  "Dimensions": {
    ComparisonOperator: "NOT_NULL"
  }
...
```

Operator perbandingan dengan satu nilai atribut

- EQ - benar jika atribut sama dengan suatu nilai.

`AttributeValueList` hanya dapat berisi satu nilai dari tipe String, Angka, Biner, Set String, Set Angka, atau Set Biner. Jika item berisi nilai dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, string "3" tidak sama dengan angka 3. Selain itu, angka 3 tidak sama dengan set angka [3, 2, 1].

- NE - benar jika atribut tidak sama dengan suatu nilai.

`AttributeValueList` hanya dapat berisi satu nilai dari tipe String, Angka, Biner, Set String, Set Angka, atau Set Biner. Jika item berisi nilai dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok.

- LE - benar jika atribut kurang dari atau sama dengan suatu nilai.

`AttributeValueList` hanya dapat berisi satu nilai dari tipe String, Angka, atau Biner (bukan suatu set). Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok.

- LT - benar jika atribut kurang dari suatu nilai.

`AttributeValueList` hanya dapat berisi satu nilai dari tipe String, Angka, atau Biner (bukan suatu set). Jika item berisi nilai dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok.

- GE - benar jika atribut lebih besar dari atau sama dengan suatu nilai.

`AttributeValueList` hanya dapat berisi satu nilai dari tipe String, Angka, atau Biner (bukan suatu set). Jika item berisi nilai dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok.

- GT - benar jika atribut lebih besar dari suatu nilai.

`AttributeValueList` hanya dapat berisi satu nilai dari tipe String, Angka, atau Biner (bukan suatu set). Jika item berisi nilai dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok.

- `CONTAINS` - benar jika nilai ada dalam suatu set, atau jika satu nilai berisi nilai lain.

`AttributeValueList` hanya dapat berisi satu nilai dari tipe String, Angka, atau Biner (bukan suatu set). Jika atribut target perbandingan adalah String, maka operator memeriksa kecocokan substring. Jika atribut target perbandingan adalah Biner, maka operator mencari urutan dari target yang cocok dengan input. Jika atribut target perbandingan adalah suatu set, maka operator bernilai `true` jika menemukan yang sama persis dengan anggota set.

- `NOT_CONTAINS` - benar jika nilai tidak ada dalam suatu set, atau jika satu nilai tidak berisi nilai lain.

`AttributeValueList` hanya dapat berisi satu nilai dari tipe String, Angka, atau Biner (bukan suatu set). Jika atribut target perbandingan adalah String, maka operator memeriksa tidak adanya kecocokan substring. Jika atribut target perbandingan adalah Biner, maka operator mencari tidak adanya urutan pada target yang cocok dengan input. Jika atribut target perbandingan adalah suatu set, maka operator akan bernilai `true` jika tidak menemukan pasangan yang sama persis dengan anggota set.

- `BEGINS_WITH` - benar jika beberapa karakter pertama atribut cocok dengan nilai yang disediakan. Jangan gunakan operator ini untuk membandingkan angka.

`AttributeValueList` hanya dapat berisi satu nilai dari tipe String atau Biner (bukan Angka atau suatu set). Atribut target perbandingan harus berupa String atau Biner (bukan Angka atau suatu set).

Gunakan operator ini untuk membandingkan atribut dengan nilai. Anda harus menentukan `AttributeValueList` yang berisi nilai tunggal. Untuk sebagian besar operator, nilai ini harus berupa skalar; namun, operator `EQ` dan `NE` juga mendukung himpunan.

Contoh

Ekspresi berikut bernilai `true` jika:

- Harga produk lebih besar dari 100.

```
...  
  "Price": {
```

```
    ComparisonOperator: "GT",
    AttributeValueList: [ {"N":"100"} ]
  }
  ...
```

- Kategori produk dimulai dengan "Bo".

```
...
  "ProductCategory": {
    ComparisonOperator: "BEGINS_WITH",
    AttributeValueList: [ {"S":"Bo"} ]
  }
  ...
```

- Produk tersedia dalam warna merah, hijau, atau hitam:

```
...
  "Color": {
    ComparisonOperator: "EQ",
    AttributeValueList: [
      [ {"S":"Black"}, {"S":"Red"}, {"S":"Green"} ]
    ]
  }
  ...
```

Note

Saat membandingkan jenis data set, urutan elemen tidak masalah. DynamoDB hanya akan mengembalikan item dengan set nilai yang sama, terlepas dari urutan yang Anda tentukan dalam permintaan Anda.

Operator perbandingan dengan dua nilai atribut

- BETWEEN - benar jika nilai adalah antara batas bawah dan batas atas, titik akhir inklusif.

`AttributeValueList` harus berisi dua elemen dari tipe yang sama, baik String, Angka, atau Biner (bukan suatu set). Suatu atribut target cocok jika nilai target lebih besar dari, atau sama dengan, elemen pertama dan kurang dari, atau sama dengan, elemen kedua. Jika item berisi nilai dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok.

Gunakan operator ini untuk menentukan apakah nilai atribut termasuk dalam kisaran. `AttributeValueList` harus berisi dua elemen skalar dari tipe yang sama - String, Angka, atau Biner.

Contoh

Ekspresi berikut bernilai true jika harga produk adalah antara 100 dan 200.

```
...
  "Price": {
    ComparisonOperator: "BETWEEN",
    AttributeValueList: [ {"N":"100"}, {"N":"200"} ]
  }
...
```

Operator perbandingan dengan n nilai atribut

- `IN` - benar jika nilai sama dengan salah satu nilai dalam daftar yang dienumerasi. Hanya nilai skalar yang didukung dalam daftar, bukan himpunan. Atribut target harus dari jenis yang sama dan nilai yang tepat agar cocok.

`AttributeValueList` dapat berisi satu atau beberapa elemen tipe String, Angka, atau Biner (bukan suatu set). Atribut ini dibandingkan dengan atribut jenis non-set suatu item. Jika setiap elemen dari set input ada dalam atribut item, ekspresi bernilai true.

`AttributeValueList` dapat berisi satu atau beberapa nilai dari tipe String, Angka, atau Biner (bukan suatu set). Atribut target perbandingan harus dari jenis yang sama dan nilai yang tepat agar cocok. Suatu String tidak pernah cocok dengan set String.

Gunakan operator ini untuk menentukan apakah nilai yang disediakan berada dalam daftar yang dienumerasi. Anda dapat menentukan sejumlah nilai skalar di `AttributeValueList`, tetapi jumlah tersebut semua harus dari tipe data yang sama.

Contoh

Ekspresi berikut bernilai true jika nilai untuk `Id` adalah 201, 203, atau 205.

```
...
  "Id": {
    ComparisonOperator: "IN",
```

```
    AttributeValueList: [ {"N":"201"}, {"N":"203"}, {"N":"205"} ]
  }
  ...
```

Menggunakan beberapa kondisi

DynamoDB memungkinkan Anda menggabungkan beberapa syarat untuk membentuk ekspresi kompleks. Anda melakukan ini dengan menyediakan setidaknya dua ekspresi, dengan [ConditionalOperator \(warisan\)](#) opsional.

Secara default, ketika Anda menentukan lebih dari satu syarat, semua syarat harus bernilai true agar seluruh ekspresi bernilai true. Dengan kata lain, operasi AND sedang berlangsung secara implisit.

Contoh

Ekspresi berikut mengevaluasi benar jika suatu produk adalah buku yang memiliki setidaknya 600 halaman. Kedua syarat harus bernilai true, karena keduanya secara menjalankan AND secara implisit dan bersama-sama.

```
...
  "ProductCategory": {
    ComparisonOperator: "EQ",
    AttributeValueList: [ {"S":"Book"} ]
  },
  "PageCount": {
    ComparisonOperator: "GE",
    AttributeValueList: [ {"N":"600"} ]
  }
  ...
```

Anda dapat menggunakan [ConditionalOperator \(warisan\)](#) untuk memperjelas bahwa operasi AND akan berlangsung. Contoh berikut berperilaku dengan cara yang sama seperti sebelumnya.

```
...
  "ConditionalOperator" : "AND",
  "ProductCategory": {
    "ComparisonOperator": "EQ",
    "AttributeValueList": [ {"N":"Book"} ]
  },
  "PageCount": {
    "ComparisonOperator": "GE",
```



```

    "AttributeValueList": [ {"N":600} ]
  }
  ...

```

Anda juga dapat mengatur `ConditionalOperator` menjadi `OR`, yang berarti bahwa Paling tidak salah satu syarat harus bernilai `true`.

Contoh

Ekspresi berikut bernilai `true` jika produk adalah sepeda gunung, jika produk tersebut adalah nama merek tertentu, atau jika harganya lebih besar dari 100.

```

...
ConditionalOperator : "OR",
"BicycleType": {
  "ComparisonOperator": "EQ",
  "AttributeValueList": [ {"S":"Mountain" } ]
},
"Brand": {
  "ComparisonOperator": "EQ",
  "AttributeValueList": [ {"S":"Brand-Company A" } ]
},
"Price": {
  "ComparisonOperator": "GT",
  "AttributeValueList": [ {"N":"100"} ]
}
...

```

Note

Dalam ekspresi yang kompleks, syarat diproses secara berurutan, mulai syarat pertama hingga terakhir.

Anda tidak dapat menggunakan kedua `AND` dan `OR` dalam satu ekspresi.

Operator bersyarat lainnya

Dalam rilis DynamoDB sebelumnya, parameter `Expected` berperilaku berbeda untuk menulis bersyarat. Setiap item dalam peta `Expected` mewakili nama atribut bagi DynamoDB untuk memeriksa, beserta hal berikut:

- `Value` – nilai untuk membandingkan terhadap atribut.
- `Exists` – menentukan apakah nilai ada sebelum mencoba operasi.

Pilihan `Value` dan `Exists` tetap didukung dalam DynamoDB; namun, keduanya hanya membiarkan Anda menguji syarat kesetaraan, atau apakah atribut ada. Kami menyarankan agar Anda menggunakan `ComparisonOperator` dan `AttributeValueList` sebagai gantinya, karena pilihan ini memungkinkan Anda membangun berbagai syarat yang lebih luas.

Example

`DeleteItem` dapat memeriksa untuk melihat apakah buku tidak lagi dalam publikasi, dan hanya menghapusnya jika syarat ini benar. Berikut adalah contoh AWS CLI menggunakan kondisi legasi:

```
aws dynamodb delete-item \  
  --table-name ProductCatalog \  
  --key '{  
    "Id": {"N":"600"}}  
' \  
  --expected '{  
    "InPublication": {  
      "Exists": true,  
      "Value": {"BOOL":false}  
    }  
'
```

Contoh berikut melakukan hal yang sama, tetapi tidak menggunakan kondisi legasi:

```
aws dynamodb delete-item \  
  --table-name ProductCatalog \  
  --key '{  
    "Id": {"N":"600"}}  
' \  
  --expected '{  
    "InPublication": {  
      "ComparisonOperator": "EQ",  
      "AttributeValueList": [ {"BOOL":false} ]  
    }  
'
```

Example

PutItem operasi dapat melindungi dari menimpa item yang ada dengan atribut kunci utama yang sama. Berikut adalah contoh menggunakan kondisi legasi:

```
aws dynamodb put-item \  
  --table-name ProductCatalog \  
  --item '{  
    "Id": {"N":"500"},  
    "Title": {"S":"Book 500 Title"}  
  }' \  
  --expected '{  
    "Id": { "Exists": false }  
  }'
```

Contoh berikut melakukan hal yang sama, tetapi tidak menggunakan kondisi legasi:

```
aws dynamodb put-item \  
  --table-name ProductCatalog \  
  --item '{  
    "Id": {"N":"500"},  
    "Title": {"S":"Book 500 Title"}  
  }' \  
  --expected '{  
    "Id": { "ComparisonOperator": "NULL" }  
  }'
```

Note

Untuk kondisi di peta Expected, jangan gunakan pilihan legasi Value dan Exists bersama dengan ComparisonOperator dan AttributeValueList. Jika melakukan hal ini, penulisan bersyarat Anda akan gagal.

Versi API tingkat rendah sebelumnya (2011-12-05)

Bagian ini mendokumentasikan operasi yang tersedia dalam versi API tingkat rendah DynamoDB sebelumnya (2011-12-05). Versi API tingkat rendah ini dipertahankan untuk kompatibilitas mundur dengan aplikasi yang ada.

Aplikasi baru harus menggunakan versi API terkini (2012-08-10). Untuk informasi selengkapnya, lihat [Referensi API tingkat rendah](#).

Note

Sebaiknya Anda memigrasikan aplikasi Anda ke API versi terbaru (2012-08-10), karena fitur DynamoDB baru tidak akan di-backport ke versi API sebelumnya.

Topik

- [BatchGetItem](#)
- [BatchWriteItem](#)
- [CreateTable](#)
- [DeleteItem](#)
- [DeleteTable](#)
- [DescribeTables](#)
- [GetItem](#)
- [ListTables](#)
- [PutItem](#)
- [Kueri](#)
- [Pemindaian](#)
- [UpdateItem](#)
- [UpdateTable](#)

BatchGetItem

Important

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

Operasi `BatchGetItem` mengembalikan atribut untuk beberapa item dari sejumlah tabel menggunakan kode utamanya. Jumlah maksimum item yang dapat diambil untuk suatu operasi adalah 100. Selain itu, jumlah item yang diambil dibatasi pada ukuran 1 MB. Jika batas ukuran respon terlampaui atau hasil parsial dikembalikan karena throughput yang disediakan tabel terlampaui, atau karena kegagalan pemrosesan internal, DynamoDB mengembalikan nilai `UnprocessedKeys` agar Anda dapat mencoba lagi operasi dimulai dengan item berikutnya untuk didapatkan. DynamoDB secara otomatis menyesuaikan jumlah item yang dikembalikan per halaman untuk memberlakukan batas ini. Misalnya, meskipun Anda meminta untuk mengambil 100 item, tetapi setiap item berukuran 50 KB, sistem akan mengembalikan 20 item dan nilai `UnprocessedKeys` yang sesuai agar Anda dapat memperoleh halaman hasil berikutnya. Jika diinginkan, aplikasi Anda dapat mencakup logikanya sendiri untuk menyusun halaman hasil menjadi satu set.

Jika tidak ada item yang dapat diproses karena throughput yang disediakan tidak memadai pada masing-masing tabel yang terlibat dalam permintaan, DynamoDB akan mengembalikan `ProvisionedThroughputExceededException` kesalahan.

Note

Secara default, `BatchGetItem` melakukan pembacaan yang akhirnya konsisten di setiap tabel dalam permintaan. Anda dapat mengatur parameter `ConsistentRead` menjadi `true`, sesuai setiap tabelnya, jika Anda ingin pembacaan yang konsisten sebagai gantinya. `BatchGetItem` mengambil item secara paralel untuk meminimalkan latensi respons. Saat merancang aplikasi Anda, ingatlah bahwa DynamoDB tidak menjamin cara atribut diperintahkan dalam respons yang dikembalikan. Sertakan nilai kunci primer dalam `AttributesToGet` untuk item dalam permintaan Anda guna membantu menguraikan respons berdasarkan item.

Jika item yang diminta tidak ada, tidak ada yang dikembalikan dalam respons untuk item tersebut. Permintaan untuk item yang tidak ada menggunakan unit kapasitas baca minimum sesuai dengan jenis pembacaan. Untuk informasi selengkapnya, lihat [Ukuran dan format item DynamoDB](#).

Permintaan

Sintaksis

```
// This header is abbreviated. For a sample of a complete header, see API tingkat rendah DynamoDB.
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"]
  }
}
```

Nama	Deskripsi	Wajib
RequestItems	Sebuah kontainer nama tabel dan item yang sesuai untuk mendapatkan kunci utama. Saat meminta item, setiap nama tabel dapat diaktifkan hanya sekali per operasi. Jenis: String Default: Tidak ada	Ya

Nama	Deskripsi	Wajib
Table	<p>Nama tabel yang berisi item untuk didapatkan. Entri hanya string yang menentukan tabel yang sudah ada tanpa label.</p> <p>Jenis: String</p> <p>Default: Tidak ada</p>	Ya
Table:Keys	<p>Nilai kunci utama yang menentukan item dalam tabel tertentu. Untuk informasi lebih lanjut tentang kunci utama, lihat Kunci primer.</p> <p>Jenis: Kunci</p>	Ya
Table:AttributesToGet	<p>Susunan nama Atribut dalam tabel yang ditentukan. Jika nama atribut tidak ditentukan maka semua atribut akan dikembalikan. Jika beberapa atribut tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil.</p> <p>Jenis: Susunan</p>	Tidak
Table:ConsistentRead	<p>Jika diatur menjadi true, pembacaan yang konsisten akan diterbitkan, jika tidak, pada akhirnya yang konsisten yang digunakan.</p> <p>Jenis: Boolean</p>	Tidak

Respons

Sintaksis

```

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"]}
  }

```


}

Nama	Deskripsi
Responses	<p>Nama tabel dan masing-masing atribut item dari tabel.</p> <p>Jenis: Map</p>
Table	<p>Nama tabel yang berisi item. Entri hanya berupa string yang menentukan tabel tanpa label.</p> <p>Jenis: String</p>
Items	<p>Kontainer untuk nama dan nilai atribut yang memenuhi parameter operasi.</p> <p>Jenis: Peta nama atribut serta nilai serta jenis data.</p>
ConsumedCapacityUnits	<p>Jumlah unit kapasitas baca yang dikonsumsi, untuk setiap tabel. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang Anda sediakan. Permintaan untuk item yang tidak ada menggunakan unit kapasitas baca minimum, tergantung pada jenis bacaan. Untuk informasi selengkapnya, lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB.</p> <p>Jenis: Jumlah</p>
UnprocessedKeys	<p>Berisi susunan tabel dan masing-masing kuncinya yang tidak diproses dengan respons terkini, kemungkinan karena mencapai batas ukuran respons. Nilai UnprocessedKeys berada dalam formulir yang sama dengan parameter RequestItems (sehingga</p>

Nama	Deskripsi
	<p>nilai dapat diberikan langsung ke operasi <code>BatchGetItem</code> berikutnya). Untuk informasi selengkapnya, lihat parameter <code>RequestItems</code> di atas.</p> <p>Jenis: Susunan</p>
<p><code>UnprocessedKeys : Table: Keys</code></p>	<p>Nilai atribut kunci utama yang menentukan item dan atribut yang terkait dengan item. Untuk informasi lebih lanjut tentang kunci utama, lihat Kunci primer.</p> <p>Jenis: Susunan pasangan nama-nilai atribut.</p>
<p><code>UnprocessedKeys : Table: AttributesToGet</code></p>	<p>Nama atribut dalam tabel yang ditentukan. Jika nama atribut tidak ditentukan maka semua atribut akan dikembalikan. Jika beberapa atribut tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil.</p> <p>Jenis: Array nama atribut.</p>
<p><code>UnprocessedKeys : Table: ConsistentRead</code></p>	<p>Jika diatur menjadi <code>true</code>, maka pembacaan konsisten digunakan untuk tabel tertentu, atau pembacaan yang akhirnya konsisten akan digunakan.</p> <p>Jenis: Boolean.</p>

Kesalahan Khusus

Kesalahan	Deskripsi
<p><code>ProvisionedThroughputExceededException</code></p>	<p>Throughput yang disediakan maksimum milik Anda telah terlampaui.</p>

Contoh

Contoh berikut menunjukkan permintaan HTTP POST dan respons menggunakan BatchGetItem operasi. Untuk contoh menggunakan SDK AWS, lihat [Bekerja dengan item dan atribut](#).

Permintaan sampel

Contoh berikut meminta atribut dari dua tabel yang berbeda.

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"]}
  }
}
```

Contoh respons

Contoh berikut ini adalah responsnya.

```
HTTP/1.1 200 OK
x-amzn-RequestId: GTPQVRM4VJS792J1UFJTKUBVV4KQNS05AEMVJF66Q9ASUAAJG
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

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB Dynamo](#).

Deskripsi

Operasi ini memungkinkan Anda untuk menempatkan atau menghapus beberapa item di beberapa tabel dalam satu panggilan.

Untuk mengunggah satu item, Anda dapat menggunakan `PutItem`, dan untuk menghapus satu item, Anda dapat menggunakan `DeleteItem`. Namun, jika Anda ingin mengunggah atau menghapus sejumlah besar data, seperti mengunggah sejumlah besar data dari Amazon EMR (Amazon EMR) atau memigrasi data dari database lain ke DynamoDB, `BatchWriteItem` menawarkan alternatif yang efisien.

Jika Anda menggunakan bahasa seperti Java, Anda dapat menggunakan thread untuk mengunggah item secara paralel. Hal ini menambah kompleksitas dalam aplikasi Anda untuk menangani thread. Bahasa lain tidak mendukung threading. Misalnya, jika Anda menggunakan PHP, Anda harus mengunggah atau menghapus item satu per satu. Dalam kedua situasi, `BatchWriteItem` menyediakan alternatif di mana operasi menempatkan dan menghapus yang ditentukan diproses

secara paralel, memberikan Anda kekuatan pendekatan thread pool tanpa harus memasukkan kompleksitas dalam aplikasi Anda.

Perhatikan bahwa setiap operasi menempatkan dan menghapus individual yang ditentukan dalam operasi `BatchWriteItem` berbiaya sama dalam hal unit kapasitas yang digunakan. Namun, karena `BatchWriteItem` melakukan operasi yang ditentukan secara paralel, Anda mendapatkan latensi yang lebih rendah. Operasi hapus pada item yang tidak ada mengonsumsi 1 unit kapasitas tulis. Untuk informasi lebih lanjut tentang unit kapasitas yang digunakan, lihat [Bekerja dengan tabel dan data di DynamoDB](#).

Saat menggunakan `BatchWriteItem`, perhatikan batasan berikut:

- Operasi maksimum dalam satu permintaan – Anda dapat menentukan hingga total 25 operasi menempatkan atau menghapus; namun, total permintaan ukuran tidak dapat melampaui 1 MB (payload HTTP).
- Anda dapat menggunakan operasi `BatchWriteItem` hanya untuk menempatkan dan menghapus item. Anda tidak dapat menggunakannya untuk memperbarui item yang ada.
- Bukan operasi atom – Operasi individu yang ditentukan dalam `BatchWriteItem` adalah atomik; namun `BatchWriteItem` secara keseluruhan adalah operasi "upaya terbaik" dan bukan operasi atomik. Artinya, dalam permintaan `BatchWriteItem`, beberapa operasi mungkin berhasil dan yang lainnya mungkin gagal. Operasi yang gagal dikembalikan ke bidang `UnprocessedItems` dalam respons. Sejumlah kegagalan ini mungkin karena Anda melampaui throughput yang disediakan dan dikonfigurasi untuk tabel atau kegagalan sementara seperti kesalahan jaringan. Anda dapat menyelidiki dan secara opsional mengirim ulang permintaan. Biasanya, Anda memanggil `BatchWriteItem` dalam satu putaran dan di setiap pemeriksaan pengulangan untuk item yang belum diproses, dan mengirimkan permintaan `BatchWriteItem` baru dengan item yang belum diproses tersebut.
- Tidak mengembalikan item apa pun – `BatchWriteItem` dirancang untuk mengunggah data dalam jumlah besar secara efisien. Hal itu tidak menyediakan sejumlah kecanggihan yang ditawarkan oleh `PutItem` dan `DeleteItem`. Sebagai contoh, `DeleteItem` mendukung bidang `ReturnValues` di isi permintaan Anda untuk meminta item yang dihapus dalam respons. Operasi `BatchWriteItem` tidak mengembalikan item apa pun dalam respons.
- Tidak seperti `PutItem` dan `DeleteItem`, `BatchWriteItem` tidak memungkinkan Anda untuk menentukan syarat pada permintaan tulis individu dalam operasi.
- Nilai atribut tidak boleh null; atribut jenis string dan biner harus memiliki panjang lebih dari nol; dan atribut jenis set tidak boleh kosong. Permintaan yang memiliki nilai kosong akan ditolak dengan `ValidationException`.

DynamoDB menolak keseluruhan batch operasi penulisan jika salah satu dari yang berikut ini benar:

- Jika satu atau beberapa tabel yang ditentukan dalam permintaan `BatchWriteItem` tidak ada.
- Jika atribut kunci primer yang ditentukan pada item dalam permintaan tidak cocok dengan skema kunci primer dalam tabel yang sesuai.
- Jika Anda mencoba untuk melakukan beberapa operasi pada item yang sama dalam permintaan `BatchWriteItem` yang sama. Sebagai contoh, Anda tidak dapat menempatkan dan menghapus item yang sama dalam permintaan `BatchWriteItem` yang sama.
- Jika ukuran permintaan total melampaui batas ukuran permintaan 1 MB (payload HTTP).
- Jika setiap item individu dalam suatu batch melampaui batas ukuran item 64 KB.

Permintaan

Sintaksis

```
// This header is abbreviated. For a sample of a complete header, see API tingkat rendah DynamoDB.
```

```
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", ... ]
}
```

Dalam isi permintaan, objek JSON `RequestItems` menggambarkan operasi yang ingin Anda lakukan. Operasi dikelompokkan berdasarkan tabel. Anda dapat menggunakan `BatchWriteItem` untuk memperbarui atau menghapus beberapa item di beberapa tabel. Untuk setiap permintaan penulisan khusus, Anda harus mengidentifikasi jenis permintaan (`PutItem`, `DeleteItem`) diikuti dengan informasi terperinci tentang operasi.

- Untuk `PutRequest`, Anda memberikan item, yaitu daftar atribut dan nilai-nilainya.
- Untuk `DeleteRequest`, Anda memberikan nama dan nilai kunci utama.

Respons

Sintaksis

Berikut ini adalah sintaks dari isi JSON yang dikembalikan dalam respons.

```
{
  "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.

Kesalahan Khusus

Tidak ada kesalahan khusus untuk operasi ini.

Contoh

Contoh berikut menunjukkan permintaan HTTP POST dan respons menggunakan dari operasi `BatchWriteItem`. Permintaan menentukan operasi berikut pada tabel Balas dan Thread:

- Menempatkan item dan menghapus item dari tabel Balas
- Menempatkan item ke dalam tabel Thread

Untuk contoh menggunakan SDK AWS, lihat [Bekerja dengan item dan atribut](#).

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat rendah DynamoDB.
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"
            }
          }
        }
      }
    ]
  }
}

```

Contoh respons

Respons contoh berikut menunjukkan operasi put pada tabel Thread dan Balas yang berhasil dan operasi hapus pada tabel Balas yang gagal (karena alasan seperti throttling yang disebabkan saat Anda melampaui throughput yang disediakan di tabel). Perhatikan hal berikut dalam respons JSON:

- Objek Responses menunjukkan satu unit kapasitas yang digunakan pada tabel Thread dan Reply sebagai hasil dari kesuksesan operasi put pada setiap tabel ini.
- Objek UnprocessedItems menunjukkan kegagalan operasi hapus pada tabel Reply. Anda kemudian dapat mengeluarkan panggilan BatchWriteItem baru untuk mengatasi permintaan yang belum diproses ini.

```

HTTP/1.1 200 OK
x-amzn-RequestId: G8M9ANL0E5QA26AEUHJKJE0ASBVV4KQNS05AEMVJF66Q9ASUAAJG
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

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

Operasi `CreateTable` menambahkan tabel baru ke akun Anda.

Nama tabel harus unik di antara yang terkait dengan Akun AWS yang mengeluarkan permintaan, dan wilayah AWS yang menerima permintaan (seperti `dynamodb.us-west-2.amazonaws.com`). Setiap titik akhir DynamoDB bersifat independen sepenuhnya. Misalnya, jika Anda memiliki dua tabel yang disebut "MyTable," satu di `dynamodb.us-west-2.amazonaws.com` dan satu di `dynamodb.us-west-1.amazonaws.com`, keduanya independen sepenuhnya dan tidak berbagi data apapun.

Operasi `CreateTable` memicu alur kerja asinkron untuk mulai membuat tabel. DynamoDB segera mengembalikan status tabel (CREATING) hingga tabel berada dalam status ACTIVE. Setelah tabel berada dalam status ACTIVE, Anda dapat melakukan operasi bidang data.

Gunakan operasi [DescribeTables](#) untuk memeriksa status tabel.

Permintaan


Sintaksis

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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}
}
```

Nama	Deskripsi	Wajib
TableName	Nama tabel untuk dibuat. Karakter yang diperbolehkan adalah a-z, A-Z, 0-9, '_' (garis bawah), '-' (dash), dan '.' (titik).	Ya

Nama	Deskripsi	Wajib
	<p>Panjang nama dapat antara 3 dan 255 karakter.</p> <p>Jenis: Tali</p>	
KeySchema	<p>Struktur kunci utama (sederhana atau komposit) untuk tabel. Pasangan nama-nilai untuk HashKeyElement diperlukan, dan pasangan nama-nilai untuk RangeKeyElement adalah opsional (hanya diperlukan untuk kunci utama komposit). Untuk informasi lebih lanjut tentang kunci utama, lihat Kunci primer.</p> <p>Nama elemen kunci utama dapat antara 1 dan 255 karakter tanpa batasan karakter.</p> <p>Kemungkinan nilai untuk AttributeType adalah "S" (string), "N" (numerik), atau "B" (biner).</p> <p>Jenis: Peta dari HashKeyElement, atau HashKeyElement dan RangeKeyElement untuk kunci utama komposit.</p>	Ya

Nama	Deskripsi	Wajib
ProvisionedThroughput	<p>Throughput baru untuk tabel tertentu, yang terdiri dari nilai-nilai untuk <code>ReadCapacityUnits</code> dan <code>WriteCapacityUnits</code> . Untuk rincian selengkapnya, lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB.</p> <div data-bbox="591 684 1029 1050" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px;"><p> Note</p><p>Untuk nilai maksimum/minimum saat ini, lihat Layanan, akun, dan tabel kuota di Amazon DynamoDB.</p></div> <p>Jenis: Susunan</p>	Ya

Nama	Deskripsi	Wajib
ProvisionedThroughput : ReadCapacityUnits	<p>Menetapkan jumlah minimum ReadCapacityUnits yang konsisten digunakan per detik untuk tabel tertentu sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya.</p> <p>Operasi baca yang konsisten pada bagian akhir memerlukan lebih sedikit usaha daripada operasi baca yang konsisten, jadi pengaturan 50 ReadCapacityUnits yang konsisten per detik akan menghasilkan 100 ReadCapacityUnits yang konsisten pada bagian akhir per detik.</p> <p>Jenis: Jumlah</p>	Ya
ProvisionedThroughput : WriteCapacityUnits	<p>Menetapkan jumlah minimum WriteCapacityUnits yang digunakan per detik untuk tabel tertentu sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya.</p> <p>Jenis: Jumlah</p>	Ya

Respons

Sintaksis

```
HTTP/1.1 200 OK
```


```
x-amzn-RequestId: CS0C7TJPLR000KIRLG0HVAICUFVV4KQNS05AEMVJF66Q9ASUAAJG
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"
  }
}
```

Nama	Deskripsi
TableDescription	Kontainer untuk properti tabel.
CreationDateTime	Tanggal ketika tabel dibuat di jangka waktu UNIX . Jenis: Jumlah
KeySchema	Struktur kunci utama (sederhana atau komposit) untuk tabel. Pasangan nama-nilai untuk HashKeyElement diperlukan, dan pasangan nama-nilai untuk RangeKeyElement adalah opsional (hanya diperlukan untuk kunci utama komposit). Untuk informasi lebih lanjut tentang kunci utama, lihat Kunci primer . Jenis: Peta dari HashKeyElement, atau HashKeyElement dan RangeKeyElement untuk kunci utama komposit.
ProvisionedThroughput	Throughput untuk tabel tertentu, yang terdiri dari nilai untuk ReadCapacityUnits dan

Nama	Deskripsi
	<p>WriteCapacityUnits . Lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB.</p> <p>Jenis: Susunan</p>
ProvisionedThroughput :ReadCapacityUnits	<p>Jumlah minimum ReadCapacityUnits yang digunakan per detik sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya</p> <p>Jenis: Jumlah</p>
ProvisionedThroughput :WriteCapacityUnits	<p>Jumlah minimum ReadCapacityUnits yang digunakan per detik sebelum WriteCapacityUnits menyeimbangkan beban dengan operasi lainnya</p> <p>Jenis: Jumlah</p>
TableName	<p>Nama tabel yang dibuat.</p> <p>Jenis: String</p>
TableStatus	<p>Keadaan tabel saat ini (CREATING). Setelah tabel berada dalam status ACTIVE, Anda dapat menempatkan data di dalamnya.</p> <p>Gunakan API DescribeTables untuk memeriksa status tabel.</p> <p>Jenis: String</p>

Kesalahan Khusus

Kesalahan	Deskripsi
ResourceInUseException	Mencoba untuk membuat kembali tabel yang sudah ada.
LimitExceededException	Jumlah permintaan tabel simultan (jumlah kumulatif tabel dalam status CREATING, DELETING atau UPDATING) melampaui jumlah maksimum yang diizinkan. <div data-bbox="829 680 1507 947"><p> Note</p><p>Untuk nilai maksimum/minimum saat ini, lihat Layanan, akun, dan tabel kuota di Amazon DynamoDB.</p></div>

Contoh

Contoh berikut membuat tabel dengan kunci utama komposit yang berisi string dan angka. Untuk contoh menggunakan SDK AWS, lihat [Bekerja dengan tabel dan data di DynamoDB](#).

Permintaan sampel

```
// This header is abbreviated.  
// For a sample of a complete header, see API tingkat rendah DynamoDB.  
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}
}
```

Contoh respons

```
HTTP/1.1 200 OK
x-amzn-RequestId: CS0C7TJPLR000KIRLG0HVAICUFVV4KQNS05AEMVJF66Q9ASUAAJG
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"
  }
}
```

Tindakan terkait

- [DescribeTables](#)
- [DeleteTable](#)

Deleteltem

Important

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

Menghapus satu item dalam tabel dengan kunci utama. Anda dapat melakukan operasi hapus bersyarat yang menghapus item jika ada, atau jika memiliki nilai atribut yang diharapkan.

Note

Jika Anda menentukan `DeleteItem` tanpa atribut atau nilai, semua atribut untuk item akan dihapus.

Kecuali jika Anda menentukan syarat, `DeleteItem` adalah operasi idempoten; menjalankannya beberapa kali pada item atau atribut yang sama tidak menghasilkan respons kesalahan.

Penghapusan bersyarat berguna hanya untuk menghapus item dan atribut jika kondisi tertentu terpenuhi. Jika syarat terpenuhi, DynamoDB melakukan penghapusan. Jika tidak, item tersebut tidak akan dihapus.

Anda dapat melakukan pemeriksaan bersyarat yang diharapkan pada satu atribut per operasi.

Permintaan


Sintaksis

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"}
}
```

Nama	Deskripsi	Wajib
TableName	<p>Nama tabel yang berisi item yang akan dihapus.</p> <p>Jenis: Tali</p>	Ya
Key	<p>Kunci utama yang menentukan item. Untuk informasi lebih lanjut tentang kunci utama, lihat Kunci primer.</p> <p>Jenis: Peta dari HashKeyElement untuk nilainya dan RangeKeyElement untuk nilainya.</p>	Ya
Expected	<p>Menunjuk atribut untuk penghapusan bersyarat. Parameter Expected memungkinkan Anda untuk memberikan nama atribut, dan apakah DynamoDB harus memeriksa bahwa atribut memiliki nilai tertentu sebelum mengubahnya.</p> <p>Jenis: Peta nama atribut.</p>	Tidak
Expected:Attribute Name	<p>Nama atribut untuk put bersyarat.</p> <p>Jenis: String</p>	Tidak
Expected:Attribute Name: ExpectedAttributeValue	<p>Gunakan parameter ini untuk menentukan apakah nilai sudah ada atau belum untuk pasangan nama-nilai atribut.</p>	Tidak

Nama	Deskripsi	Wajib
	<p>Notasi JSON berikut menghapus item jika atribut "Warna" tidak ada untuk item tersebut:</p> <pre data-bbox="594 426 1027 583">"Expected" : {"Color":{"Exists":false}}</pre> <p>Notasi JSON berikut memeriksa untuk melihat apakah atribut dengan nama "Warna" memiliki nilai "Kuning" yang ada sebelum menghapus item:</p> <pre data-bbox="594 932 1027 1131">"Expected" : {"Color":{"Exists":true}, {"Value": {"S":"Yellow"}}}</pre> <p>Secara default, jika Anda menggunakan parameter Expected dan menyediakan Value, DynamoDB mengasumsikan bahwa atribut ada dan memiliki nilai terkini untuk diganti. Jadi Anda tidak perlu menentukan {"Exists":true} , karena tersirat. Anda dapat mempersingkat permintaan untuk:</p> <pre data-bbox="594 1719 1027 1877">"Expected" : {"Color":{"Value": {"S":"Yellow"}}}</pre>	

Nama	Deskripsi	Wajib
	<p> Note</p> <p>Jika Anda menentukan {"Exists":true} tanpa nilai atribut untuk memeriksa , DynamoDB mengembalikan kesalahan.</p>	
ReturnValues	<p>Gunakan parameter ini jika Anda ingin mendapatkan pasangan nama-nilai atribut sebelum pasangan tersebut dihapus. Nilai parameter yang memungkinkan adalah NONE (default) atau ALL_OLD. Jika ALL_OLD ditentukan, isi dari item lama akan dikembalikan. Jika parameter ini tidak tersedia atau NONE, tidak ada yang dikembalikan.</p> <p>Jenis: String</p>	Tidak

Respons

Sintaksis

```
HTTP/1.1 200 OK
x-amzn-RequestId: CS0C7TJPLR000KIRLG0HVAICUFVV4KQNS05AEMVJF66Q9ASUAAJG
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
}

```

Nama	Deskripsi
Attributes	<p>Jika parameter ReturnValues disediakan sebagai ALL_OLD dalam permintaan, DynamoDB mengembalikan susunan pasangan nama-nilai atribut (pada dasarnya, item dihapus). Jika tidak, respons berisi satu set kosong.</p> <p>Jenis: Susunan pasangan nama-nilai atribut.</p>
ConsumedCapacityUnits	<p>Jumlah unit kapasitas tulis yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang Anda sediakan. Permintaan hapus pada item yang tidak ada menggunakan 1 unit kapasitas tulis. Untuk informasi selengkapnya, lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB.</p> <p>Jenis: Jumlah</p>

Kesalahan Khusus

Kesalahan	Deskripsi
ConditionalCheckFailedException	Pemeriksaan bersyarat gagal. Nilai atribut yang diharapkan tidak ditemukan.

Contoh

Permintaan sampel

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"
}
```

Contoh respons

```
HTTP/1.1 200 OK
x-amzn-RequestId: U9809LI6BBFJA5N2R0TB0P017JVV4KQNS05AEMVJF66Q9ASUAAJG
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
}
```

Tindakan terkait

- [PutItem](#)

DeleteTable

Important

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

Operasi DeleteTable menghapus tabel dan semua itemnya. Setelah permintaan DeleteTable, tabel yang ditentukan berada dalam status DELETING hingga DynamoDB menyelesaikan penghapusan. Jika tabel berada dalam status ACTIVE, Anda dapat menghapusnya. Jika tabel berada dalam status CREATING atau UPDATING, maka DynamoDB mengembalikan kesalahan ResourceInUseException. Jika tabel yang ditentukan tidak ada, DynamoDB mengembalikan ResourceNotFoundException. Jika tabel sudah berada dalam status DELETING, tidak ada kesalahan yang dikembalikan.

Note

DynamoDB dapat terus menerima permintaan operasi bidang data, seperti GetItem dan PutItem, pada tabel dalam status DELETING hingga penghapusan tabel selesai.

Tabel harus bersifat unik di antara yang terkait dengan Akun AWS yang mengeluarkan permintaan, dan wilayah AWS yang menerima permintaan (seperti dynamodb.us-west-1.amazonaws.com). Setiap titik akhir DynamoDB bersifat independen sepenuhnya. Misalnya, jika Anda memiliki dua tabel yang disebut "MyTable," satu di dynamodb.us-west-2.amazonaws.com dan satu di dynamodb.us-west-1.amazonaws.com, keduanya independen sepenuhnya dan tidak berbagi data apapun; menghapus satu data tidak menghapus yang lain.

Gunakan operasi [DescribeTables](#) untuk memeriksa status tabel.

Permintaan

Sintaksis

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Nama	Deskripsi	Wajib
TableName	Nama tabel untuk dihapus. Jenis: Tali	Ya

Respons

Sintaksis

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4H0NCKIVH1BFUDQ1U68CTG3N27VV4KQNS05AEMVJF66Q9ASUAAJG
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"
  }
}
```

Nama	Deskripsi
TableDescription	Kontainer untuk properti tabel.
CreationDateTime	<p>Tanggal ketika tabel dibuat.</p> <p>Jenis: Jumlah</p>
KeySchema	<p>Struktur kunci utama (sederhana atau komposit) untuk tabel. Pasangan nama-nilai untuk HashKeyElement diperlukan, dan pasangan nama-nilai untuk RangeKeyElement adalah opsional (hanya diperlukan untuk kunci utama komposit). Untuk informasi lebih lanjut tentang kunci utama, lihat Kunci primer.</p> <p>Jenis: Memetakan berbagai halHashKeyElement, atauHashKeyElement danRangeKeyElement untuk kunci primer komposit.</p>
ProvisionedThroughput	Throughput untuk tabel tertentu, yang terdiri dari nilai untuk ReadCapacityUnits dan WriteCapacityUnits. Lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB .
ProvisionedThroughput : ReadCapacityUnits	<p>Jumlah minimum ReadCapacityUnits yang digunakan per detik untuk tabel tertentu sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya.</p> <p>Jenis: Jumlah</p>
ProvisionedThroughput : WriteCapacityUnits	Jumlah minimum WriteCapacityUnits yang digunakan per detik untuk tabel tertentu

Nama	Deskripsi
	<p>sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya.</p> <p>Jenis: Jumlah</p>
TableName	<p>Nama tabel yang dihapus.</p> <p>Jenis: String</p>
TableStatus	<p>Keadaan tabel saat ini (DELETING). Setelah tabel dihapus, permintaan berikutnya untuk tabel mengembalikan <code>resource not found</code>.</p> <p>Gunakan operasi DescribeTables untuk memeriksa status tabel.</p> <p>Jenis: String</p>

Kesalahan Khusus

Kesalahan	Deskripsi
ResourceInUseException	Tabel dalam status CREATING atau UPDATING dan tidak dapat dihapus.

Contoh

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat rendah DynamoDB.
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"}
```

Contoh respons

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4H0NCKIVH1BFUDQ1U68CTG3N27VV4KQNS05AEMVJF66Q9ASUAAJG
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"
}
```

Tindakan terkait

- [CreateTable](#)
- [DescribeTables](#)

DescribeTables

Important

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

Mengembalikan informasi tentang tabel, termasuk status terkini tabel, skema kunci utama dan waktu pembuatan tabel. DescribeTable Hasil pada akhirnya konsisten. Jika Anda menggunakan DescribeTable terlalu dini dalam proses membuat tabel, DynamoDB

mengembalikan `ResourceNotFoundException`. Jika Anda menggunakan `DescribeTable` terlalu dini dalam proses memperbarui tabel, nilai-nilai baru mungkin tidak langsung tersedia.

Permintaan

Sintaksis

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Nama	Deskripsi	Wajib
TableName	Nama tabel untuk menggambar. Jenis: Tali	Ya

Respons

Sintaksis

```
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"
  }
}

```

Nama	Deskripsi
Table	<p>Kontainer untuk tabel yang sedang dijelaskan.</p> <p>Jenis: String</p>
CreationDateTime	<p>Tanggal ketika tabel dibuat di jangka waktu UNIX.</p>
ItemCount	<p>Jumlah item dalam tabel yang ditentukan. DynamoDB memperbarui nilai ini kira-kira setiap enam jam. Perubahan terbaru mungkin tidak tercermin dalam nilai ini.</p> <p>Jenis: Jumlah</p>
KeySchema	<p>Struktur kunci utama (sederhana atau komposit) untuk tabel. Pasangan nama-nilai untuk HashKeyElement diperlukan, dan pasangan nama-nilai untuk RangeKeyElement adalah opsional (hanya diperlukan untuk kunci utama komposit). Ukuran kunci hash maksimum adalah 2048 byte. Ukuran kunci rentang maksimum adalah 1024 byte. Kedua batas tersebut diberlakukan secara terpisah (yaitu Anda dapat memiliki kombinasi hash + rentang 2048 + 1024 kunci). Untuk informasi lebih lanjut tentang kunci utama, lihat Kunci primer.</p>
ProvisionedThroughput	<p>Throughput untuk tabel tertentu, yang terdiri dari nilai-nilai untuk LastIncreaseDateTime (jika berlaku), LastDecreaseDateTime (jika berlaku), ReadCapacityUnits</p>

Nama	Deskripsi
	<p>dan <code>WriteCapacityUnits</code> . Jika throughput untuk tabel tidak pernah meningkat atau menurun, DynamoDB tidak mengembalikan nilai untuk elemen-elemen tersebut. Lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB.</p> <p>Jenis: Susunan</p>
TableName	<p>Nama tabel yang diminta.</p> <p>Jenis: String</p>
TableSizeBytes	<p>Ukuran total tabel yang ditentukan, dalam byte. DynamoDB memperbarui nilai ini kira-kira setiap enam jam. Perubahan terbaru mungkin tidak tercermin dalam nilai ini.</p> <p>Jenis: Jumlah</p>
TableStatus	<p>Status tabel saat ini (CREATING, ACTIVE, DELETING atau UPDATING). Setelah tabel berada dalam status ACTIVE, Anda dapat menambahkan data.</p>

Kesalahan Khusus

Tidak ada kesalahan khusus dalam operasi ini.

Contoh

Contoh berikut menunjukkan permintaan HTTP POST dan respons menggunakan `DescribeTable` operasi untuk tabel bernama “comp-table”. Tabel memiliki kunci utama komposit.

Permintaan Sampel

```
// This header is abbreviated.
```

```
// For a sample of a complete header, see API tingkat rendah DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"users"}
```

Contoh respons

```
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"
  }
}
```

Tindakan terkait

- [CreateTable](#)
- [DeleteTable](#)
- [ListTables](#)

GetItem

Important

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

Operasi `GetItem` mengembalikan sekumpulan `Attributes` untuk item yang cocok dengan kunci utama. Jika tidak ada item yang cocok, `GetItem` tidak mengembalikan data apa pun.

Operasi `GetItem` menyediakan pembacaan yang pada akhirnya konsisten secara default. Jika pembacaan yang pada akhirnya konsisten tidak dapat diterima untuk aplikasi Anda, gunakan `ConsistentRead`. Meskipun operasi ini mungkin memakan waktu lebih lama dari pembaca standar, operasi ini selalu mengembalikan nilai yang terakhir diperbarui. Untuk informasi selengkapnya, lihat [Konsistensi baca](#).

Permintaan

Sintaksis

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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
}
```

Nama	Deskripsi	Wajib
TableName	Nama tabel yang berisi item yang diminta. Jenis: Tali	Ya

Nama	Deskripsi	Wajib
Key	<p>Nilai kunci utama yang menentukan item. Untuk informasi lebih lanjut tentang kunci utama, lihat Kunci primer.</p> <p>Jenis: Peta dari <code>HashKeyElement</code> untuk nilainya dan <code>RangeKeyElement</code> untuk nilainya.</p>	Ya
AttributesToGet	<p>Susunan nama Atribut. Jika nama atribut tidak ditentukan maka semua atribut akan dikembalikan. Jika beberapa atribut tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil.</p> <p>Jenis: Susunan</p>	Tidak
ConsistentRead	<p>Jika diatur menjadi <code>true</code>, pembacaan yang konsisten akan diterbitkan, jika tidak, pada akhirnya yang konsisten yang digunakan.</p> <p>Jenis: Boolean</p>	Tidak

Respons

Sintaksis

```
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
}
```

Nama	Deskripsi
Item	<p>Berisi atribut yang diminta.</p> <p>Jenis: Peta pasangan nama-nilai atribut.</p>
ConsumedCapacityUnits	<p>Jumlah unit kapasitas baca yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang Anda sediakan. Permintaan untuk item yang tidak ada menggunakan unit kapasitas baca minimum, tergantung pada jenis bacaan. Untuk informasi selengkapnya, lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB.</p> <p>Jenis: Jumlah</p>

Kesalahan Khusus

Tidak ada kesalahan khusus untuk operasi ini.

Contoh

Untuk contoh menggunakan SDK AWS, lihat [Bekerja dengan item dan atribut](#).

Permintaan sampel

```
// This header is abbreviated.
```

```
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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
}
```

Contoh respons

Perhatikan ConsumedCapacityUnits nilai adalah 1, karena parameter optionalConsistentRead diatur menjadi true. Jika ConsistentRead diatur menjadi false (atau tidak ditentukan) untuk permintaan yang sama, respons pada akhirnya akan konsisten dan ConsumedCapacityUnits nilai akan menjadi 0,5.

```
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

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

Mengembalikan susunan semua tabel yang terkait dengan akun dan titik akhir saat ini.

Setiap titik akhir DynamoDB bersifat independen sepenuhnya. Misalnya, jika Anda memiliki dua tabel yang disebut "MyTable," satu di dynamodb.us-west-2.amazonaws.com dan satu di dynamodb.us-east-1.amazonaws.com, keduanya independen sepenuhnya dan tidak berbagi data apapun.

Parameter ListTables Operasi mengembalikan semua nama tabel yang terkait dengan akun pembuat permintaan, untuk titik akhir yang menerima permintaan.

Permintaan

Sintaksis

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"Table1","Limit":3}
```

Parameter ListTables Operasi, secara default, meminta semua nama tabel yang terkait dengan akun pembuat permintaan, untuk titik akhir yang menerima permintaan.

Nama	Deskripsi	Wajib
Limit	Jumlah maksimum nama tabel yang akan dikembalikan. Jenis: Bulat	Tidak
ExclusiveStartTableName	Nama tabel yang dimulai daftar. Jika Anda sudah menjalankan ListTables operasi dan menerima LastEvaluatedTableName nilai dalam respons, gunakan	Tidak

Nama	Deskripsi	Wajib
	<p>nilai tersebut di sini untuk melanjutkan daftar.</p> <p>Jenis: String</p>	

Respons

Sintaksis

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP80JNHVHL80U2M7KRVV4KQNS05AEMVJF66Q9ASUAAJG
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"}
```

Nama	Deskripsi
TableNames	<p>Nama-nama tabel yang terkait dengan akun saat ini pada titik akhir terkini.</p> <p>Jenis: Susunan</p>
LastEvaluatedTableName	<p>Nama tabel terakhir dalam daftar saat ini, hanya jika beberapa tabel untuk akun dan titik akhir belum dikembalikan. Nilai ini tidak ada dalam respons jika semua nama tabel sudah dikembalikan. Gunakan nilai ini sebagai <code>ExclusiveStartTableName</code> dalam permintaan baru untuk melanjutkan daftar hingga semua nama tabel dikembalikan.</p> <p>Jenis: String</p>

Kesalahan Khusus

Tidak ada kesalahan khusus dalam operasi ini.

Contoh

Contoh berikut menunjukkan permintaan HTTP POST dan respons menggunakan ListTablesOperasi.

Permintaan sampel

```
// This header is abbreviated.  
// For a sample of a complete header, see API tingkat rendah DynamoDB.  
POST / HTTP/1.1  
x-amz-target: DynamoDB_20111205.ListTables  
content-type: application/x-amz-json-1.0  
  
{"ExclusiveStartTableName":"comp2","Limit":3}
```

Contoh respons

```
HTTP/1.1 200 OK  
x-amzn-RequestId: S1LEK2DPQP80JNHVHL80U2M7KRVV4KQNS05AEMVJF66Q9ASUAAJG  
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"]}
```

Tindakan terkait

- [DescribeTables](#)
- [CreateTable](#)
- [DeleteTable](#)

PutItem

Important

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

Membuat item baru, atau menggantikan item lama dengan item baru (termasuk semua atribut). Jika item sudah ada dalam tabel tertentu dengan kunci utama yang sama, item baru benar-benar menggantikan item yang ada. Anda dapat melakukan put bersyarat (masukkan item baru jika salah satu dengan kunci utama yang ditentukan tidak ada), atau mengganti item yang ada jika memiliki nilai atribut tertentu.

Nilai atribut tidak boleh null; atribut jenis string dan biner harus memiliki panjang lebih dari nol; dan atribut jenis set tidak boleh kosong. Permintaan dengan nilai kosong akan ditolak dengan `ValidationException`.

Note

Untuk memastikan bahwa item baru tidak menggantikan item yang ada, gunakan operasi put bersyarat dengan `Exists` yang diatur menjadi `false` untuk atribut kunci utama, atau atribut-atribut.

Untuk informasi selengkapnya tentang menggunakan `PutItem`, lihat [Bekerja dengan item dan atribut](#).

Permintaan

Sintaksis

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"}

```

Nama	Deskripsi	Wajib
TableName	<p>Nama tabel untuk memuat item.</p> <p>Jenis: Tali</p>	Ya
Item	<p>Peta atribut untuk item, dan harus menyertakan nilai kunci utama yang menentukan item. Pasangan nama-nilai atribut lainnya dapat diberikan untuk item tersebut. Untuk informasi lebih lanjut tentang kunci utama, lihat Kunci primer.</p> <p>Jenis: Peta nama atribut untuk nilai atribut.</p>	Ya
Expected	<p>Menunjuk atribut untuk put bersyarat. Parameter Expected memungkinkan Anda untuk memberikan nama atribut, dan apakah DynamoDB harus memeriksa keberadaan nilai atribut; atau apakah nilai atribut ada dan memiliki nilai tertentu sebelum mengubahnya.</p> <p>Jenis: Peta nama atribut untuk nilai atribut, dan apakah nilai tersebut ada.</p>	Tidak

Nama	Deskripsi	Wajib
Expected:Attribute Name	Nama atribut untuk put bersyarat. Jenis: String	Tidak

Nama	Deskripsi	Wajib
<code>Expected:Attribute Name: ExpectedA ttributeValue</code>	<p>Gunakan parameter ini untuk menentukan apakah nilai sudah ada atau belum untuk pasangan nama-nilai atribut.</p> <p>Notasi JSON berikut mengganti item jika atribut "Warna" belum ada untuk item tersebut:</p> <pre data-bbox="594 663 1029 827">"Expected" : {"Color":{"Exists":false}}</pre> <p>Notasi JSON berikut memeriksa untuk melihat apakah atribut dengan nama "Warna" memiliki nilai "Kuning" yang ada sebelum mengganti item:</p> <pre data-bbox="594 1171 1029 1373">"Expected" : {"Color":{"Exists":true,{"Value":{"S":"Yellow"}}}}</pre> <p>Secara default, jika Anda menggunakan parameter <code>Expected</code> dan menyediakan <code>Value</code>, DynamoDB mengasumsikan bahwa atribut ada dan memiliki nilai terkini untuk diganti. Jadi Anda tidak perlu menentukan <code>{"Exists":true}</code> , karena tersirat.</p>	Tidak

Nama	Deskripsi	Wajib
	<p>Anda dapat mempersingkat permintaan untuk:</p> <pre data-bbox="592 331 1031 493">"Expected" : {"Color":{"Value": {"S":"Yellow"}}}</pre> <p>Note</p> <p>Jika Anda menentukan {"Exists":true} tanpa nilai atribut untuk memeriksa , DynamoDB mengembalikan kesalahan.</p>	
ReturnValues	<p>Gunakan parameter ini jika Anda ingin mendapatkan pasangan nama-nilai atribut sebelum pasangan tersebut diperbarui dengan permintaan PutItem. Nilai parameter yang memungkinkan adalah NONE (default) atau ALL_OLD. Jika ALL_OLD ditentukan, dan PutItem menimpa pasangan nama-nilai atribut, isi dari item lama akan dikembalikan. Jika parameter ini tidak tersedia atau NONE, tidak ada yang dikembalikan.</p> <p>Jenis: String</p>	Tidak

Respons

Sintaksis

Contoh sintaks berikut mengasumsikan permintaan ditentukan dalam parameter `ReturnValues` dalam `ALL_OLD`; jika tidak, respons hanya memiliki elemen `ConsumedCapacityUnits`.

```
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
}
```

Nama	Deskripsi
<code>Attributes</code>	<p>Nilai atribut sebelum operasi put, tetapi hanya jika parameter <code>ReturnValues</code> ditentukan sebagai dalam permintaan <code>ALL_OLD</code>.</p> <p>Jenis: Peta pasangan nama-nilai atribut.</p>
<code>ConsumedCapacityUnits</code>	<p>Jumlah unit kapasitas tulis yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang Anda sediakan. Untuk informasi selengkapnya, lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB.</p> <p>Jenis: Jumlah</p>

Kesalahan Khusus

Kesalahan	Deskripsi
ConditionalCheckFailedException	Pemeriksaan bersyarat gagal. Nilai atribut yang diharapkan tidak ditemukan.
ResourceNotFoundException	Item atau atribut yang ditentukan tidak ditemukan.

Contoh

Untuk contoh menggunakan SDK AWS, lihat [Bekerja dengan item dan atribut](#).

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat rendah DynamoDB.
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"
}
```

Contoh respons

```
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
}
```

Tindakan terkait

- [UpdateItem](#)
- [DeleteItem](#)
- [GetItem](#)
- [BatchGetItem](#)

Kueri

Important

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

SEBUAHQueryoperasi mendapatkan nilai dari satu atau beberapa item dan atributnya berdasarkan kunci utama (Queryhanya tersedia untuk hash-and-range tabel kunci utama). Anda harus memberikan HashKeyValue khusus, dan dapat mempersempit ruang lingkup kueri menggunakan operator perbandingan pada RangeKeyValue kunci utama. Penggunaan parameter ScanIndexForward untuk mendapatkan hasil dalam urutan maju atau mundur dengan kunci rentang.

Pertanyaan yang tidak mengembalikan hasil menggunakan unit kapasitas baca minimum sesuai dengan jenis pembacaan.

Note

Jika jumlah total item yang memenuhi parameter kueri melampaui batas 1MB, kueri berhenti dan hasilnya dikembalikan ke pengguna dengan `LastEvaluatedKey` untuk melanjutkan kueri dalam operasi berikutnya. Tidak seperti operasi Pemindaian, operasi Kueri tidak pernah mengembalikan set hasil kosong dan sebuah `LastEvaluatedKey`. `LastEvaluatedKey` diberikan hanya jika hasil melampaui 1MB, atau jika Anda telah menggunakan parameter `Limit`. Hasilnya dapat diatur untuk pembacaan yang konsisten menggunakan parameter `ConsistentRead`.

Permintaan

Sintaksis

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"]},
}
```


Nama	Deskripsi	Wajib
TableName	Nama tabel yang berisi item yang diminta.	Ya

Nama	Deskripsi	Wajib
	Jenis: Tali	
AttributesToGet	Susunan nama Atribut. Jika nama atribut tidak ditentukan maka semua atribut akan dikembalikan. Jika beberapa atribut tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil. Jenis: Susunan	Tidak

Nama	Deskripsi	Wajib
Limit	<p>Jumlah maksimum item untuk dikembalikan (tidak selalu merupakan jumlah item yang cocok). Jika DynamoDB memproses jumlah item hingga batasnya sekaligus mengajukan kueri tabel, DynamoDB lalu menghentikan kueri dan mengembalikan nilai-nilai yang cocok hingga titik tersebut, dan <code>LastEvaluatedKey</code> untuk diterapkan dalam operasi berikutnya guna melanjutkan kueri. Selain itu, jika ukuran set hasil melampaui 1MB sebelum DynamoDB mencapai batas ini, DynamoDB akan menghentikan kueri dan mengembalikan nilai-nilai yang cocok, dan <code>LastEvaluatedKey</code> untuk diterapkan dalam operasi berikutnya guna melanjutkan kueri.</p> <p>Jenis: Jumlah</p>	Tidak
ConsistentRead	<p>Jika diatur menjadi <code>true</code>, pembacaan yang konsisten akan diterbitkan, jika tidak, pada akhirnya yang konsisten yang digunakan.</p> <p>Jenis: Boolean</p>	Tidak

Nama	Deskripsi	Wajib
Count	<p>Jika diatur menjadi <code>true</code>, DynamoDB mengembalikan jumlah total item yang cocok dengan parameter kueri, bukan daftar item yang cocok dan atributnya. Anda dapat menerapkan parameter <code>Limit</code> pada kueri yang hanya menghitung.</p> <p>Jangan diatur <code>Count</code> menjadi <code>true</code> saat menyediakan daftar <code>AttributesToGet</code> , jika tidak, DynamoDB mengembalikan kesalahan validasi. Untuk informasi selengkapnya, lihat Menghitung item dalam hasil.</p> <p>Jenis: Boolean</p>	Tidak
HashKeyValue	<p>Nilai atribut komponen hash dari kunci utama komposit.</p> <p>Jenis: String, Angka, atau Biner</p>	Ya

Nama	Deskripsi	Wajib
RangeKeyCondition	<p>Sebuah kontainer untuk nilai atribut dan operator perbandingan untuk digunakan dalam kueri. Permintaan kueri tidak memerlukan RangeKeyCondition . Jika Anda hanya memberikan HashKeyValue , DynamoDB mengembalikan semua item dengan nilai elemen kunci hash yang ditentukan.</p> <p>Jenis: Map</p>	Tidak
RangeKeyCondition : AttributeValueList	<p>Nilai atribut untuk mengevaluasi parameter kueri. AttributeValueList berisi satu nilai atribut, kecuali jika perbandingan BETWEEN ditentukan. Untuk perbandingan BETWEEN, AttributeValueList berisi dua nilai atribut.</p> <p>Jenis: PetaAttributeValue keComparisonOperator .</p>	Tidak

Nama	Deskripsi	Wajib
RangeKeyCondition : ComparisonOperator	<p>Kriteria untuk mengevaluasi atribut yang disediakan, seperti sama dengan, lebih besar dari, dll. Berikut ini adalah operator perbandingan yang valid untuk operasi kueri.</p> <div data-bbox="591 541 1029 1854" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px;"><p> Note</p><p>Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada nilai-nilai kode karakter ASCII. Sebagai contoh, a lebih besar dari A, dan aa lebih besar dari B. Untuk daftar nilai kode, lihat http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters. Untuk Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner, contohnya ketika mengevaluasi ekspresi kueri.</p></div>	Tidak

Nama	Deskripsi	Wajib
	Jenis: String atau Biner	
	<p>EQ : Sama dengan.</p> <p>Untuk EQ, Attribute ValueList hanya dapat berisi satu Attribute Value dari jenis String, Angka, atau Biner (bukan suatu set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S":"6"} tidak sama dengan {"N":"6"} . Selain itu, {"N":"6"} tidak sama dengan {"NS":["6", "2", "1"]}.</p>	

Nama	Deskripsi	Wajib
	<p>LE : Kurang dari atau sama dengan.</p> <p>Untuk LE, Attribute ValueList hanya dapat berisi satu Attribute Value dari jenis String, Angka, atau Biner (bukan suatu set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S":"6"} tidak sama dengan {"N":"6"} . Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]} .</p>	

Nama	Deskripsi	Wajib
	<p>LT : Kurang dari.</p> <p>Untuk LT, Attribute ValueList hanya dapat berisi satu Attribute Value dari jenis String, Angka, atau Biner (bukan suatu set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S":"6"} tidak sama dengan {"N":"6"} . Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]} .</p>	

Nama	Deskripsi	Wajib
	<p>GE : Lebih besar dari atau sama dengan.</p> <p>Untuk GE, Attribute ValueList hanya dapat berisi satu Attribute Value dari jenis String, Angka, atau Biner (bukan suatu set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S":"6"} tidak sama dengan {"N":"6"} . Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]} .</p>	

Nama	Deskripsi	Wajib
	<p>GT : Lebih besar dari.</p> <p>Untuk GT, <code>AttributeValueList</code> hanya dapat berisi satu <code>AttributeValue</code> dari jenis String, Angka, atau Biner (bukan suatu set). Jika item berisi <code>AttributeValue</code> dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, <code>{"S":"6"}</code> tidak sama dengan <code>{"N":"6"}</code> . Selain itu, <code>{"N":"6"}</code> tidak sebanding dengan <code>{"NS":["6", "2", "1"]}</code> .</p>	
	<p>BEGINS_WITH : memeriksa awalan.</p> <p>Untuk <code>BEGINS_WITH</code> , <code>AttributeValueList</code> hanya dapat berisi satu <code>AttributeValue</code> dari jenis String atau Biner (bukan Angka atau suatu set). Atribut target perbandingan harus berupa String atau Biner (bukan Angka atau suatu set).</p>	

Nama	Deskripsi	Wajib
	<p>BETWEEN : Lebih besar dari, atau sama dengan, nilai pertama dan kurang dari, atau sama dengan, nilai kedua.</p> <p>Untuk BETWEEN, Attribute ValueList harus berisi dua elemen Attribute Value dari jenis yang sama, baik String, Angka, atau Biner (bukan suatu set). Suatu atribut target cocok jika nilai target lebih besar dari, atau sama dengan, elemen pertama dan kurang dari, atau sama dengan, elemen kedua. Jika item berisi Attribute Value dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S": "6"} tidak sebanding dengan {"N": "6"} . Selain itu, {"N": "6"} tidak sebanding dengan {"NS": ["6", "2", "1"]}.</p>	

Nama	Deskripsi	Wajib
ScanIndexForward	<p>Menentukan traversal naik atau turun indeks. DynamoDB mengembalikan hasil yang mencerminkan urutan yang diminta ditentukan oleh kunci rentang: Jika jenis data adalah Angka, hasilnya dikembalikan dalam urutan numerik; jika tidak, traversal didasarkan pada nilai kode karakter ASCII.</p> <p>Jenis: Boolean</p> <p>Defaultnya adalah <code>true</code> (naik).</p>	Tidak

Nama	Deskripsi	Wajib
ExclusiveStartKey	<p>Kunci primer dari item untuk melanjutkan kueri sebelumnya</p> <p>a. Kueri sebelumnya mungkin memberikan nilai ini sebagai LastEvaluatedKey jika operasi kueri terganggu sebelum menyelesaikan kueri; baik karena ukuran set hasil atau parameter Limit. LastEvaluatedKey dapat diteruskan kembali dalam permintaan kueri baru untuk melanjutkan operasi dari titik tersebut.</p> <p>Jenis: HashKeyElement , atau HashKeyElement dan RangeKeyElement untuk kunci utama komposit.</p>	Tidak

Respons

Sintaksis

```
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":[{"AttributeNames":{"S":"AttributeValue1"},
"AttributeNames":{"N":"AttributeValue2"},
"AttributeNames":{"S":"AttributeValue3"}
},{
"AttributeNames":{"S":"AttributeValue3"},
"AttributeNames":{"N":"AttributeValue4"},
"AttributeNames":{"S":"AttributeValue3"},
```

```

    "AttributeName5":{"B":"dmFsdWU="}
  ]],
  "LastEvaluatedKey":{"HashKeyElement":{"AttributeValue3":"S"},
    "RangeKeyElement":{"AttributeValue4":"N"}}
  },
  "ConsumedCapacityUnits":1
}

```

Nama	Deskripsi
Items	<p>Atribut item yang memenuhi parameter kueri.</p> <p>Jenis: Peta nama atribut serta nilai dan jenis data.</p>
Count	<p>Jumlah item dalam respons. Untuk informasi selengkapnya, lihat Menghitung item dalam hasil.</p> <p>Jenis: Jumlah</p>
LastEvaluatedKey	<p>Kunci utama dari item di mana operasi kueri berhenti, termasuk set hasil sebelumnya. Gunakan nilai ini untuk memulai operasi baru terkecuali nilai ini dalam permintaan baru.</p> <p>LastEvaluatedKey adalah null ketika seluruh set hasil kueri selesai (yaitu operasi memproses “halaman terakhir”).</p> <p>Jenis: HashKeyElement , atau HashKeyElement dan RangeKeyElement untuk kunci utama komposit.</p>
ConsumedCapacityUnits	<p>Jumlah unit kapasitas baca yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang Anda sediakan. Untuk informasi selengkapnya, lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB.</p>

Nama	Deskripsi
	Jenis: Jumlah

Kesalahan Khusus

Kesalahan	Deskripsi
ResourceNotFoundException	Tabel yang ditentukan tidak ditemukan.

Contoh

Untuk contoh menggunakan SDK AWS, lihat [Operasi kueri di DynamoDB](#).

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat rendah DynamoDB.
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"}
 }
}
```

Contoh respons

```
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
}
```

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat rendah DynamoDB.
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}
```

Contoh respons

```
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  
}
```

Tindakan terkait

- [Pemindaian](#)

Pemindaian

Important

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

Operasi Scan mengembalikan satu atau beberapa item dan atributnya dengan melakukan pemindaian menyeluruh tabel. Berikan ScanFilter untuk mendapatkan hasil yang lebih spesifik.

Note

Jika jumlah total item yang dipindai melampaui batas 1MB, pemindaian berhenti dan hasil dikembalikan ke pengguna dengan LastEvaluatedKey untuk melanjutkan pemindaian dalam operasi berikutnya. Hasilnya juga mencakup jumlah item yang melampaui batas. Pemindaian dapat mengakibatkan tidak adanya tabel data yang memenuhi kriteria filter. Set hasil pada akhirnya konsisten.

Permintaan

Sintaksis

```
// This header is abbreviated.  
// For a sample of a complete header, see API tingkat rendah DynamoDB.  
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"]},
}
```

Nama	Deskripsi	Wajib
TableName	Nama tabel yang berisi item yang diminta. Jenis: Tali	Ya
AttributesToGet	Susunan nama Atribut. Jika nama atribut tidak ditentukan maka semua atribut akan dikembalikan. Jika beberapa atribut tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil. Jenis: Susunan	Tidak
Limit	Jumlah maksimum item untuk dievaluasi (belum tentu jumlah item yang cocok). Jika DynamoDB memproses jumlah item hingga batasnya sekaligus memproses hasil, DynamoDB lalu berhenti	Tidak

Nama	Deskripsi	Wajib
	<p>dan mengembalikan nilai-nilai yang cocok hingga titik tersebut, dan <code>LastEvaluatedKey</code> untuk diterapkan dalam operasi berikutnya guna melanjutkan mengambil item. Selain itu, jika ukuran set data yang dipindai melampaui 1MB sebelum DynamoDB mencapai batas ini, DynamoDB akan menghentikan pemindaian dan mengembalikan nilai-nilai yang cocok hingga batas tersebut, dan <code>LastEvaluatedKey</code> untuk diterapkan dalam operasi berikutnya guna melanjutkan pemindaian.</p> <p>Jenis: Jumlah</p>	

Nama	Deskripsi	Wajib
Count	<p>Jika diatur menjadi <code>true</code>, DynamoDB mengembalikan jumlah total item untuk operasi Pemindaian, bahkan jika operasi tidak memiliki item yang cocok untuk filter yang ditetapkan. Anda dapat menerapkan parameter <code>Batas</code> untuk pemindaian yang hanya menghitung.</p> <p>Jangan diatur <code>Count</code> menjadi <code>true</code> saat menyediakan daftar <code>AttributesToGet</code> , jika tidak, DynamoDB mengembalikan kesalahan validasi. Untuk informasi selengkapnya, lihat Menghitung item dalam hasil.</p> <p>Jenis: Boolean</p>	Tidak
ScanFilter	<p>Mengevaluasi hasil pemindaian dan mengembalikan hanya nilai-nilai yang diinginkan. Beberapa syarat diperlakukan sebagai operasi "DAN": semua syarat harus dipenuhi agar disertakan dalam hasil.</p> <p>Jenis: Peta nama atribut untuk nilai dengan operator perbandingan.</p>	Tidak

Nama	Deskripsi	Wajib
ScanFilter :Attribute ValueList	Nilai dan syarat untuk mengevaluasi hasil pemindaia n untuk filter. Jenis: PetaAttribute Value keCondition .	Tidak

Nama	Deskripsi	Wajib
ScanFilter : ComparisonOperator	<p>Kriteria untuk mengevaluasi atribut yang disediakan, seperti sama dengan, lebih besar dari, dll. Berikut ini adalah operator perbandingan yang valid untuk operasi pemindaian.</p> <div data-bbox="591 590 1029 1766" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px;"><p> Note</p><p>Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada nilai-nilai kode karakter ASCII. Sebagai contoh, a lebih besar dari A, dan aa lebih besar dari B. Untuk daftar nilai kode, lihat http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters. Untuk Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner, contohnya ketika</p></div>	Tidak

Nama	Deskripsi	Wajib
	<p data-bbox="591 205 1031 338">mengevaluasi ekspresi kueri.</p> <p data-bbox="591 405 922 443">Jenis: String atau Biner</p>	
	<p data-bbox="591 485 862 522">EQ : Sama dengan.</p> <p data-bbox="591 564 1003 1272">Untuk EQ, Attribute ValueList hanya dapat berisi satu Attribute Value dari jenis String, Angka, atau Biner (bukan suatu set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S": "6"} tidak sama dengan {"N": "6"} . Selain itu, {"N": "6"} tidak sama dengan {"NS": ["6", "2", "1"]}.</p>	

Nama	Deskripsi	Wajib
	<p>NE : Tidak Sama dengan.</p> <p>Untuk NE, Attribute ValueList hanya dapat berisi satu Attribute Value dari jenis String, Angka, atau Biner (bukan suatu set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S":"6"} tidak sama dengan {"N":"6"} . Selain itu, {"N":"6"} tidak sama dengan {"NS":["6", "2", "1"]}.</p>	

Nama	Deskripsi	Wajib
	<p>LE : Kurang dari atau sama dengan.</p> <p>Untuk LE, Attribute ValueList hanya dapat berisi satu Attribute Value dari jenis String, Angka, atau Biner (bukan suatu set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S":"6"} tidak sama dengan {"N":"6"} . Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]} .</p>	

Nama	Deskripsi	Wajib
	<p>LT : Kurang dari.</p> <p>Untuk LT, Attribute ValueList hanya dapat berisi satu Attribute Value dari jenis String, Angka, atau Biner (bukan suatu set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S":"6"} tidak sama dengan {"N":"6"} . Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]} .</p>	

Nama	Deskripsi	Wajib
	<p>GE : Lebih besar dari atau sama dengan.</p> <p>Untuk GE, Attribute ValueList hanya dapat berisi satu Attribute Value dari jenis String, Angka, atau Biner (bukan suatu set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S":"6"} tidak sama dengan {"N":"6"} . Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]} .</p>	

Nama	Deskripsi	Wajib
	<p>GT : Lebih besar dari.</p> <p>Untuk GT, Attribute ValueList hanya dapat berisi satu Attribute Value dari jenis String, Angka, atau Biner (bukan suatu set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S":"6"} tidak sama dengan {"N":"6"} . Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]} .</p>	
	NOT_NULL : Atribut ada.	
	NULL : Atribut tidak ada.	

Nama	Deskripsi	Wajib
	<p>CONTAINS : memeriksa urutan, atau nilai dalam suatu set.</p> <p>Untuk CONTAINS, Attribute ValueList hanya dapat berisi satu Attribute Value dari jenis String, Angka, atau Biner (bukan suatu set). Jika atribut target perbandingan adalah String, maka operasi memeriksa kecocokan substring. Jika atribut target perbandingan adalah Biner, maka operasi mencari urutan dari target yang cocok dengan input. Jika atribut target perbandingan adalah suatu set ("SS", "NS", atau "BS"), maka operasi memeriksa anggota set tersebut (bukan sebagai substring).</p>	

Nama	Deskripsi	Wajib
	<p>NOT_CONTAINS : memeriksa tidak adanya urutan, atau tidak adanya nilai dalam suatu set.</p> <p>Untuk NOT_CONTAINS , AttributeValueList hanya dapat berisi satu AttributeValue dari jenis String, Angka, atau Biner (bukan suatu set). Jika atribut target perbandingan adalah String, maka operasi memeriksa tidak adanya kecocokan substring . Jika atribut target perbandingan adalah Biner, maka operasi mencari tidak adanya urutan pada target yang cocok dengan input. Jika atribut target perbandingan adalah suatu set ("SS", "NS", atau "BS"), maka operasi memeriksa tidak adanya anggota set tersebut (bukan sebagai substring).</p>	

Nama	Deskripsi	Wajib
	<p>BEGINS_WITH : memeriksa awalan.</p> <p>Untuk BEGINS_WITH , <code>AttributeValueList</code> hanya dapat berisi satu <code>AttributeValue</code> dari jenis String atau Biner (bukan Angka atau suatu set). Atribut target perbandingan harus berupa String atau Biner (bukan Angka atau suatu set).</p>	
	<p>IN : memeriksa kecocokan yang tepat.</p> <p>Untuk IN, <code>AttributeValueList</code> dapat berisi lebih dari satu <code>AttributeValue</code> jenis String, Angka, atau Biner (bukan set). Atribut target perbandingan harus dari jenis yang sama dan nilai yang tepat agar cocok. Suatu String tidak pernah cocok dengan set String.</p>	

Nama	Deskripsi	Wajib
	<p>BETWEEN : Lebih besar dari, atau sama dengan, nilai pertama dan kurang dari, atau sama dengan, nilai kedua.</p> <p>Untuk BETWEEN, Attribute ValueList harus berisi dua elemen Attribute Value dari jenis yang sama, baik String, Angka, atau Biner (bukan suatu set). Suatu atribut target cocok jika nilai target lebih besar dari, atau sama dengan, elemen pertama dan kurang dari, atau sama dengan, elemen kedua. Jika item berisi Attribute Value dengan jenis berbeda dari yang ditentukan dalam permintaan, nilai tidak cocok. Misalnya, {"S": "6"} tidak sebanding dengan {"N": "6"} . Selain itu, {"N": "6"} tidak sebanding dengan {"NS": ["6", "2", "1"]}</p>	

Nama	Deskripsi	Wajib
ExclusiveStartKey	<p>Kunci utama item yang untuk melanjutkan pemindaian sebelumnya. Pemindaian sebelumnya mungkin memberikan nilai ini jika operasi pemindaian terganggu sebelum memindai seluruh tabel; baik karena ukuran set hasil atau parameter Limit. LastEvaluatedKey dapat diteruskan kembali dalam permintaan pemindaian baru untuk melanjutkan operasi dari titik tersebut.</p> <p>Jenis: HashKeyElement , atau HashKeyElement dan RangeKeyElement untuk kunci utama komposit.</p>	Tidak

Respons

Sintaksis

```
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":[{"AttributeNames":{"S":"AttributeValue1"},
"AttributeNames":{"S":"AttributeValue2"},
"AttributeNames":{"S":"AttributeValue3"}
},{
"AttributeNames":{"S":"AttributeValue4"},
"AttributeNames":{"S":"AttributeValue5"},
"AttributeNames":{"S":"AttributeValue6"},
```

```

    "AttributeName5":{"B":"dmFsdWU="}
  ]],
  "LastEvaluatedKey":
    {"HashKeyElement":{"S":"AttributeName1"},
     "RangeKeyElement":{"N":"AttributeName2"},
    "ConsumedCapacityUnits":1,
    "ScannedCount":2}
}

```

Nama	Deskripsi
Items	<p>Kontainer untuk atribut yang memenuhi parameter operasi.</p> <p>Jenis: Peta nama atribut serta nilai serta nilai dan jenis data.</p>
Count	<p>Jumlah item dalam respons. Untuk informasi selengkapnya, lihat Menghitung item dalam hasil.</p> <p>Jenis: Jumlah</p>
ScannedCount	<p>Jumlah item dalam pemindaian lengkap sebelum filter diterapkan. Nilai ScannedCount yang tinggi dengan hasil Count yang sedikit, atau tidak ada menunjukkan operasi Pemindaian yang tidak efisien. Untuk informasi selengkapnya, lihat Menghitung item dalam hasil.</p> <p>Jenis: Jumlah</p>
LastEvaluatedKey	<p>Kunci utama item di mana operasi pemindaian berhenti. Masukkan nilai ini dalam operasi pemindaian berikutnya untuk melanjutkan operasi dari titik tersebut.</p>

Nama	Deskripsi
	LastEvaluatedKey adalah null ketika seluruh set hasil pemindaian selesai (yaitu operasi memproses “halaman terakhir”).
ConsumedCapacityUnits	Jumlah unit kapasitas baca yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang Anda sediakan. Untuk informasi selengkapnya, lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB . Jenis: Jumlah

Kesalahan Khusus

Kesalahan	Deskripsi
ResourceNotFoundException	Tabel yang ditentukan tidak ditemukan.

Contoh

Untuk contoh menggunakan SDK AWS, lihat [Bekerja dengan pemindaian di DynamoDB](#).

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat rendah DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable","ScanFilter":{}}
```

Contoh respons

```
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}
```

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat rendah DynamoDB.
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"}
}
```

```
}

```

Contoh respons

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNS05AEMVJF66Q9ASUAAJG
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
}
```

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat rendah DynamoDB.
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"}}
```

```
}
```

Contoh respons

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNS05AEMVJF66Q9ASUAAJG
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
 }
```

Tindakan terkait

- [Kueri](#)
- [BatchGetItem](#)

UpdateItem

Important

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

Mengedit atribut item yang ada. Anda dapat melakukan pembaruan bersyarat (memasukkan pasangan nama-nilai atribut baru jika tidak ada, atau mengganti pasangan nama-nilai yang ada jika memiliki nilai atribut tertentu yang diharapkan).

Note

Anda tidak dapat memperbarui atribut kunci utama menggunakan `UpdateItem`. Sebagai gantinya, hapus item dan gunakan `PutItem` untuk membuat item baru dengan atribut baru.

Parameter `UpdateItem` operasi meliputi `Action` parameter, yang mendefinisikan cara melakukan pembaruan. Anda dapat menempatkan, menghapus, atau menambahkan nilai atribut.

Nilai atribut tidak boleh null; atribut jenis string dan biner harus memiliki panjang lebih dari nol; dan atribut jenis set tidak boleh kosong. Permintaan dengan nilai kosong akan ditolak dengan `ValidationException`.

Jika item yang ada memiliki kunci utama yang ditentukan:

- **PUT** – Menambahkan atribut yang ditentukan. Jika atribut ada, atribut akan digantikan oleh nilai baru.
- **DELETE** – Jika tidak ada nilai yang ditentukan, ini akan menghapus atribut dan nilainya. Jika suatu set nilai ditentukan, maka nilai-nilai dalam set yang ditentukan akan dihapus dari set lama. Jadi jika nilai atribut berisi [a,b,c] dan tindakan menghapus berisi [a,c], maka nilai atribut akhir adalah [b]. Jenis nilai yang ditentukan harus cocok dengan jenis nilai yang ada. Menentukan suatu set kosong tidaklah valid.
- **ADD** – Gunakan hanya tindakan `add` untuk angka atau jika atribut target adalah suatu set (termasuk set string). `ADD` tidak berfungsi jika atribut target adalah nilai string tunggal atau nilai biner skalar. Nilai yang ditentukan ditambahkan ke nilai numerik (menambah atau mengurangi nilai numerik yang ada) atau ditambahkan sebagai nilai tambahan dalam suatu set string. Jika suatu set nilai ditentukan, nilai-nilai ditambahkan ke set yang ada. Misalnya jika set asli adalah [1,2] dan nilai yang disediakan adalah [3], maka setelah operasi `add`, set adalah [1,2,3], bukan [4,5]. Terjadi kesalahan jika tindakan `Add` ditentukan untuk atribut set dan jenis atribut yang ditentukan tidak cocok dengan jenis set yang ada.

Jika Anda menggunakan `ADD` untuk atribut yang tidak ada, atribut dan nilai-nilainya ditambahkan ke item.

Jika tidak ada item yang cocok dengan kunci utama yang ditentukan:

- PUT – Membuat item baru dengan kunci utama yang ditentukan. Kemudian menambahkan atribut yang ditentukan.
- DELETE – Tidak ada yang terjadi.
- ADD – Membuat item dengan kunci utama dan angka yang disediakan (atau set angka) untuk nilai atribut. Tidak berlaku untuk jenis string atau biner.

Note

Jika Anda menggunakan ADD untuk menambah atau mengurangi nilai angka untuk item yang tidak ada sebelum pembaruan, DynamoDB menggunakan 0 sebagai nilai awal. Selain itu, jika Anda memperbarui item menggunakan ADD guna menambah atau mengurangi nilai angka untuk atribut yang tidak ada sebelum pembaruan (tetapi item ada), DynamoDB menggunakan 0 sebagai nilai awal. Sebagai contoh, Anda menggunakan ADD untuk menambahkan +3 untuk atribut yang tidak ada sebelum pembaruan. DynamoDB menggunakan 0 untuk nilai awal, dan nilai setelah pembaruan adalah 3.

Untuk informasi lebih lanjut tentang operasi ini, lihat [Bekerja dengan item dan atribut](#).

Permintaan

Sintaksis

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"
```

}

Nama	Deskripsi	Wajib
TableName	<p>Nama tabel yang berisi item untuk diperbarui.</p> <p>Jenis: Tali</p>	Ya
Key	<p>Kunci utama yang menentukan item. Untuk informasi lebih lanjut tentang kunci utama, lihat Kunci primer.</p> <p>Jenis: Peta dari HashKeyElement untuk nilainya dan RangeKeyElement untuk nilainya.</p>	Ya
AttributeUpdates	<p>Peta nama atribut dengan nilai dan tindakan baru untuk diperbarui. Nama atribut menentukan atribut untuk memodifikasi, dan tidak dapat berisi atribut kunci utama.</p> <p>Jenis: Peta nama atribut, nilai, dan tindakan untuk pembaruan atribut.</p>	
AttributeUpdates :Action	<p>Menentukan cara melakukan pembaruan. Kemungkinan nilai: PUT (default), ADD atau DELETE. Semantik dijelaskan dalam UpdateItem deskripsi.</p> <p>Jenis: String</p> <p>Default: PUT</p>	Tidak

Nama	Deskripsi	Wajib
Expected	<p>Menunjuk atribut untuk pembaruan bersyarat. Parameter Expected memungkinkan Anda untuk memberikan nama atribut, dan apakah DynamoDB harus memeriksa keberadaan nilai atribut; atau apakah nilai atribut ada dan memiliki nilai tertentu sebelum mengubahnya.</p> <p>Jenis: Peta nama atribut.</p>	Tidak
Expected:Attribute Name	<p>Nama atribut untuk put bersyarat.</p> <p>Jenis: String</p>	Tidak

Nama	Deskripsi	Wajib
Expected:Attribute Name: ExpectedAttribute Value	<p>Gunakan parameter ini untuk menentukan apakah nilai sudah ada atau belum untuk pasangan nama-nilai atribut.</p> <p>Notasi JSON berikut memperbarui item jika atribut "Warna" belum ada untuk item tersebut:</p> <pre data-bbox="594 663 1029 827">"Expected" : {"Color":{"Exists":false}}</pre> <p>Notasi JSON berikut memeriksa apakah atribut dengan nama "Warna" memiliki nilai yang ada berupa "Kuning" sebelum memperbarui item:</p> <pre data-bbox="594 1171 1029 1373">"Expected" : {"Color":{"Exists":true}, {"Value": {"S":"Yellow"}}</pre> <p>Secara default, jika Anda menggunakan parameter Expected dan menyediakan Value, DynamoDB mengasumsikan bahwa atribut ada dan memiliki nilai terkini untuk diganti. Jadi Anda tidak perlu menentukan {"Exists":true} , karena tersirat.</p>	Tidak

Nama	Deskripsi	Wajib
	<p>Anda dapat mempersingkat permintaan untuk:</p> <pre data-bbox="594 331 1029 491">"Expected" : {"Color":{"Value": {"S":"Yellow"}}}</pre> <p> Note</p> <p>Jika Anda menentukan {"Exists":true} tanpa nilai atribut untuk memeriksa , DynamoDB mengembalikan kesalahan.</p>	

Nama	Deskripsi	Wajib
ReturnValues	<p>Gunakan parameter ini jika Anda ingin mendapatkan pasangan nama-nilai atribut sebelum pasangan tersebut diperbarui dengan permintaan UpdateItem . Nilai parameter yang mungkin adalah NONE (default) atau ALL_OLD, UPDATED_OLD , ALL_NEW atau UPDATED_NEW . Jika ALL_OLD ditentukan, dan UpdateItem menerima pasangan nama-nilai atribut, isi dari item lama akan dikembalikan. Jika parameter ini tidak tersedia atau NONE, tidak ada yang dikembalikan. Jika ALL_NEW ditentukan, maka semua atribut versi baru dari item tersebut akan dikembalikan. Jika UPDATED_NEW ditentukan, maka versi baru hanya dari atribut yang diperbarui akan dikembalikan.</p> <p>Jenis: String</p>	Tidak

Respons

Sintaksis

Contoh sintaks berikut mengasumsikan permintaan ditentukan dalam parameter ReturnValues dalam ALL_OLD; jika tidak, respons hanya memiliki elemen ConsumedCapacityUnits.

```

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
}

```

Nama	Deskripsi
Attributes	<p>Sebuah peta pasangan nama-nilai atribut, tetapi hanya jika parameter ReturnValues ditentukan sebagai sesuatu selain NONE dalam permintaan.</p> <p>Jenis: Peta pasangan nama-nilai atribut.</p>
ConsumedCapacityUnits	<p>Jumlah unit kapasitas tulis yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang Anda sediakan. Untuk informasi selengkapnya, lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB.</p> <p>Jenis: Jumlah</p>

Kesalahan Khusus

Kesalahan	Deskripsi
<code>ConditionalCheckFailedException</code>	Pemeriksaan bersyarat gagal. Atribut ("+ name +") nilai adalah ("+ value +") tapi diharapkan ("+ expValue +")
<code>ResourceNotFoundExceptions</code>	Item atau atribut yang ditentukan tidak ditemukan.

Contoh

Untuk contoh menggunakan SDK AWS, lihat [Bekerja dengan item dan atribut](#).

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat rendah DynamoDB.
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"
}
```

Contoh respons

```
HTTP/1.1 200 OK
x-amzn-RequestId: 5IMH07F01Q9P7Q6QMKKMI3R3QRVV4KQNS05AEMVJF66Q9ASUAAJG
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
}
```

Tindakan terkait

- [PutItem](#)
- [DeleteItem](#)

UpdateTable

Important

Bagian ini mengacu pada API versi 05/12/2011, yang sudah usang dan tidak boleh digunakan untuk aplikasi baru.

Untuk dokumentasi tentang API tingkat rendah saat ini, lihat [Referensi API Amazon DynamoDB](#).

Deskripsi

Memperbarui throughput yang disediakan untuk tabel yang ditentukan. Mengatur throughput untuk tabel membantu Anda mengelola kinerja dan merupakan bagian dari fitur throughput yang disediakan dalam DynamoDB. Untuk informasi selengkapnya, lihat [Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB](#).

Nilai throughput yang disediakan dapat ditingkatkan atau diturunkan berdasarkan nilai maksimum dan minimum yang tercantum dalam [Layanan, akun, dan tabel kuota di Amazon DynamoDB](#).

Tabel harus berada dalam status ACTIVE agar operasi ini berhasil. UpdateTable adalah operasi asinkron; saat menjalankan operasi, tabel berada dalam UPDATING status. Meskipun berada dalam status UPDATING, tabel masih memiliki throughput yang disediakan sejak sebelum panggilan.

Pengaturan throughput yang disediakan baru ini berlaku hanya ketika tabel kembali ke ACTIVE negara setelah UpdateTable operasi.

Permintaan

Sintaksis

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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}
}
```

Nama	Deskripsi	Wajib
TableName	Nama ruang tabel untuk diperbarui. Jenis: Tali	Ya
ProvisionedThroughput	Throughput baru untuk tabel tertentu, yang terdiri dari nilai-nilai untuk ReadCapacityUnits dan WriteCapacityUnits. Lihat Mengelola pengaturan pada tabel kapasitas yang disediakan DynamoDB . Jenis: Susunan	Ya
ProvisionedThroughput:ReadCapacityUnits	Menetapkan jumlah minimum ReadCapacityUnits yang konsisten digunakan per detik untuk tabel tertentu sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya.	Ya

Nama	Deskripsi	Wajib
	<p>Operasi baca yang konsisten pada bagian akhir memerlukan lebih sedikit usaha daripada operasi baca yang konsisten, jadi pengaturan 50 ReadCapacityUnits yang konsisten per detik akan menghasilkan 100 ReadCapacityUnits yang konsisten pada bagian akhir per detik.</p> <p>Jenis: Jumlah</p>	
ProvisionedThroughput :WriteCapacityUnits	<p>Menetapkan jumlah minimum WriteCapacityUnits yang digunakan per detik untuk tabel tertentu sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya.</p> <p>Jenis: Jumlah</p>	Ya

Respons

Sintaksis

```

HTTP/1.1 200 OK
x-amzn-RequestId: CS0C7TJPLR000KIRLG0HVAICUFVV4KQNS05AEMVJF66Q9ASUAAJG
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"}}

```

Nama	Deskripsi
CreationDateTime	<p>Tanggal ketika tabel dibuat.</p> <p>Jenis: Jumlah</p>
KeySchema	<p>Struktur kunci utama (sederhana atau komposit) untuk tabel. Pasangan nama-nilai untuk HashKeyElement diperlukan, dan pasangan nama-nilai untuk RangeKeyElement adalah opsional (hanya diperlukan untuk kunci utama komposit). Ukuran kunci hash maksimum adalah 2048 byte. Ukuran kunci rentang maksimum adalah 1024 byte. Kedua batas tersebut diberlakukan secara terpisah (yaitu Anda dapat memiliki kombinasi hash + rentang 2048 + 1024 kunci). Untuk informasi lebih lanjut tentang kunci utama, lihat Kunci primer.</p> <p>Jenis: Memetakan HashKeyElement , atau HashKeyElement dan RangeKeyElement untuk kunci primer komposit.</p>
ProvisionedThroughput	<p>Pengaturan throughput saat ini untuk tabel yang ditentukan, termasuk nilai untuk LastIncreaseDateTime (jika berlaku), LastDecreaseDateTime (jika berlaku).</p>

Nama	Deskripsi
	Jenis: Susunan
TableName	Nama tabel yang diperbarui.
	Jenis: String
TableStatus	Keadaan tabel saat ini (CREATING, ACTIVE, DELETING atau UPDATING), yang seharusnya UPDATING.
	Gunakan operasi DescribeTables untuk memeriksa status tabel.
	Jenis: String

Kesalahan Khusus

Kesalahan	Deskripsi
ResourceNotFoundException	Tabel yang ditentukan tidak ditemukan.
ResourceInUseException	Tabel ini tidak berada dalam status ACTIVE.

Contoh

Permintaan sampel

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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}
}
```

Contoh respons

```
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"}
}
```

Tindakan terkait

- [CreateTable](#)
- [DescribeTables](#)
- [DeleteTable](#)

AWS Contoh SDK for Java 1.x

Bagian ini berisi kode contoh aplikasi DAX menggunakan SDK for Java 1.x.

Topik

- [Menggunakan DAX dengan AWS SDK for Java 1.x](#)
- [Memodifikasi SDK for Java 1.x yang ada agar menggunakan DAX](#)
- [Menanyakan indeks sekunder global dengan SDK for Java 1.x](#)

Menggunakan DAX dengan AWS SDK for Java 1.x

Ikuti prosedur ini untuk menjalankan sampel Java untuk Amazon DynamoDB Accelerator (DAX) pada instans Amazon EC2 Anda.

Note

Petunjuk ini ditujukan untuk aplikasi yang menggunakan AWS SDK for Java 1.x. Untuk aplikasi yang menggunakan AWS SDK for Java 2.x, lihat [Java dan DAX](#).

Cara menjalankan sampel Java untuk DAX

1. Instal Java Development Kit (JDK).

```
sudo yum install -y java-devel
```

2. Unduh AWS SDK for Java (file .zip), kemudian ekstrak.

```
wget http://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip  
unzip aws-java-sdk.zip
```

3. Unduh versi terbaru dari klien DAX Java (file .jar).

```
wget http://dax-sdk.s3-website-us-west-2.amazonaws.com/java/DaxJavaClient-latest.jar
```

Note

Klien untuk DAX SDK for Java tersedia di Apache Maven. Untuk informasi selengkapnya, lihat [Menggunakan klien sebagai dependensi Apache Maven](#).

4. Atur variabel CLASSPATH Anda. Dalam contoh ini, ganti *sdkVersion* dengan nomor versi AWS SDK for Java sebenarnya (misalnya, 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. Unduh sampel kode sumber program (file .zip).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/TryDax.zip
```

Setelah unduhan selesai, ekstrak file sumber.

```
unzip TryDax.zip
```

6. Buka direktori kode Java dan kompilasi kode sebagai berikut.

```
cd TryDax/java/  
javac TryDax*.java
```

7. Jalankan program.

```
java TryDax
```

Anda akan melihat output seperti yang berikut ini.

```
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.
```

Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian GetItem, Query, dan Scan.

8. Pada langkah sebelumnya, Anda menjalankan program terhadap titik akhir DynamoDB. Sekarang jalankan program lagi, tetapi kali ini, operasi GetItem, Query dan Scan diproses oleh kluster DAX Anda.

Untuk menentukan titik akhir kluster DAX Anda, pilih salah satu dari berikut ini:

- Menggunakan konsol DynamoDB — Pilih kluster DAX Anda. Titik akhir kluster ditampilkan pada konsol, seperti dalam contoh berikut.

```
dax://my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com
```

- Menggunakan AWS CLI — Masukkan perintah berikut.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

Titik akhir kluster ditampilkan pada output, seperti dalam contoh berikut.

```
{
  "Address": "my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com",
  "Port": 8111,
  "URL": "dax://my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

Sekarang jalankan program lagi, tetapi kali ini tentukan titik akhir kluster sebagai parameter baris perintah.

```
java TryDax dax://my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com
```

Periksa sisa output dan perhatikan informasi waktu. Waktu berlalu untuk `GetItem`, `Query`, dan `Scan` harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

Untuk informasi selengkapnya tentang program ini, lihat bagian berikut:

- [TryDax.jawa](#)
- [TryDaxHelper.jawa](#)
- [TryDaxTests.jawa](#)

Menggunakan klien sebagai dependensi Apache Maven

Ikuti langkah-langkah berikut untuk menggunakan klien untuk DAX SDK for Java dalam aplikasi Anda sebagai dependensi.

Cara menggunakan klien sebagai dependensi Maven

1. Unduh dan instal Apache Maven. Untuk informasi selengkapnya, lihat [Mengunduh Apache Maven](#) dan [Menginstal Apache Maven](#).
2. Tambahkan dependensi Maven klien ke file Project Object Model (POM) aplikasi Anda. Dalam contoh ini, ganti `x.x.x.x` dengan nomor versi klien sebenarnya (misalnya, `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

File `TryDax.java` berisi metode `main`. Jika Anda menjalankan program tanpa parameter baris perintah, klien Amazon DynamoDB akan dibuat dan digunakan untuk semua operasi API. Jika Anda menentukan titik akhir kluster DynamoDB Accelerator (DAX) pada baris perintah, program juga membuat klien DAX dan menggunakannya untuk operasi `GetItem`, `Query`, dan `Scan`.

Anda dapat memodifikasi program dengan beberapa cara:

- Menggunakan klien DAX, bukan klien DynamoDB. Untuk informasi selengkapnya, lihat [Java dan DAX](#).
- Memilih nama yang berbeda untuk tabel pengujian.
- Memodifikasi jumlah item yang ditulis dengan mengubah parameter `helper.writeData`. Parameter kedua adalah bilangan kunci partisi dan parameter ketiga adalah kunci urutan. Secara default, program menggunakan 1-10 untuk nilai kunci partisi dan 1-10 untuk nilai kunci urutan, dengan total 100 item yang ditulis ke tabel. Untuk informasi selengkapnya, lihat [TryDaxHelper.java](#).
- Memodifikasi jumlah pengujian `GetItem`, `Query`, dan `Scan`, serta memodifikasi parameternya.
- Mengomentari baris yang berisi `helper.createTable` dan `helper.deleteTable` (jika Anda tidak ingin membuat dan menghapus tabel setiap kali menjalankan program).

Note

Untuk menjalankan program ini, Anda dapat mengatur Maven agar menggunakan klien untuk DAX SDK for Java dan AWS SDK for Java sebagai dependensi. Untuk informasi selengkapnya, lihat [Menggunakan klien sebagai dependensi Apache Maven](#).

Selain itu, Anda juga dapat mengunduh dan menyertakan klien DAX Java dan AWS SDK for Java di classpath Anda. Lihat [Java dan DAX](#) untuk contoh pengaturan variabel `CLASSPATH`.

```
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

File `TryDaxHelper.java` berisi metode utilitas.

Metode `getDynamoDBClient` dan `getDaxClient` menyediakan klien untuk Amazon DynamoDB dan DynamoDB Accelerator (DAX). Untuk operasi bidang kontrol (`CreateTable`, `DeleteTable`) dan operasi penulisan, program menggunakan klien DynamoDB. Jika Anda menentukan titik akhir kluster DAX, program utama membuat klien DAX untuk melakukan operasi baca (`GetItem`, `Query`, `Scan`).

Metode `TryDaxHelper` lainnya (`createTable`, `writeData`, `deleteTable`) digunakan untuk mengatur dan menghapus tabel DynamoDB dan datanya.

Anda dapat memodifikasi program dengan beberapa cara:

- Menggunakan pengaturan throughput lain yang disediakan untuk tabel.
- Memodifikasi ukuran setiap item yang ditulis (lihat variabel `stringSize` di metode `writeData`).
- Memodifikasi jumlah pengujian `GetItem`, `Query`, dan `Scan` serta parameternya.
- Mengomentari baris yang berisi `helper.CreateTable` dan `helper.DeleteTable` (jika Anda tidak ingin membuat dan menghapus tabel setiap kali menjalankan program).

Note

Untuk menjalankan program ini, Anda dapat mengatur Maven agar menggunakan klien untuk DAX SDK for Java dan AWS SDK for Java sebagai dependensi. Untuk informasi selengkapnya, lihat [Menggunakan klien sebagai dependensi Apache Maven](#).

Selain itu, Anda juga dapat mengunduh dan menyertakan klien DAX Java dan AWS SDK for Java di classpath Anda. Lihat [Java dan DAX](#) untuk contoh pengaturan variabel `CLASSPATH`.

```
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

File `TryDaxTests.java` berisi metode yang melakukan operasi pembacaan terhadap tabel pengujian di Amazon DynamoDB. Metode ini tidak peduli dengan cara data diakses (menggunakan klien DynamoDB atau klien DAX), sehingga tidak memerlukan modifikasi logika aplikasi.

Anda dapat memodifikasi program dengan beberapa cara:

- Memodifikasi metode `queryTest` agar menggunakan `KeyConditionExpression` yang berbeda.
- Menambahkan `ScanFilter` ke metode `scanTest` sehingga hanya beberapa item yang dikembalikan kepada Anda.

Note

Untuk menjalankan program ini, Anda dapat mengatur Maven agar menggunakan klien untuk DAX SDK for Java dan AWS SDK for Java sebagai dependensi. Untuk informasi selengkapnya, lihat [Menggunakan klien sebagai dependensi Apache Maven](#).

Selain itu, Anda juga dapat mengunduh dan menyertakan klien DAX Java dan AWS SDK for Java di classpath Anda. Lihat [Java dan DAX](#) untuk contoh pengaturan variabel CLASSPATH.

```
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));
    }
}
```

Memodifikasi SDK for Java 1.x yang ada agar menggunakan DAX

Jika sudah memiliki aplikasi Java yang menggunakan Amazon DynamoDB, Anda harus memodifikasinya agar dapat mengakses kluster DynamoDB Accelerator (DAX). Anda tidak harus menulis ulang seluruh aplikasi karena klien Java DAX mirip dengan klien tingkat rendah DynamoDB yang disertakan dalam AWS SDK for Java.

Note

Petunjuk ini ditujukan untuk aplikasi yang menggunakan AWS SDK for Java 1.x. Untuk aplikasi yang menggunakan AWS SDK for Java 2.x, lihat [Memodifikasi aplikasi yang ada untuk menggunakan DAX](#).

Anggaplah Anda memiliki tabel DynamoDB bernama Music. Kunci partisi untuk tabel ini adalah Artist, dan kunci urutannya adalah SongTitle. Program berikut membaca item secara langsung dari tabel Music.

```
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());
}
}
```

Untuk mengubah program, ganti klien DynamoDB dengan klien DAX.

```
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)
        ** ...
        */
    }
}
```

```
    */  
  }  
}
```

Menggunakan API dokumen DynamoDB

AWS SDK for Java menyediakan antarmuka dokumen untuk DynamoDB. API dokumen bertindak sebagai pembungkus klien tingkat rendah DynamoDB. Untuk informasi selengkapnya, lihat [Antarmuka dokumen](#).

Antarmuka dokumen juga dapat digunakan dengan klien DAX tingkat rendah, seperti yang ditunjukkan dalam contoh berikut.

```
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.getItemOutcome(  
                "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());
    }
}
}
```

Klien asinkron DAX

`AmazonDaxClient` bersifat sinkron. Untuk operasi API DAX yang berjalan lama, seperti Scan dari tabel besar, operasi ini dapat memblokir eksekusi program hingga operasi selesai. Jika program perlu melakukan pekerjaan lain saat operasi API DAX sedang berlangsung, Anda dapat menggunakan `ClusterDaxAsyncClient` sebagai gantinya.

Program berikut menunjukkan cara menggunakan `ClusterDaxAsyncClient`, bersama dengan Java Future, untuk menerapkan solusi non-blocking.

```
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();
```



```
}  
}
```

Menanyakan indeks sekunder global dengan SDK for Java 1.x

Anda dapat menggunakan Amazon DynamoDB Accelerator (DAX) untuk meng-kueri [global secondary indexes](#) menggunakan [antarmuka program](#) DynamoDB.

Contoh berikut mendemonstrasikan cara menggunakan DAX untuk queryCreateDateIndex indeks sekunder global yang dibuat di [Contoh: Indeks sekunder global menggunakan AWS SDK for Java API dokumen](#).

Kelas `DaxClient` memberi contoh objek klien yang diperlukan untuk berinteraksi dengan antarmuka pemrograman DynamoDB.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;  
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;  
import com.amazonaws.services.dynamodbv2.document.DynamoDB;  
import com.amazonaws.util.EC2MetadataUtils;  
import com.amazonaws.services.dynamodbv2.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);
}
}
```

Anda dapat meng-kueri indeks sekunder global dengan cara berikut:

- Gunakan metode `queryIndex` pada kelas `QueryIndexDax` yang didefinisikan dalam contoh berikut. `QueryIndexDax` dijadikan sebagai parameter objek klien yang dikembalikan oleh metode `getDaxDocClient` pada kelas `DaxClient`.
- Jika Anda menggunakan [antarmuka persitensi objek](#), gunakan metode `queryIndexMapper` pada kelas `QueryIndexDax` yang didefinisikan dalam contoh berikut. `queryIndexMapper` dijadikan sebagai parameter objek klien yang dikembalikan oleh metode yang didefinisikan di `getDaxMapperClient` pada kelas `DaxClient`.

```
import java.util.Iterator;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import java.util.List;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBQueryExpression;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import java.util.HashMap;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.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());
}
}
```

Definisi kelas di bawah ini merupakan tabel Masalah dan digunakan dalam metode `queryIndexMapper`.

```
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIndexHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIndexRangeKey;
import com.amazonaws.services.dynamodbv2.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;
    }
}
```

Riwayat dokumen untuk DynamoDB

Tabel berikut menjelaskan perubahan penting dalam setiap rilis Panduan Developer DynamoDB mulai tanggal 3 Juli 2018 dan seterusnya. Untuk notifikasi tentang pembaruan dokumentasi ini, Anda dapat berlangganan ke umpan RSS (di sudut kiri atas halaman ini).

Perubahan	Deskripsi	Tanggal
Kebijakan berbasis sumber daya untuk sumber daya Amazon DynamoDB	DynamoDB sekarang mendukung kebijakan berbasis sumber daya untuk tabel, indeks, dan aliran. Kebijakan berbasis sumber daya memungkinkan Anda menentukan izin akses dengan menentukan siapa yang memiliki akses ke setiap sumber daya, dan tindakan yang diizinkan untuk dilakukan pada setiap sumber daya. Untuk informasi selengkapnya, lihat Menggunakan kebijakan berbasis sumber daya untuk DynamoDB.	Maret 20, 2024
Pembaruan kebijakan terkelola DynamoDB	Menambahkan izin baru <code>dynamodb:GetResourcePolicy</code> ke kebijakan <code>AmazonDynamoDBReadOnlyAccess</code> terkelola. Izin ini menyediakan akses ke kebijakan berbasis sumber daya baca yang dilampirkan ke sumber daya DynamoDB. Untuk informasi selengkapnya, lihat kebijakan AWS	Maret 20, 2024

[terkelola: AmazonDynamo DB
ReadOnlyAccess.](#)

[AWS PrivateLink untuk
Amazon DynamoDB](#)

Amazon DynamoDB sekarang mendukung. AWS PrivateLink Dengan AWS PrivateLink, Anda dapat menyederhanakan konektivitas jaringan pribadi antara virtual private cloud (VPC), DynamoDB, dan pusat data lokal Anda menggunakan titik akhir VPC antarmuka dan alamat IP pribadi. Untuk informasi selengkapnya, lihat [AWS PrivateLink DynamoDB](#).

Maret 19, 2024

[Pemrograman dengan
JavaScript panduan](#)

Amazon DynamoDB menyajikan panduan pemrograman untuk. AWS SDK for JavaScript Pelajari tentang, lapisan abstraksi AWS SDK for JavaScript, mengonfigurasi koneksi, menangani kesalahan, mendefinisikan kebijakan coba lagi, mengelola keep-alive, dan banyak lagi. Untuk informasi lebih lanjut, lihat [Pemrograman dengan JavaScript](#).

Maret 6, 2024

[Pemrograman dengan AWS SDK for Java 2.x panduan](#)

Membuat panduan pemrograman baru yang membahas secara mendalam tentang antarmuka tingkat tinggi, tingkat rendah, dan dokumen, klien HTTP dan konfigurasi, penanganan kesalahan, dan alamat pengaturan konfigurasi paling umum yang harus Anda pertimbangkan saat menggunakan SDK for Java 2.x. Untuk informasi selengkapnya, lihat [Memprogram Amazon AWS SDK for Java 2.x DynamoDB](#) dengan.

Maret 5, 2024

[Kloning tabel dengan NoSQL Workbench](#)

Izinkan pengembang menggunakan NoSQL Workbench untuk menyalin atau mengkloning tabel antara lingkungan pengembangan dan wilayah (DynamoDB Local dan DynamoDB web). Untuk informasi selengkapnya, lihat [Mengkloning tabel dengan NoSQL Workbench](#).

Februari 26, 2024

[Pemrograman dengan panduan Python](#)

Membuat panduan baru yang membahas secara mendalam tentang pustaka tingkat tinggi dan tingkat rendah dan membahas pengaturan konfigurasi paling umum yang harus dipertimbangkan saat menggunakan Python SDK. Untuk informasi selengkapnya, lihat [Pemrograman dengan Python](#).

Januari 5, 2024

[Tulis ulang topik Time to Live \(TTL\)](#)

Sepenuhnya menulis ulang bagian TTL dari panduan ini. Panduan baru ini membantu Anda memulai TTL dengan memberikan cuplikan ready-to-use kode di sepanjang jalan. Cuplikan kode saat ini yang disediakan dalam Python dan Javascript. Untuk informasi lebih lanjut, lihat [TTL](#).

Desember 20, 2023

[Praktik Terbaik untuk Memahami Laporan AWS Penagihan dan Penggunaan Anda](#)

Menambahkan bagian baru yang menjelaskan berbagai jenis penggunaan dan biaya untuk jenis penggunaan tersebut di DynamoDB. Untuk informasi selengkapnya, lihat [Laporan penagihan dan penggunaan](#).

15 Desember 2023

[Integrasi Amazon DynamoDB Zero-ETL dengan Amazon Service OpenSearch](#)

Amazon DynamoDB sekarang mendukung integrasi nol-ETL dengan OpenSearch Amazon Service, yang memungkinkan Anda melakukan pencarian pada data DynamoDB Anda dengan secara otomatis mereplikasi dan mengubahnya tanpa kode atau infrastruktur khusus. Untuk informasi selengkapnya, lihat [Integrasi DynamoDB Zero-ETL dengan Amazon Service](#). OpenSearch

28 November 2023

[Migrasi ke DynamoDB dari database relasional](#)

Membuat [panduan migrasi](#) untuk membantu pengguna memahami dan cara bermigrasi ke DynamoDB dari database relasional.

27 November 2023

[Menghasilkan data sampel dengan NoSQL Workbench](#)

NoSQL Workbench untuk Amazon DynamoDB sekarang mendukung pembuatan model data langsung dari [sampel templat model data](#) untuk membantu Anda merancang skema data untuk beban kerja Anda. Anda dapat menggunakan fitur ini untuk membiasakan diri dengan praktik terbaik pemodelan data NoSQL saat membangun aplikasi di DynamoDB.

28 September 2023

[Ekspor Tambahan ke S3](#)

Anda sekarang dapat mengekspor data yang dimasukkan, diperbarui, atau dihapus, sedikit demi sedikit. Dengan [ekspor tambahan](#), Anda dapat mengekspor data yang diubah mulai dari beberapa megabyte hingga terabyte dengan beberapa klik di AWS Management Console, panggilan API, atau Antarmuka Baris Perintah. AWS

26 September 2023

[Pemodelan data untuk DynamoDB](#)

Anda sekarang dapat mempelajari lebih lanjut tentang [pemodelan data](#) dengan contoh DynamoDB yang berfokus pada kasus penggunaan tertentu, pola aksesnya, step-by-step dan panduan dalam mewujudkan pola akses tersebut.

14 Juli 2023

[Bagian pemecahan masalah](#)

Sekarang Anda dapat menemukan [konten pemecahan masalah](#) untuk masalah latensi dan throttling yang mungkin terjadi di tabel DynamoDB Anda.

13 Maret 2023

[Perlindungan penghapusan untuk Amazon DynamoDB](#)

Perlindungan penghapusan sekarang tersedia untuk tabel Amazon DynamoDB di semua Wilayah AWS . DynamoDB kini memungkinkan Anda melindungi tabel Anda dari penghapusan yang tidak disengaja saat melakukan operasi manajemen tabel reguler.

8 Maret 2023

[AWS CloudFormation dukungan untuk KDSD dalam tabel global](#)

Amazon Kinesis Data Streams untuk DynamoDB AWS CloudFormation sekarang mendukung tabel global DynamoDB, yang berarti Anda dapat mengaktifkan streaming ke Amazon Kinesis Data Streams di tabel global DynamoDB Anda dengan templat. CloudFormation

15 Februari 2023

[DynamoDB lokal mendukung 100 tindakan per transaksi](#)

Anda sekarang dapat melakukan hingga 100 tindakan dalam satu transaksi di DynamoDB lokal.

9 Februari 2023

[Menggunakan DynamoDB Well-Architected Lens untuk mengoptimalkan beban kerja DynamoDB Anda](#)

Anda sekarang dapat menggunakan [DynamoDB Well-Architected Lens](#), kumpulan prinsip dan panduan desain yang dapat Anda gunakan untuk merancang beban kerja DynamoDB yang dirancang dengan baik.

3 Februari 2023

Ketersediaan PartiQL GovCloud	PartiQL—Bahasa kueri yang kompatibel dengan SQL untuk Amazon DynamoDB sekarang didukung di (AS-Timur) dan (AS-Barat). AWS GovCloud AWS GovCloud	21 Desember 2022
Rangkaian instalasi tunggal untuk NoSQL Workbench dan DynamoDB lokal	NoSQL Workbench untuk DynamoDB sekarang menyertakan proses instalasi DynamoDB lokal terpandu untuk menyederhanakan pengaturan lingkungan pengembangan DynamoDB lokal Anda.	6 Desember 2022
Impor massal dari S3	Amazon DynamoDB sekarang memudahkan Anda untuk bermigrasi dan memuat data ke tabel DynamoDB baru dengan mendukung impor data massal dari Amazon S3 .	18 Agustus 2022
Integrasi yang ditingkatkan dengan Kuota Layanan	Kuota Layanan sekarang memungkinkan Anda untuk mengelola kuota tabel dan akun Anda secara proaktif. Anda dapat melihat nilai saat ini, mengatur alarm ketika penggunaan kuota melebihi ambang batas yang dapat dikonfigurasi, dan banyak lagi.	15 Juni 2022

[NoSQL Workbench menambahkan dukungan GSI dan tabel](#)

Anda sekarang dapat menggunakan NoSQL Workbench untuk tabel dan operasi [bidang kontrol indeks sekunder global \(GSI\)](#) seperti CreateTable,, dan. UpdateTable DeleteTable

2 Juni 2022

[Kelas tabel Standard-infrequent Access kini tersedia di Tiongkok](#)

Kelas tabel Amazon DynamoDB Standard-Infrequent Access tersedia di Wilayah Tiongkok. Kurangi [biaya DynamoDB Anda hingga 60 persen](#), dengan menggunakan kelas tabel baru ini untuk tabel yang menyimpan data yang jarang diakses.

18 April 2022

[Peningkatan kuota layanan default dan operasi manajemen tabel](#)

[DynamoDB meningkatkan kuota default untuk jumlah tabel per akun dan Wilayah](#) dari 256 menjadi 2.500 tabel, dan meningkatkan jumlah operasi manajemen tabel bersamaan dari 50 menjadi 500.

9 Maret 2022

[Pembatasan item opsional dengan PartiQL untuk DynamoDB](#)

DynamoDB dapat [membatasi jumlah item yang diproses di PartiQL](#) untuk operasi DynamoDB sebagai parameter opsional pada setiap permintaan.

8 Maret 2022

[AWS Backup integrasi tersedia di Wilayah China \(Beijing dan Ningxia\)](#)

[AWS Backup](#) sekarang terintegrasi dengan DynamoDB di Wilayah Tiongkok (Beijing dan Ningxia). Anda dapat memenuhi persyaratan kepatuhan dan kelangsungan bisnis dengan lebih mudah melalui fitur pencadangan yang disempurnakan AWS Backup, seperti pencadangan lintas akun dan lintas wilayah.

26 Januari 2022

[Informasi kapasitas throughput melalui panggilan API PartiQL](#)

DynamoDB dapat mengembalikan kapasitas throughput yang digunakan oleh panggilan [API PartiQL](#) untuk membantu Anda mengoptimalkan kueri dan biaya throughput.

18 Januari 2022

[AWS Backup integrasi](#)

DynamoDB kini membantu Anda memenuhi persyaratan kepatuhan dan kelangsungan bisnis dengan lebih mudah melalui fitur pencadangan yang ditingkatkan di [AWS Backup](#), seperti pencadangan lintas akun dan lintas Wilayah.

24 November 2021

[Set data impor/ekspor NoSQL Workbench dalam CSV](#)

[NoSQL Workbench untuk Amazon DynamoDB](#) sekarang memungkinkan Anda untuk mengimpor dan secara otomatis mengisi data sampel untuk membantu membangun dan memvisualisasikan model data Anda.

11 Oktober 2021

[Filter dan ambil aktivitas bidang data Amazon DynamoDB Streams dengan AWS CloudTrail](#)

Amazon DynamoDB kini memberi Anda kontrol pencatatan audit yang lebih mendetail dengan memungkinkan Anda [memfilter aktivitas API bidang data Streams di AWS CloudTrail](#).

22 September 2021

[Pembaruan konsol](#)

[Konsol DynamoDB](#) kini menjadi konsol default Anda untuk membantu mengelola data dengan lebih mudah, menyederhanakan tugas umum, dan memberi Anda akses lebih cepat ke sumber daya dan fitur.

25 Agustus 2021

[DAX SDK For Java 2.x sekarang tersedia](#)

[DynamoDB Accelerator \(DAX\) SDK for Java 2.x](#) sekarang tersedia dan kompatibel dengan SDK for Java 2.x. AWS Anda bisa mendapatkan keuntungan dari fitur-fitur terbaru, termasuk non-blocking I/O.

29 Juli 2021

Pembaruan fitur NoSQL Workbench mencakup operasi bidang kontrol	NoSQL Workbench untuk Amazon DynamoDB sekarang membantu Anda menjalankan operasi yang sering dijalankan dengan lebih mudah untuk mengubah dan mengakses data tabel.	28 Juli 2021
Tabel global DynamoDB kini tersedia di Wilayah Asia Pasifik	Tabel global DynamoDB kini tersedia di Wilayah Asia Pasifik (Osaka). Mereplikasi tabel DynamoDB Anda secara otomatis di 22 Wilayah AWS pilihan Anda.	28 Juli 2021
DAX kini tersedia di Tiongkok	DynamoDB Accelerator (DAX) kini tersedia di Wilayah Tiongkok (Beijing), yang dioperasikan oleh Sinnet.	28 Juli 2021
Enkripsi bergerak DAX	DynamoDB Accelerator (DAX) sekarang mendukung enkripsi bergerak untuk data antara aplikasi Anda dan klaster DAX, dan di antara simpul dalam klaster DAX.	24 Juli 2021
CloudFormation dan CloudTrail integrasi	Integrasi dengan AWS CloudFormation dan peningkatan keamanan dengan CloudFormation data-plane logging.	18 Juni 2021

[CloudFormation sekarang didukung untuk tabel global](#)

Tabel [global Amazon DynamoDB](#) sekarang [AWS CloudFormation](#) mendukung, yang berarti Anda dapat membuat tabel global dan mengelola pengaturannya dengan templat. CloudFormation

14 Mei 2021

[Dukungan Amazon DynamoDB lokal untuk Java 2.x](#)

Anda sekarang dapat menggunakan [AWS SDK for Java 2.x](#) dengan [DynamoDB lokal](#), versi Amazon DynamoDB yang dapat diunduh. Dengan DynamoDB lokal, Anda dapat mengembangkan dan menguji aplikasi menggunakan versi DynamoDB yang berjalan di lingkungan pengembangan lokal Anda tanpa menimbulkan biaya tambahan.

3 Mei 2021

[NoSQL Workbench sekarang mendukung AWS CloudFormation](#)

[NoSQL Workbench untuk Amazon DynamoDB AWS CloudFormation](#) sekarang mendukung, sehingga Anda dapat mengelola dan memodifikasi model data DynamoDB dengan template. CloudFormation Selain itu, Anda sekarang dapat mengonfigurasi pengaturan kapasitas tabel di NoSQL Workbench.

22 April 2021

[DynamoDB dan sekarang integrasi fitur AWS Amplify](#)

[AWS Amplify](#) sekarang mengorkestrasi beberapa pembaruan indeks sekunder global DynamoDB dalam satu deployment.

20 April 2021

[AWS CloudTrail untuk mencatat API bidang data Amazon DynamoDB Streams](#)

Anda sekarang dapat menggunakan [AWS CloudTrail](#) untuk mencatat aktivitas API bidang data Amazon [DynamoDB Streams](#), dan memantau dan menyelidiki perubahan tingkat item di tabel DynamoDB Anda.

20 April 2021

[Amazon Kinesis Data Streams untuk Amazon DynamoDB sekarang mendukung AWS CloudFormation](#)

[Amazon Kinesis Data Streams untuk Amazon DynamoDB](#) AWS CloudFormation sekarang mendukung, yang berarti Anda dapat mengaktifkan streaming ke aliran data Amazon Kinesis di tabel DynamoDB Anda dengan templat. CloudFormation Dengan melakukan streaming perubahan data DynamoDB ke aliran data Kinesis, Anda dapat membangun aplikasi streaming tingkat lanjut dengan layanan Amazon Kinesis.

12 April 2021

[Amazon Keyspaces sekarang menawarkan titik akhir yang sesuai dengan FIPS 140-2](#)

[Amazon Keyspaces \(untuk Apache Cassandra\)](#) sekarang menawarkan titik akhir yang sesuai dengan Federal Information Processing Standards (FIPS) 140-2 untuk membantu Anda menjalankan beban kerja yang teregulasi dengan lebih mudah. FIPS 140-2 adalah standar pemerintah AS dan Kanada yang menentukan persyaratan keamanan untuk modul kriptografi yang melindungi informasi sensitif.

8 April 2021

[Instans Amazon EC2 T3 untuk DAX](#)

DAX sekarang mendukung [jenis instans Amazon EC2 T3](#), yang memberikan tingkat dasar performa CPU dengan kemampuan untuk meningkatkan di atas tingkat dasar saat diperlukan.

15 Februari 2021

[Dukungan NoSQL Workbench untuk Amazon DynamoDB untuk PartiQL](#)

Anda sekarang dapat menggunakan [NoSQL Workbench untuk DynamoDB](#) untuk membuat pernyataan [PartiQL](#) untuk DynamoDB.

4 Desember 2020

[PartiQL untuk DynamoDB](#)

Sekarang Anda dapat menggunakan [PartiQL untuk DynamoDB](#)—bahasa kueri yang kompatibel dengan SQL—untuk berinteraksi dengan tabel DynamoDB dan menjalankan kueri ad hoc dengan menggunakan API,, dan [DynamoDB untuk PartiQL](#).
AWS Management Console
AWS Command Line Interface

23 November 2020

[Amazon Kinesis Data Streams untuk Amazon DynamoDB](#)

Anda sekarang dapat menggunakan [Amazon Kinesis Data Streams untuk Amazon DynamoDB](#) dengan tabel DynamoDB Anda untuk menangkap perubahan tingkat item dan mereplikasinya ke aliran data Kinesis.

23 November 2020

[Ekspor tabel DynamoDB](#)

Sekarang Anda dapat [mengekspor tabel DynamoDB Anda ke Amazon S3](#), memungkinkan Anda melakukan analitik dan kueri kompleks pada data Anda dengan layanan seperti Athena,, dan Lake Formation.
AWS Glue

9 November 2020

Dukungan untuk nilai kosong	DynamoDB sekarang mendukung nilai kosong untuk String non kunci dan atribut Biner di tabel DynamoDB. Dukungan nilai kosong memberi Anda fleksibilitas lebih dalam menggunakan atribut untuk serangkaian kasus penggunaan yang lebih luas tanpa harus mengubah atribut tersebut sebelum mengirimkannya ke DynamoDB. Jenis Data set, Daftar, dan Peta juga mendukung String kosong dan nilai-nilai Biner.	18 Mei 2020
Dukungan NoSQL Workbench untuk Amazon DynamoDB untuk Linux	NoSQL Workbench untuk Amazon DynamoDB sekarang didukung di Linux- Ubuntu, Fedora, and Debian .	4 Mei 2020
CloudWatch Wawasan Kontributor untuk DynamoDB - GA	CloudWatchWawasan Kontributor untuk DynamoDB umumnya tersedia . CloudWatchContributor Insights for DynamoDB adalah alat diagnostik yang memberikan tampilan at-a-glance tren lalu lintas tabel DynamoDB Anda dan membantu Anda mengidentifikasi kunci tabel yang paling sering diakses (juga dikenal sebagai tombol pintas).	2 April 2020

[Meningkatkan tabel global](#)

Tabel global Anda sekarang dapat diperbarui dari versi 2017.11.29 ke [tabel global versi terbaru \(2019.11.21\)](#), dengan beberapa klik di Konsol DynamoDB. Dengan memutakhirkan versi tabel global Anda, Anda dapat meningkatkan ketersediaan tabel DynamoDB Anda dengan mudah dengan memperluas tabel yang ada ke Wilayah AWS tambahan, tanpa perlu membangun kembali tabel.

16 Maret 2020

[NoSQL Workbench untuk Amazon DynamoDB – GA](#)

[NoSQL Workbench untuk Amazon DynamoDB](#) tersedia secara umum. Gunakan NoSQL Workbench untuk merancang, membuat, mengkueri, dan mengelola tabel DynamoDB.

2 Maret 2020

[Metrik kluster cache DAX](#)

Dukungan DAX untuk [CloudWatch metrik](#) baru, yang memungkinkan Anda untuk lebih memahami kinerja cluster DAX Anda.

6 Februari 2020

[CloudWatch Wawasan Kontributor untuk DynamoDB - Pratinjau](#)

[CloudWatchContributor Insights for DynamoDB](#) adalah alat diagnostik yang memberikan tampilan at-a-glance tren lalu lintas tabel DynamoDB Anda dan membantu Anda mengidentifikasi kunci tabel yang paling sering diakses (juga dikenal sebagai tombol pintas).

26 November 2019

[Dukungan kapasitas adaptif untuk beban kerja yang tidak seimbang](#)

Kapasitas adaptif Amazon DynamoDB kini [menangani](#) beban kerja yang tidak seimbang dengan lebih baik, dengan mengisolasi item yang sering diakses secara otomatis. Jika aplikasi Anda mendorong lalu lintas tinggi secara tidak proporsional ke satu atau beberapa item, DynamoDB akan menyeimbangkan kembali partisi Anda sehingga item yang sering diakses tidak berada di partisi yang sama.

26 November 2019

[Dukungan untuk kunci yang dikelola pelanggan](#)

DynamoDB sekarang [mendukung kunci yang dikelola pelanggan](#), yang berarti Anda dapat memiliki kontrol penuh atas cara Anda mengenkripsi dan mengelola keamanan data DynamoDB Anda.

25 November 2019

[Dukungan NoSQL Workbench untuk DynamoDB lokal \(Versi yang Dapat Diunduh\)](#)

NoSQL Workbench kini mendukung koneksi ke [DynamoDB lokal \(Versi yang Dapat Diunduh\)](#) untuk merancang, membuat, mengkueri, dan mengelola tabel DynamoDB.

8 November 2019

[NoSQL Workbench - Pratinjau](#)

Ini adalah rilis awal NoSQL Workbench untuk DynamoDB. Gunakan NoSQL Workbench untuk merancang, membuat, mengkueri, dan mengelola tabel DynamoDB. Untuk informasi selengkapnya, lihat [NoSQL Workbench untuk Amazon DynamoDB \(Pratinjau\)](#).

16 September 2019

[DAX menambahkan dukungan untuk operasi transaksi onal menggunakan Python dan .NET](#)

DAX mendukung API TransactWriteItems dan TransactGetItems untuk aplikasi yang ditulis dalam Go, Java, .NET, Node.js, dan Python. Untuk informasi selengkapnya, lihat [Akselerasi Dalam Memori dengan DAX](#).

14 Februari 2019

[Pembaruan Amazon DynamoDB lokal \(Versi yang Dapat Diunduh\)](#)

DynamoDB lokal (Versi yang Dapat Diunduh) sekarang mendukung API transaksional, kapasitas baca/tulis sesuai permintaan, pelaporan kapasitas untuk operasi baca dan tulis, dan 20 indeks sekunder global. Untuk informasi selengkapnya, lihat [Perbedaan antara DynamoDB yang Dapat Diunduh dan Layanan Web DynamoDB](#).

4 Februari 2019

[Amazon DynamoDB Sesuai Permintaan](#)

DynamoDB sesuai permintaan adalah opsi penagihan fleksibel yang mampu melayani ribuan permintaan per detik tanpa perencanaan kapasitas. DynamoDB on-demand pay-per-request menawarkan harga untuk permintaan baca dan tulis sehingga Anda hanya membayar untuk apa yang Anda gunakan. Untuk informasi selengkapnya, lihat [Mode Kapasitas Baca/Tulis](#).

28 November 2018

[Amazon DynamoDB Transactions](#)

Transaksi DynamoDB membuat terkoordinasi all-or-nothing , perubahan pada beberapa item baik di dalam maupun di seluruh tabel, memberikan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) di DynamoDB. Untuk informasi selengkapnya, lihat [Amazon DynamoDB Transactions](#).

27 November 2018

[Amazon DynamoDB mengenkripsi semua data diam pelanggan](#)

Enkripsi diam DynamoDB menyediakan lapisan tambahan perlindungan data dengan mengamankan data Anda dalam tabel terenkripsi, termasuk kunci primer, indeks sekunder lokal dan global, aliran, tabel global, cadangan, dan klaster DAX setiap kali data disimpan dalam media yang tahan lama. Untuk informasi selengkapnya, lihat [Enkripsi Diam Amazon DynamoDB](#).

15 November 2018

[Menggunakan Amazon
DynamoDB Lokal Lebih
Mudah dengan Citra Docker
Baru](#)

Sekarang, penggunaan DynamoDB lokal, versi DynamoDB yang dapat diunduh, menjadi lebih mudah untuk membantu Anda mengembangkan dan menguji aplikasi DynamoDB Anda dengan menggunakan citra Docker DynamoDB lokal baru. Untuk informasi selengkapnya, lihat [DynamoDB \(Versi yang Dapat Diunduh\) dan Docker](#).

22 Agustus 2018

[DynamoDB Accelerator \(DAX\)
Menambahkan Dukungan
untuk Enkripsi Diam](#)

DynamoDB Accelerator (DAX) kini mendukung enkripsi diam untuk kluster DAX baru guna membantu Anda mempercepat pembacaan dari tabel Amazon DynamoDB dalam aplikasi yang sensitif terhadap keamanan yang tunduk pada persyaratan kepatuhan dan regulasi yang ketat. Untuk informasi selengkapnya, lihat [Enkripsi Diam DAX](#).

9 Agustus 2018

[point-in-time DynamoDB recovery \(PITR\) menambahkan dukungan untuk memulihkan tabel yang dihapus](#)

Jika Anda menghapus tabel dengan point-in-time pemulihan diaktifkan, cadangan sistem secara otomatis dibuat dan dipertahankan selama 35 hari (tanpa biaya tambahan). Untuk informasi selengkapnya, lihat [Sebelum Anda Mulai Menggunakan Pemulihan Titik Waktu](#).

7 Agustus 2018

[Pembaruan kini tersedia melalui RSS](#)

Sekarang, Anda dapat berlangganan [umpan RSS](#) (di sudut kiri atas halaman ini) untuk menerima notifikasi tentang pembaruan Panduan Developer Amazon DynamoDB.

3 Juli 2018

Pembaruan sebelumnya

Tabel berikut menjelaskan perubahan penting Panduan Developer DynamoDB sebelum 3 Juli 2018.

Perubahan	Deskripsi	Tanggal Diubah
Dukungan Go untuk DAX	Sekarang, Anda dapat mengaktifkan performa baca mikrodetik untuk tabel Amazon DynamoDB dalam aplikasi Anda yang ditulis dalam bahasa pemrograman Go dengan menggunakan DynamoDB Accelerator (DAX) SDK for Go baru. Untuk	26 Juni 2018

Perubahan	Deskripsi	Tanggal Diubah
	informasi selengkapnya, lihat DAX SDK for Go .	
DynamoDB mengumumkan SLA	DynamoDB telah merilis SLA ketersediaan publik. Untuk informasi selengkapnya, lihat Perjanjian Tingkat Layanan Amazon DynamoDB .	19 Juni 2018
Pencadangan berkelanjutan DynamoDB dan pemulihan titik waktu (PITR)	oint-in-time Pemulihan P membantu melindungi tabel Amazon DynamoDB Anda dari operasi tulis atau hapus yang tidak disengaja. Dengan pemulihan titik waktu, Anda tidak perlu khawatir dengan membuat, mengelola, atau menjadwalkan pencadangan sesuai permintaan. Misalnya, anggaplah skrip pengujian tidak sengaja menulis ke tabel DynamoDB produksi. Dengan point-in-time pemulihan, Anda dapat mengembalikan tabel itu ke titik waktu mana pun selama 35 hari terakhir. DynamoDB memelihara cadangan tambahan tabel Anda. Untuk informasi selengkapnya, lihat oint-in-time Pemulihan P untuk DynamoDB .	25 April 2018

Perubahan	Deskripsi	Tanggal Diubah
Enkripsi diam untuk DynamoDB	Enkripsi diam DynamoDB, tersedia untuk tabel DynamoDB baru, membantu mengamankan data aplikasi Anda dalam tabel Amazon DynamoDB menggunakan kunci enkripsi yang dikelola AWS yang disimpan di AWS Key Management Service. Untuk informasi selengkapnya, lihat Enkripsi DynamoDB saat diam .	8 Februari 2018
Pencadangan DynamoDB dan pemulihan	Pencadangan Sesuai Permintaan memungkinkan Anda membuat cadangan lengkap data tabel DynamoDB untuk pengarsipan data, membantu Anda memenuhi persyaratan regulasi perusahaan dan pemerintah. Anda dapat mencadangkan tabel mulai dari beberapa megabyte hingga ratusan terabyte data, tanpa memengaruhi performa dan ketersediaan aplikasi produksi Anda. Untuk informasi selengkapnya, lihat Menggunakan cadangan Sesuai Permintaan dan DynamoDB Permintaan .	29 November 2017

Perubahan	Deskripsi	Tanggal Diubah
Tabel Global DynamoDB	Tabel Global dibangun berdasarkan jejak global DynamoDB untuk memberi Anda basis data yang dikelola sepenuhnya, multi-wilayah, dan multi-aktif yang memberikan performa baca dan tulis yang cepat serta lokal, untuk aplikasi global berskala besar. Tabel Global mereplikasi tabel Amazon DynamoDB Anda secara otomatis di seluruh wilayah pilihan Anda. AWS Untuk informasi selengkapnya, lihat Tabel global - Replikasi multi-Wilayah untuk DynamoDB .	29 November 2017
Dukungan Node.js untuk DAX	Developer Node.js dapat memanfaatkan DynamoDB Accelerator (DAX), menggunakan klien DAX untuk Node.js. Untuk informasi selengkapnya, lihat Akselerasi dalam memori dengan DynamoDB Accelerator (DAX) .	5 Oktober 2017

Perubahan	Deskripsi	Tanggal Diubah
Titik akhir VPC untuk DynamoDB	Titik akhir DynamoDB mengizinkan instans Amazon EC2 di Amazon VPC Anda untuk mengakses DynamoDB, tanpa paparan Internet publik. Lalu lintas jaringan antara VPC Anda dan DynamoDB tidak meninggalkan jaringan Amazon. Untuk informasi selengkapnya, lihat Menggunakan titik akhir Amazon VPC untuk mengakses DynamoDB .	16 Agustus 2017

Perubahan	Deskripsi	Tanggal Diubah
Penskalaan Otomatis untuk DynamoDB	<p>Penskalaan otomatis DynamoDB menghilangkan kebutuhan untuk menetapkan atau menyesuaikan secara manual pengaturan throughput yang ditetapkan. Sebaliknya, penskalaan otomatis DynamoDB menyesuaikan kapasitas baca dan tulis secara dinamis dalam merespons pola lalu lintas aktual. Ini memungkinkan tabel atau indeks sekunder global meningkatkan kapasitas baca dan tulis yang disediakan untuk menangani peningkatan lalu lintas tiba-tiba, tanpa throttling. Ketika beban kerja menurun, penskalaan otomatis DynamoDB mengurangi kapasitas yang disediakan. Untuk informasi selengkapnya, lihat Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB.</p>	14 Juni 2017

Perubahan	Deskripsi	Tanggal Diubah
DynamoDB Accelerator (DAX)	DynamoDB Accelerator (DAX) adalah cache dalam memori yang terkelola sepenuhnya dan sangat tersedia untuk DynamoDB yang memberikan peningkatan performa hingga 10x – dari milidetik hingga mikrodetik – bahkan pada jutaan permintaan per detik. Untuk informasi selengkapnya, lihat Akselerasi dalam memori dengan DynamoDB Accelerator (DAX) .	19 April 2017
DynamoDB sekarang mendukung kedaluwarsa item otomatis dengan Waktu untuk Tayang (TTL)	Waktu untuk Tayang (TTL) Amazon DynamoDB memungkinkan Anda untuk menghapus item yang kedaluwarsa dari tabel Anda secara otomatis, tanpa biaya tambahan. Untuk informasi selengkapnya, lihat Waktu untuk Hidup (TTL) .	27 Februari 2017
DynamoDB sekarang mendukung Tag Alokasi Biaya	Anda sekarang dapat menambahkan tag ke tabel Amazon DynamoDB Anda untuk meningkatkan penggunaan kategorisasi dan pelaporan biaya yang lebih mendetail. Untuk informasi selengkapnya, lihat Menambahkan tag dan label ke sumber daya .	19 Januari 2017

Perubahan	Deskripsi	Tanggal Diubah
API DescribeLimits DynamoDB baru	<p>DescribeLimits API mengembalikan batas kapasitas yang disediakan saat ini untuk AWS akun Anda di suatu wilayah, baik untuk wilayah secara keseluruhan maupun untuk satu tabel DynamoDB yang Anda buat di sana. Ini memungkinkan Anda untuk menentukan batas tingkat akun saat ini sehingga Anda dapat membandingkannya dengan kapasitas yang disediakan yang saat ini Anda gunakan, dan memiliki banyak waktu untuk mengajukan peningkatan sebelum mencapai batas. Untuk informasi selengkapnya, lihat Layanan, akun, dan tabel kuota di Amazon DynamoDB dan DescribeLimits di Referensi Amazon DynamoDB API.</p>	1 Maret 2016

Perubahan	Deskripsi	Tanggal Diubah
Pembaruan Konsol DynamoDB dan Terminologi Baru untuk Atribut Kunci Primer	<p>Konsol manajemen DynamoDB telah didesain ulang agar lebih intuitif dan mudah digunakan. Sebagai bagian dari pembaruan ini, kami memperkenalkan terminologi baru untuk atribut kunci primer:</p> <ul style="list-style-type: none">• Kunci Partisi—juga dikenal sebagai atribut hash.• Kunci Urutan juga dikenal sebagai atribut rentang. <p>Hanya nama yang berubah; fungsionalitasnya tetap sama.</p> <p>Ketika membuat tabel atau indeks sekunder, Anda dapat memilih kunci primer sederhana (hanya kunci partisi), atau kunci primer komposit (kunci partisi dan kunci urutan). Dokumentasi DynamoDB telah diperbarui untuk mencerminkan perubahan ini.</p>	12 November 2015

Perubahan	Deskripsi	Tanggal Diubah
Backend Penyimpanan Amazon DynamoDB untuk Titan	Backend Penyimpanan DynamoDB untuk Titan adalah backend penyimpanan untuk basis data grafik Titan yang diimplementasikan di atas Amazon DynamoDB. Saat menggunakan Backend Penyimpanan DynamoDB untuk Titan, data Anda mendapat manfaat dari perlindungan DynamoDB, yang berjalan di seluruh pusat data Amazon dengan ketersediaan tinggi. Plugin ini tersedia untuk Titan versi 0.4.4 (terutama untuk kompatibilitas dengan aplikasi yang ada) dan Titan versi 0.5.4 (direkomendasikan untuk aplikasi baru). Seperti backend penyimpanan lainnya untuk Titan, plugin ini mendukung tumpukan Tinkerpop (versi 2.4 dan 2.5), termasuk API Blueprints dan shell Gremlin. Untuk informasi selengkapnya, lihat Backend Penyimpanan Amazon DynamoDB untuk Titan .	20 Agustus 2015

Perubahan	Deskripsi	Tanggal Diubah
DynamoDB Streams, Replikasi Lintas Wilayah, dan Memindai dengan Bacaan yang Sangat Konsisten	<p>DynamoDB Streams menangkap urutan waktu modifikasi tingkat item di tabel DynamoDB mana pun, dan menyimpan informasi ini dalam log hingga 24 jam. Aplikasi dapat mengakses log ini dan melihat item data seperti yang ditampilkan sebelum dan setelah dimodifikasi, mendekati waktu nyata. Untuk informasi selengkapnya, lihat Ubah tangkapan data DynamoDB Streams dan Referensi API DynamoDB Streams.</p> <p>Replikasi lintas wilayah DynamoDB adalah solusi sisi klien untuk mempertahankan salinan identik tabel DynamoDB di berbagai wilayah, dalam waktu dekat. AWS Anda dapat menggunakan replikasi lintas wilayah untuk mencadangkan tabel DynamoDB, atau untuk menyediakan akses latensi rendah ke data tempat pengguna didistribusikan secara geografis.</p> <p>Operasi Scan DynamoDB menggunakan bacaan akhir konsisten, secara default.</p>	16 Juli 2015

Perubahan	Deskripsi	Tanggal Diubah
	<p>Anda dapat menggunakan bacaan sangat konsisten dengan mengatur parameter <code>ConsistentRead</code> ke <code>true</code>. Untuk informasi selengkapnya, lihat Konsistensi baca untuk kueri dan Pindai di Referensi API Amazon DynamoDB.</p>	
AWS CloudTrail dukungan untuk Amazon DynamoDB	<p>DynamoDB sekarang terintegrasi dengan CloudTrail. CloudTrail menangkap panggilan API yang dibuat dari konsol DynamoDB atau dari DynamoDB API dan melacaknya dalam file log. Untuk informasi selengkapnya, lihat Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail dan Panduan Pengguna AWS CloudTrail.</p>	28 Mei 2015

Perubahan	Deskripsi	Tanggal Diubah
Peningkatan dukungan untuk ekspresi Kueri	Rilis ini menambahkan parameter <code>KeyConditionExpression</code> baru untuk API <code>Query</code> . <code>Query</code> membaca item dari tabel atau indeks menggunakan nilai-nilai kunci primer. Parameter <code>KeyConditionExpression</code> adalah string yang mengidentifikasi nama-nama kunci primer, dan ketentuan yang akan diterapkan ke nilai-nilai kunci; <code>Query</code> hanya mengambil item-item yang memenuhi ekspresi. Sintaks <code>KeyConditionExpression</code> mirip dengan parameter ekspresi lainnya di DynamoDB, dan memungkinkan Anda untuk menentukan variabel pengganti untuk nama dan nilai dalam ekspresi. Untuk informasi selengkapnya, lihat Operasi kueri di DynamoDB .	27 April 2015

Perubahan	Deskripsi	Tanggal Diubah
Fungsi perbandingan baru untuk penulisan bersyarat	Dalam DynamoDB, parameter <code>ConditionExpression</code> menentukan apakah <code>PutItem</code> , <code>UpdateItem</code> , atau <code>DeleteItem</code> berhasil: Item tersebut ditulis hanya jika ketentuan bernilai true. Rilis ini menambahkan dua fungsi baru, <code>attribute_type</code> dan <code>size</code> , untuk digunakan dengan <code>ConditionExpression</code> . Fungsi-fungsi ini memungkinkan Anda untuk melakukan penulisan bersyarat berdasarkan jenis data atau ukuran atribut dalam sebuah tabel. Untuk informasi selengkapnya, lihat Ekspresi kondisi .	27 April 2015

Perubahan	Deskripsi	Tanggal Diubah
Memindai API untuk indeks sekunder	<p>Di DynamoDB, operasi Scan membaca semua item dalam sebuah tabel, menerapkan kriteria pemfilteran yang ditetapkan pengguna, dan mengembalikan item data yang dipilih ke aplikasi. Kemampuan yang sama ini sekarang tersedia untuk indeks sekunder juga. Untuk memindai indeks sekunder lokal atau indeks sekunder global, Anda menentukan nama indeks dan nama tabel induknya. Secara default, indeks Scan mengembalikan semua data dalam indeks; Anda dapat menggunakan ekspresi filter untuk mempersempit hasil yang dikembalikan ke aplikasi. Untuk informasi selengkapnya, lihat Bekerja dengan pemindaian di DynamoDB.</p>	10 Februari 2015

Perubahan	Deskripsi	Tanggal Diubah
Operasi online untuk indeks sekunder global	<p>Pengindeksan online memungkinkan Anda menambah atau menghapus indeks sekunder global pada tabel yang ada. Dengan pengindeksan online, Anda tidak perlu menentukan semua indeks tabel ketika Anda membuat tabel; sebaliknya, Anda dapat menambahkan indeks baru setiap saat. Demikian pula, jika memutuskan tidak memerlukan indeks lagi, Anda dapat menghapusnya kapan pun. Operasi pengindeksan online bersifat non-blocking, sehingga tabel tetap tersedia untuk aktivitas baca dan tulis saat indeks ditambahkan atau dihapus. Untuk informasi selengkapnya, lihat Mengelola Indeks Sekunder Global.</p>	27 Januari 2015

Perubahan	Deskripsi	Tanggal Diubah
Dukungan model dokumen dengan JSON	<p>DynamoDB memungkinkan Anda untuk menyimpan dan mengambil dokumen dengan dukungan penuh untuk model dokumen. Jenis data baru sepenuhnya kompatibel dengan standar JSON dan memungkinkan Anda menyusun elemen dokumen satu sama lain. Anda dapat menggunakan operator dereferensi jalur dokumen untuk membaca dan menulis masing-masing elemen, tanpa harus mengambil seluruh dokumen. Rilis ini juga memperkenalkan parameter ekspresi baru untuk menentukan proyeksi, kondisi, dan tindakan pembaruan ketika membaca atau menulis item data. Untuk mempelajari selengkapnya tentang dukungan model dokumen dengan JSON, lihat Tipe Data dan Menggunakan ekspresi di DynamoDB.</p>	7 Oktober 2014

Perubahan	Deskripsi	Tanggal Diubah
Penskalaan fleksibel	Untuk tabel dan indeks sekunder global, Anda dapat meningkatkan kapasitas throughput baca dan tulis yang disediakan dengan jumlah berapa pun, asalkan Anda tetap berada dalam batas per tabel dan per akun. Untuk informasi selengkapnya, lihat Layanan, akun, dan tabel kuota di Amazon DynamoDB .	7 Oktober 2014
Ukuran item yang lebih besar	Ukuran item maksimum di DynamoDB telah meningkat dari 64 KB menjadi 400 KB. Untuk informasi selengkapnya, lihat Layanan, akun, dan tabel kuota di Amazon DynamoDB .	7 Oktober 2014

Perubahan	Deskripsi	Tanggal Diubah
Ekspresi bersyarat yang ditingkatkan	<p>DynamoDB memperluas operator yang tersedia untuk ekspresi bersyarat, sehingga memberikan fleksibilitas tambahan untuk penempatan, pembaruan, dan penghapusan bersyarat. Operator baru yang tersedia memungkinkan Anda memeriksa apakah atribut ada atau tidak ada, lebih besar dari atau kurang dari nilai tertentu, berada di antara dua nilai, dimulai dengan karakter tertentu, dan banyak lagi. DynamoDB juga menyediakan operator OR opsional untuk mengevaluasi beberapa kondisi. Secara default, beberapa kondisi dalam sebuah ekspresi di-AND bersama-sama, sehingga ekspresi hanya bernilai true jika semua kondisinya benar. Jika Anda menentukan OR, ekspresinya adalah true jika satu atau beberapa kondisi adalah true. Untuk informasi selengkapnya, lihat Bekerja dengan item dan atribut.</p>	24 April 2014

Perubahan	Deskripsi	Tanggal Diubah
Filter kueri	<p>API Query DynamoDB mendukung opsi <code>QueryFilter</code> baru. Secara default, Query menemukan item yang cocok dengan nilai kunci partisi tertentu dan kondisi kunci urutan opsional. Query menerapkan ekspresi bersyarat ke atribut non kunci lain; jika filter Query ada, maka item yang tidak cocok dengan kondisi filter akan dibuang sebelum hasil Query dikembalikan ke aplikasi. Untuk informasi selengkapnya, lihat Operasi kueri di DynamoDB.</p>	24 April 2014

Perubahan	Deskripsi	Tanggal Diubah
Ekspor dan impor data menggunakan AWS Management Console	Konsol DynamoDB telah ditingkatkan untuk menyederhanakan ekspor dan impor data dalam tabel DynamoDB. Hanya dengan beberapa klik, Anda dapat mengatur AWS Data Pipeline untuk mengatur alur kerja, dan MapReduce cluster Amazon Elastic untuk menyalin data dari tabel DynamoDB ke bucket Amazon S3, atau sebaliknya. Anda dapat melakukan ekspor atau impor satu kali saja, atau mengatur pekerjaan ekspor harian. Anda bahkan dapat melakukan ekspor dan impor lintas wilayah, menyalin data DynamoDB dari tabel di satu AWS wilayah ke tabel di wilayah lain. AWS Untuk informasi selengkapnya, lihat Mengekspor dan mengimpor data DynamoDB menggunakan AWS Data Pipeline .	6 Maret 2014

Perubahan	Deskripsi	Tanggal Diubah
Menata ulang dokumentasi API tingkat yang lebih tinggi	<p>Informasi tentang API berikut sekarang lebih mudah ditemukan:</p> <ul style="list-style-type: none">• Java: DynamoDBMapper• .NET: Model dokumen dan model persistensi objek <p>API tingkat yang lebih tinggi ini sekarang didokumentasikan di sini: Antarmuka pemrograman tingkat tinggi untuk DynamoDB.</p>	20 Januari 2014

Perubahan	Deskripsi	Tanggal Diubah
Indeks sekunder global	DynamoDB menambahkan dukungan untuk indeks sekunder global. Seperti pada indeks sekunder lokal, Anda menentukan indeks sekunder global dengan menggunakan kunci alternatif dari tabel lalu mengeluarkan permintaan Kueri pada indeks. Tidak seperti indeks sekunder lokal, kunci partisi untuk indeks sekunder global tidak harus sama dengan tabel; kunci partisi tersebut dapat berupa atribut skalar dari tabel. Kunci urutan bersifat opsional, dan juga dapat berupa atribut skalar apa pun. Indeks sekunder global juga memiliki pengaturan throughput tersendiri, yang terpisah dari tabel induk. Untuk informasi selengkapnya, lihat Meningkatkan akses data dengan indeks sekunder dan Menggunakan Indeks Sekunder Global di DynamoDB .	12 Desember 2013

Perubahan	Deskripsi	Tanggal Diubah
Kontrol akses detail	<p>DynamoDB menambahkan dukungan untuk kontrol akses detail. Fitur ini memungkinkan pelanggan menentukan (pengguna, grup, atau peran) utama yang dapat mengakses masing-masing item dan atribut dalam tabel DynamoDB atau indeks sekunder. Aplikasi juga dapat memanfaatkan federasi identitas web untuk memindahkan tugas autentikasi pengguna ke penyedia identitas pihak ketiga, seperti Facebook, Google, atau Login with Amazon. Dengan cara ini, aplikasi (termasuk aplikasi seluler) dapat menangani pengguna dalam jumlah yang sangat besar, sekaligus memastikan bahwa tidak ada seorang pun yang dapat mengakses item data DynamoDB kecuali mereka yang diberi wewenang untuk melakukannya. Untuk informasi selengkapnya, lihat Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci.</p>	29 Oktober 2013

Perubahan	Deskripsi	Tanggal Diubah
Ukuran unit kapasitas baca 4 KB	<p>Ukuran unit kapasitas untuk baca telah meningkat dari 1 KB menjadi 4 KB. Peningkatan ini dapat mengurangi jumlah unit kapasitas baca yang disediakan yang diperlukan untuk banyak aplikasi. Sebagai contoh, sebelum rilis ini, pembacaan item 10 KB akan mengonsumsi 10 unit kapasitas baca; sekarang pembacaan 10 KB yang sama hanya akan mengonsumsi 3 unit (10 KB / 4 KB, dibulatkan ke batas 4 KB berikutnya). Untuk informasi selengkapnya, lihat Mode kapasitas baca/tulis.</p>	14 Mei 2013

Perubahan	Deskripsi	Tanggal Diubah
Pemindaian paralel	DynamoDB menambahkan dukungan untuk operasi Pindai paralel. Aplikasi sekarang dapat membagi tabel menjadi beberapa segmen logis dan memindai semua segmen secara bersamaan. Fitur ini mengurangi waktu yang diperlukan untuk menyelesaikan Pemindaian, dan sepenuhnya memanfaatkan kapasitas baca yang disediakan tabel. Untuk informasi selengkapnya, lihat Bekerja dengan pemindaian di DynamoDB .	14 Mei 2013
Indeks sekunder lokal	DynamoDB menambahkan dukungan untuk indeks sekunder lokal. Anda dapat menentukan indeks kunci urutan pada atribut non-kunci, lalu menggunakan indeks ini dalam permintaan Kueri. Dengan indeks sekunder lokal, aplikasi dapat mengambil item data di beberapa dimensi secara efisien. Untuk informasi selengkapnya, lihat Indeks Sekunder Lokal .	18 April 2013

Perubahan	Deskripsi	Tanggal Diubah
Versi API baru	Dengan rilis ini, DynamoDB memperkenalkan versi API baru (2012-08-10). Versi API sebelumnya (2011-12-05) masih didukung untuk kompatibilitas mundur dengan aplikasi yang ada. Aplikasi baru harus menggunakan versi API terkini 2012-08-10. Sebaiknya Anda memigrasikan aplikasi yang ada ke API versi 2012-08-10, karena fitur DynamoDB baru (seperti indeks sekunder lokal) tidak akan di-backport ke versi API sebelumnya. Untuk informasi selengkapnya tentang API versi 2012-08-10, lihat Referensi API Amazon DynamoDB .	18 April 2013

Perubahan	Deskripsi	Tanggal Diubah
Dukungan variabel kebijakan IAM	<p>Bahasa kebijakan akses IAM kini mendukung variabel. Ketika suatu kebijakan dievaluasi, setiap variabel kebijakan diganti dengan nilai yang disediakan oleh informasi berbasis konteks dari sesi pengguna yang diautentikasi. Anda dapat menggunakan variabel kebijakan untuk menentukan kebijakan tujuan umum tanpa mencantumkan semua komponen kebijakan secara eksplisit. Untuk informasi selengkapnya tentang variabel kebijakan, buka Variabel Kebijakan di panduan AWS Identity and Access Management Menggunakan IAM.</p> <p>Untuk contoh variabel kebijakan di DynamoDB, lihat Manajemen Identitas dan Akses untuk Amazon DynamoDB.</p>	4 April 2013

Perubahan	Deskripsi	Tanggal Diubah
contoh kode PHP diperbarui untuk AWS SDK for PHP versi 2	Versi 2 AWS SDK for PHP sekarang tersedia. Contoh kode PHP di Panduan Developer Amazon DynamoDB telah diperbarui untuk menggunakan SDK baru ini. Untuk informasi selengkapnya tentang Versi 2 untuk SDK ini, lihat AWS SDK for PHP .	23 Januari 2013
Titik akhir baru	DynamoDB memperluas ke wilayah (AS-Barat) AWS GovCloud . Untuk daftar titik akhir dan protokol layanan saat ini, lihat Wilayah dan Titik Akhir .	3 Desember 2012
Titik akhir baru	Ekspansi DynamoDB ke wilayah Amerika Selatan (Sao Paulo). Untuk daftar titik akhir yang didukung saat ini, lihat Wilayah dan Titik Akhir .	3 Desember 2012
Titik akhir baru	Ekspansi DynamoDB ke wilayah Asia Pasifik (Sydney). Untuk daftar titik akhir yang didukung saat ini, lihat Wilayah dan Titik Akhir .	13 November 2012

Perubahan	Deskripsi	Tanggal Diubah
<p>DynamoDB mengimplementasikan dukungan untuk checksum CRC32, mendukung pengambilan batch yang sangat konsisten, dan menghilangkan batasan pada pembaruan tabel secara bersamaan.</p>	<ul style="list-style-type: none">• DynamoDB menghitung checksum CRC32 dari muatan HTTP dan mengembalikan checksum ini di header baru, <code>x-amz-crc32</code>. Untuk informasi selengkapnya, lihat API tingkat rendah DynamoDB.• Secara default, operasi baca yang dilakukan oleh API <code>BatchGetItem</code> pada akhirnya konsisten. Parameter <code>ConsistentRead</code> baru di <code>BatchGetItem</code> memungkinkan Anda memilih konsistensi baca yang kuat, untuk tabel dalam permintaan. Untuk informasi selengkapnya, lihat Deskripsi.• Rilis ini menghilangkan beberapa batasan saat memperbarui banyak tabel secara bersamaan. Jumlah total tabel yang dapat diperbarui sekaligus masih 10; tetapi, tabel ini sekarang dapat berupa kombinasi dari status <code>CREATING</code>, <code>UPDATING</code>, atau <code>DELETING</code>. Selain itu, tidak ada lagi jumlah minimum untuk menambah atau mengurangi <code>ReadCapacity</code>.	<p>2 November 2012</p>

Perubahan	Deskripsi	Tanggal Diubah
	ityUnitsatau WriteCapacityUnitsuntuk tabel. Untuk informasi selengkapnya, lihat Layanan, akun, dan tabel kuota di Amazon DynamoDB .	
Dokumentasi praktik terbaik	Panduan Developer Amazon DynamoDB mengidentifikasi praktik terbaik untuk bekerja dengan tabel dan item, serta rekomendasi untuk operasi kueri dan pindai.	28 September 2012

Perubahan	Deskripsi	Tanggal Diubah
Dukungan untuk jenis data biner	<p>Selain jenis Bilangan dan String, DynamoDB kini mendukung jenis data Biner.</p> <p>Sebelum rilis ini, untuk menyimpan data biner, Anda mengonversi data biner ke dalam format string dan menyimpannya dalam DynamoDB. Selain pekerjaan konversi yang diperlukan di sisi klien, konversi sering kali meningkatkan ukuran item data yang memerlukan lebih banyak penyimpanan dan potensi kapasitas throughput yang disediakan tambahan.</p> <p>Dengan atribut jenis biner, Anda sekarang dapat menyimpan data biner apa pun, seperti data terkompresi, data terenkripsi, dan gambar. Untuk informasi selengkapnya, lihat Tipe Data. Untuk contoh kerja penanganan data tipe biner menggunakan AWS SDK, lihat bagian berikut:</p> <ul style="list-style-type: none">• Contoh: Penanganan atribut jenis biner menggunakan AWS SDK for Java document API• Contoh: Penanganan atribut jenis biner menggunakan	21 Agustus 2012

Perubahan	Deskripsi	Tanggal Diubah
	<p>API tingkat rendah AWS SDK for .NET</p> <p>Untuk dukungan tipe data biner yang ditambahkan di AWS SDK, Anda perlu mengunduh SDK terbaru dan Anda mungkin juga perlu memperbarui aplikasi yang ada. Untuk informasi tentang mengunduh SDK AWS, lihat Contoh kode .NET.</p>	
Item tabel DynamoDB dapat diperbarui dan disalin menggunakan konsol DynamoDB	Pengguna DynamoDB sekarang dapat memperbarui dan menyalin item tabel menggunakan konsol DynamoDB, selain dapat menambahkan dan menghapus item. Fungsi baru ini menyederhanakan pembuatan perubahan pada masing-masing item melalui Konsol tersebut.	14 Agustus 2012
DynamoDB menurunkan persyaratan throughput tabel minimum	DynamoDB kini mendukung persyaratan throughput tabel minimum yang lebih rendah, yaitu 1 unit kapasitas tulis dan 1 unit kapasitas baca. Untuk informasi selengkapnya, lihat topik Layanan, akun, dan tabel kuota di Amazon DynamoDB di Panduan Developer Amazon DynamoDB.	9 Agustus 2012

Perubahan	Deskripsi	Tanggal Diubah
Dukungan Signature Versi 4	DynamoDB kini mendukung Signature Versi 4 untuk mengautentikasi permintaan.	20 Juli 2012
Dukungan eksplorer tabel di Konsol DynamoDB	Konsol DynamoDB kini mendukung eksplorer tabel yang memungkinkan Anda untuk menelusuri dan mengkueri data dalam tabel Anda. Anda juga dapat memasukkan item baru atau menghapus item yang ada. Bagian Membuat tabel dan memuat data untuk contoh kode di DynamoDB dan Menggunakan konsol telah diperbarui untuk fitur ini.	22 Mei 2012
Titik akhir baru	<p>Ketersediaan DynamoDB diperluas dengan titik akhir baru di wilayah AS Barat (California Utara), wilayah AS Barat (Oregon), dan wilayah Asia Pasifik (Singapura).</p> <p>Untuk daftar titik akhir yang didukung saat ini, buka Wilayah dan Titik Akhir.</p>	24 April 2012

Perubahan	Deskripsi	Tanggal Diubah
BatchWriteItem Dukungan API	<p>DynamoDB kini mendukung API penulisan batch yang memungkinkan Anda memasukkan dan menghapus beberapa item dari satu atau beberapa tabel dalam satu panggilan API. Untuk informasi selengkapnya tentang API penulisan batch DynamoDB, lihat BatchWriteItem.</p> <p>Untuk informasi tentang bekerja dengan item dan menggunakan fitur batch write menggunakan AWS SDK, lihat Bekerja dengan item dan atribut dan Contoh kode .NET.</p>	19 April 2012
Mendokumentasikan lebih banyak kode kesalahan	Untuk informasi selengkapnya, lihat Penanganan kesalahan dengan DynamoDB .	5 April 2012
Titik akhir baru	Ekspansi DynamoDB ke wilayah Asia Pasifik (Tokyo). Untuk daftar titik akhir yang didukung saat ini, lihat Wilayah dan Titik Akhir .	29 Februari 2012

Perubahan	Deskripsi	Tanggal Diubah
Metrik ReturnedItemCount ditambahkan	Metrik baru, ReturnedItemCount, memberikan jumlah item yang dikembalikan dalam respons operasi Query atau Scan untuk DynamoDB tersedia untuk dipantau. CloudWatch Untuk informasi selengkapnya, lihat Pencatatan dan pemantauan di DynamoDB .	24 Februari 2012
Menambahkan contoh untuk nilai tambahan	DynamoDB mendukung tambahan dan pengurangan nilai numerik yang ada. Contoh menunjukkan penambahan nilai yang ada di bagian "Memperbarui Item" di: Bekerja dengan item: Java . Bekerja dengan item: .NET .	25 Januari 2012
Rilis produk awal	DynamoDB diperkenalkan sebagai layanan baru dalam rilis Beta.	18 Januari 2012

Fitur lama DynamoDB

Topik berikut adalah fitur lama yang masih didukung DynamoDB. Tidak ada pengembangan aktif yang dilakukan pada fitur-fitur ini.

Topik

- [Versi tabel global 2017.11.29 \(Legacy\)](#)

Versi tabel global 2017.11.29 (Legacy)

Important

Dokumentasi ini ditujukan untuk versi 2017.11.29 (Lama) tabel global, yang tidak boleh digunakan untuk tabel global baru. Pelanggan harus menggunakan [Tabel Global versi 2019.11.21 \(Saat Ini\)](#) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).

Untuk menentukan versi mana yang sedang Anda gunakan, lihat [Menentukan versi tabel global yang Anda gunakan](#). Untuk memperbarui tabel global yang ada dari versi 2017.11.29 (Lama) ke versi 2019.11.21 (Terbaru), lihat [Meningkatkan tabel global](#).

Topik

- [Tabel global: Cara kerjanya](#)
- [Praktik terbaik dan persyaratan untuk mengelola tabel global](#)
- [Membuat tabel global](#)
- [Memantau tabel global](#)
- [Menggunakan IAM dengan tabel global](#)

Tabel global: Cara kerjanya

Important

Dokumentasi ini ditujukan untuk versi 2017.11.29 (Lama) tabel global, yang tidak boleh digunakan untuk tabel global baru. Pelanggan harus menggunakan [Tabel Global versi](#)

[2019.11.21 \(Saat Ini\)](#) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).

Untuk menentukan versi mana yang sedang Anda gunakan, lihat [Menentukan versi tabel global yang Anda gunakan](#). Untuk memperbarui tabel global yang ada dari versi 2017.11.29 (Lama) ke versi 2019.11.21 (Terbaru), lihat [Meningkatkan tabel global](#).

Bagian berikut membantu Anda memahami konsep dan perilaku tabel global di Amazon DynamoDB.

Konsep tabel global untuk Versi 2017.11.29 (Lama)

Tabel global adalah koleksi dari satu atau beberapa tabel replika, semuanya dimiliki oleh satu akun AWS.

Tabel replika (atau replika, untuk singkatnya) adalah tabel DynamoDB tunggal yang berfungsi sebagai bagian dari tabel global. Setiap replika menyimpan set item data yang sama. Setiap tabel global tertentu hanya dapat memiliki satu tabel replika per Wilayah AWS.

Berikut ini adalah gambaran umum konseptual tentang cara pembuatan tabel global.

1. Buat tabel DynamoDB biasa, dengan DynamoDB Streams diaktifkan, di Wilayah AWS.
2. Ulangi langkah 1 untuk setiap Wilayah lain tempat Anda ingin mereplikasi data.
3. Definisikan tabel global DynamoDB berdasarkan tabel yang telah Anda buat.

AWS Management Console mengotomatiskan tugas-tugas ini, sehingga Anda dapat membuat tabel global lebih cepat dan mudah. Untuk informasi selengkapnya, lihat [Membuat tabel global](#).

Tabel global DynamoDB yang dihasilkan terdiri dari beberapa tabel replika, satu per Wilayah, yang diperlakukan DynamoDB sebagai satu unit. Setiap replika memiliki nama tabel yang sama dan skema kunci primer yang sama. Ketika aplikasi menulis data ke tabel replika di satu wilayah, DynamoDB otomatis menyebarkan aktivitas tulis tersebut ke tabel replika lain di AWS Wilayah lain.

Important

Agar data tabel Anda tetap sinkron, tabel global otomatis membuat atribut berikut untuk setiap item:

- `aws:rep:deleting`

- `aws:rep:updatetime`
- `aws:rep:updateregion`

Jangan mengubah atribut ini atau membuat atribut dengan nama yang sama.

Anda dapat menambahkan tabel replika ke tabel global sehingga dapat tersedia di Wilayah tambahan. (Untuk melakukannya, tabel global harus kosong. Dengan kata lain, tidak boleh ada tabel replika yang berisi data apa pun.)

Anda juga dapat menghapus tabel replika dari tabel global. Jika Anda melakukannya, tabel benar-benar terpisah dari tabel global. Tabel yang baru independen ini tidak lagi berinteraksi dengan tabel global, dan data tidak lagi disebarkan ke atau dari tabel global.

Warning

Perlu diketahui bahwa menghapus replika bukanlah proses atom. Untuk memastikan perilaku yang konsisten dan status yang diketahui, Anda sebaiknya mempertimbangkan untuk mengalihkan lalu lintas tulis aplikasi dari replika untuk dihapus sebelumnya. Setelah menghapusnya, tunggu hingga semua titik akhir wilayah replika menunjukkan bahwa replika tersebut tidak terkait sebelum melakukan penulisan lebih lanjut ke replika tersebut sebagai tabel regional terisolasinya sendiri.

Tugas umum

Tugas umum untuk tabel global berfungsi sebagai berikut.

Anda dapat menghapus tabel replika tabel global sama seperti tabel biasa. Ini akan menghentikan replikasi ke Wilayah itu dan menghapus salinan tabel yang disimpan di Wilayah itu. Anda tidak dapat memutuskan replikasi dan membuat salinan tabel ada sebagai entitas independen.

Note

Anda tidak akan dapat menghapus tabel sumber hingga setidaknya 24 jam setelah tabel tersebut digunakan untuk memulai Wilayah baru. Jika mencoba menghapusnya terlalu cepat, Anda akan menerima pesan kesalahan.

Konflik dapat terjadi jika aplikasi memperbarui item yang sama di Wilayah yang berbeda pada waktu yang sama. Untuk membantu memastikan konsistensi akhirnya, tabel global DynamoDB menggunakan metode “penulis terakhir menang” untuk merekonsiliasi antara pembaruan yang dilakukan secara bersamaan. Semua replika akan menyetujui pembaruan terkini dan berkumpul menuju status ketika semua replika memiliki data yang identik.

Note

Ada beberapa cara untuk menghindari konflik, antara lain:

- Menggunakan kebijakan IAM untuk hanya mengizinkan penulisan ke tabel di satu wilayah.
- Menggunakan kebijakan IAM untuk mengarahkan pengguna hanya ke satu wilayah dan menjadikan wilayah lainnya sebagai siaga, atau secara bergantian merutekan pengguna ganjil ke satu wilayah dan pengguna genap ke wilayah lain.
- Menghindari penggunaan pembaruan non-idempoten seperti `Bookmark = Bookmark + 1`, dan mendukung pembaruan statis seperti `Bookmark=25`.

Memantau tabel global

Anda dapat menggunakan CloudWatch untuk mengamati metrik `ReplicationLatency`. Metrik ini melacak waktu yang telah berlalu antara saat item yang diperbarui muncul di aliran DynamoDB untuk satu tabel replika, dan kapan item itu muncul di replika lain di tabel global. `ReplicationLatency` dinyatakan dalam milidetik dan dipancarkan untuk setiap pasangan `Source-region` dan `Destination-region`. Ini adalah satu-satunya CloudWatch metrik yang disediakan oleh Global Tables v2.

Latensi yang akan Anda lihat bergantung pada jarak antara Wilayah yang Anda pilih, serta variabel lainnya. Latensi dalam kisaran 0,5 hingga 2,5 detik untuk Wilayah mungkin umum terjadi dalam wilayah geografis yang sama.

Waktu Untuk Tayang (TTL)

Anda dapat menggunakan Waktu Untuk Tayang (TTL) untuk menentukan nama atribut yang nilainya menunjukkan waktu kedaluwarsa item tersebut. Nilai ini ditentukan sebagai angka dalam detik sejak dimulainya zaman Unix.

Dengan versi warisan tabel global, penghapusan TTL tidak secara otomatis direplikasi di seluruh replika lainnya. Ketika item dihapus melalui aturan TTL, pekerjaan itu dilakukan tanpa menggunakan Unit Tulis.

Perlu diketahui jika tabel sumber dan target memiliki kapasitas tulis yang disediakan yang sangat rendah, hal ini dapat menyebabkan throttling karena penghapusan TTL memerlukan kapasitas tulis.

Aliran dan transaksi dengan tabel global

Setiap tabel global menghasilkan aliran independen berdasarkan semua tulisannya, terlepas dari titik asal untuk penulisan tersebut. Anda dapat memilih untuk menggunakan aliran DynamoDB ini di satu Wilayah atau di semua Wilayah secara independen.

Jika Anda ingin penulisan lokal yang diproses tetapi bukan penulisan yang direplikasi, Anda dapat menambahkan atribut wilayah Anda sendiri ke setiap item. Kemudian, Anda dapat menggunakan filter peristiwa Lambda untuk hanya menginvokasi Lambda untuk penulisan di Wilayah lokal.

Operasi transaksional memberikan jaminan ACID (Atomicity, Consistency, Isolation, dan Durability) HANYA di Wilayah tempat awal tulis dilakukan. Transaksi tidak didukung di seluruh Wilayah dalam tabel global.

Misalnya, jika Anda memiliki tabel global dengan replika di Wilayah AS Timur (Ohio) dan AS Barat (Oregon) dan melakukan TransactWriteItems operasi di Wilayah AS Timur (Ohio), Anda dapat mengamati transaksi yang diselesaikan sebagian di Wilayah AS Barat (Oregon) saat perubahan direplikasi. Perubahan hanya akan direplikasi ke Wilayah lain setelah perubahan telah dilakukan di Wilayah sumber.

Note

- Tabel global “write-around” DynamoDB Accelerator dengan memperbarui DynamoDB secara langsung. Akibatnya, DAX tidak akan mengetahui jika sedang menyimpan data usang. Cache DAX hanya akan disegarkan ketika TTL cache kedaluwarsa.
- Tanda pada tabel global tidak disebarluaskan secara otomatis.

Throughput baca dan tulis

Tabel global mengelola throughput baca dan tulis dengan cara berikut.

- Kapasitas tulis harus sama di semua instans tabel di seluruh Wilayah.
- Dengan Versi 2019.11.21 (Terbaru), jika tabel diatur untuk mendukung penskalaan otomatis atau berada dalam mode sesuai permintaan, maka kapasitas tulis tetap sinkron secara otomatis. Jumlah kapasitas tulis saat ini yang disediakan di setiap Wilayah akan naik dan turun secara independen dalam pengaturan penskalaan otomatis yang disinkronkan tersebut. Jika tabel ditempatkan dalam mode sesuai permintaan, mode tersebut akan disinkronkan ke replika lainnya.
- Kapasitas baca dapat berbeda antarWilayah karena baca mungkin tidak sama. Saat menambahkan replika global ke tabel, kapasitas Wilayah sumber disebarakan. Setelah pembuatan, Anda dapat menyesuaikan kapasitas baca untuk satu replika, dan pengaturan baru ini tidak ditransfer ke sisi lain.

Konsistensi dan resolusi konflik

Setiap perubahan yang dibuat pada item apa pun di tabel replika mana pun akan direplikasi ke semua replika lain dalam tabel global yang sama. Dalam tabel global, item yang baru ditulis biasanya disebarakan ke semua tabel replika dalam hitungan detik.

Dengan tabel global, setiap tabel replika menyimpan set item data yang sama. DynamoDB tidak mendukung replikasi parsial hanya beberapa item.

Aplikasi dapat membaca dan menulis data ke tabel replika mana pun. DynamoDB mendukung bacaan akhir konsisten di seluruh Wilayah, tetapi tidak mendukung bacaan sangat konsisten di seluruh Wilayah. Jika aplikasi Anda hanya menggunakan bacaan akhir konsisten dan hanya mengeluarkan aktivitas baca terhadap satu Wilayah AWS, aplikasi akan berjalan tanpa perubahan apa pun. Namun, jika aplikasi Anda membutuhkan bacaan sangat konsisten, aplikasi harus melakukan semua penulisan dan bacaan sangat konsisten di Wilayah yang sama. Jika tidak, jika Anda menulis ke satu Wilayah dan membaca dari Wilayah lain, maka respons baca mungkin menyertakan data usang yang tidak mencerminkan hasil penulisan yang baru saja diselesaikan di Wilayah lain.

Konflik dapat terjadi jika aplikasi memperbarui item yang sama di Wilayah yang berbeda pada waktu yang sama. Untuk membantu memastikan konsistensi akhir, tabel global DynamoDB menggunakan rekonsiliasi penulis terakhir menang antara pembaruan serentak, dengan DynamoDB melakukan upaya terbaik untuk menentukan penulis terakhir. Dengan mekanisme resolusi konflik ini, semua replika akan menyetujui pembaruan terkini dan berkumpul menuju status ketika semua replika memiliki data yang identik.

Ketersediaan dan daya tahan

Jika satu Wilayah AWS menjadi terisolasi atau terdegradasi, aplikasi Anda dapat mengalihkan ke Wilayah yang berbeda dan melakukan aktivitas baca dan tulis terhadap tabel replika yang berbeda. Anda dapat menerapkan logika bisnis khusus untuk menentukan kapan harus mengalihkan permintaan ke Wilayah lain.

Jika suatu Wilayah menjadi terisolasi atau terdegradasi, DynamoDB melacak setiap penulisan yang telah dilakukan tetapi belum disebar ke semua tabel replika. Setelah Wilayah kembali online, DynamoDB melanjutkan penyebaran penulisan yang tertunda dari Wilayah tersebut ke tabel replika di Wilayah lain. DynamoDB juga melanjutkan penyebaran penulisan dari tabel replika lain ke Wilayah yang saat ini kembali online. Semua aktivitas tulis yang berhasil sebelumnya akan disebar pada akhirnya, terlepas dari berapa lama Wilayah tersebut terisolasi.

Praktik terbaik dan persyaratan untuk mengelola tabel global

Dengan menggunakan tabel global Amazon DynamoDB, Anda dapat mereplikasi data tabel di AWS Wilayah. Tabel replika dan indeks sekunder di tabel global Anda harus memiliki pengaturan kapasitas tulis yang identik untuk memastikan replikasi data yang tepat.

Topik

- [Versi tabel global](#)
- [Persyaratan untuk menambahkan tabel replika baru](#)
- [Praktik terbaik dan persyaratan untuk mengelola kapasitas](#)

Versi tabel global

Ada dua versi tabel global DynamoDB yang tersedia: [Tabel Global versi 2019.11.21 \(Saat Ini\)](#) dan [Versi tabel global 2017.11.29 \(Legacy\)](#). Pelanggan harus menggunakan Tabel Global versi 2019.11.21 (Saat Ini) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).

Untuk menentukan versi mana yang sedang Anda gunakan, lihat [Menentukan versi tabel global yang Anda gunakan](#). Untuk memperbarui tabel global yang ada dari Versi 2017.11.29 (Lama) ke Versi 2019.11.21 (Terbaru), lihat [Meningkatkan tabel global](#).

Persyaratan untuk menambahkan tabel replika baru

Jika Anda ingin menambahkan tabel replika baru ke tabel global, masing-masing syarat berikut harus benar:

- Tabel harus memiliki kunci partisi yang sama seperti semua replika lainnya.
- Tabel harus memiliki pengaturan manajemen kapasitas tulis yang sama yang ditentukan.
- Tabel harus memiliki nama yang sama seperti semua replika lainnya.
- Tabel harus memiliki DynamoDB Streams yang diaktifkan, dengan aliran yang berisi gambar item baru dan lama.
- Tabel replika baru atau yang sudah ada di tabel global tidak boleh berisi data.

Jika indeks sekunder global ditentukan, syarat berikut juga harus dipenuhi:

- Indeks sekunder global harus memiliki nama yang sama.
- Indeks sekunder global harus memiliki kunci partisi yang sama dan kunci urutan (jika ada).

Important

Pengaturan kapasitas tulis harus diatur secara konsisten di seluruh tabel replika tabel global Anda dan indeks sekunder yang cocok. Untuk memperbarui pengaturan kapasitas tulis untuk tabel global Anda, sebaiknya Anda menggunakan konsol DynamoDB atau operasi API `UpdateGlobalTableSettings`. `UpdateGlobalTableSettings` menerapkan perubahan pada pengaturan kapasitas tulis ke semua tabel replika dan indeks sekunder yang cocok dalam tabel global secara otomatis. Jika menggunakan operasi `UpdateTable`, `RegisterScalableTarget`, atau `PutScalingPolicy`, Anda harus menerapkan perubahan pada setiap tabel replika dan indeks sekunder yang cocok satu per satu. Untuk informasi selengkapnya, lihat [UpdateGlobalTableSettings](#) di Referensi [Amazon DynamoDB API](#).

Sebaiknya Anda mengaktifkan penskalaan otomatis untuk mengelola pengaturan kapasitas tulis yang disediakan. Jika lebih memilih untuk mengelola pengaturan kapasitas tulis secara manual, Anda harus menyediakan unit kapasitas tulis yang direplikasi yang setara untuk semua tabel replika Anda. Sediakan juga unit kapasitas tulis yang direplikasi yang setara untuk indeks sekunder yang cocok di seluruh tabel global Anda.

Anda juga harus memiliki izin AWS Identity and Access Management (IAM) yang sesuai. Untuk informasi selengkapnya, lihat [Menggunakan IAM dengan tabel global](#).

Praktik terbaik dan persyaratan untuk mengelola kapasitas

Pertimbangkan hal berikut ketika mengelola pengaturan kapasitas untuk replika tabel di DynamoDB.

Menggunakan penskalaan otomatis DynamoDB

Menggunakan penskalaan otomatis DynamoDB adalah cara yang direkomendasikan untuk mengelola pengaturan kapasitas throughput untuk tabel replika yang menggunakan mode ditetapkan. Penskalaan otomatis DynamoDB otomatis menyesuaikan unit kapasitas baca (RCU) dan unit kapasitas tulis (WCU) untuk setiap tabel replika berdasarkan beban kerja aplikasi aktual. Untuk informasi selengkapnya, lihat [Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB](#).

Jika Anda membuat tabel replika menggunakan AWS Management Console, penskalaan otomatis diaktifkan secara default untuk setiap tabel replika, dengan pengaturan penskalaan otomatis default untuk mengelola unit kapasitas baca dan unit kapasitas tulis.

Perubahan pada pengaturan penskalaan otomatis untuk tabel replika atau indeks sekunder yang dibuat melalui konsol DynamoDB atau menggunakan panggilan `UpdateGlobalTableSettings` diterapkan untuk semua tabel replika dan indeks sekunder yang cocok dalam tabel global secara otomatis. Perubahan ini menimpa pengaturan penskalaan otomatis yang sudah ada. Hal ini memastikan pengaturan kapasitas tulis yang disediakan konsisten di seluruh tabel replika dan indeks sekunder di tabel global Anda. Jika menggunakan operasi panggilan `UpdateTable`, `RegisterScalableTarget`, atau `PutScalingPolicy`, Anda harus menerapkan perubahan pada setiap tabel replika dan indeks sekunder yang cocok satu per satu.

Note

Jika penskalaan otomatis tidak memenuhi perubahan kapasitas aplikasi Anda (beban kerja yang tidak dapat diprediksi), atau jika Anda tidak ingin mengonfigurasi pengaturannya (pengaturan target untuk ambang batas minimum, maksimum, atau pemanfaatan), Anda dapat menggunakan mode sesuai permintaan untuk mengelola kapasitas tabel global. Untuk informasi selengkapnya, lihat [Mode sesuai permintaan](#).

Jika Anda mengaktifkan mode sesuai permintaan pada tabel global, konsumsi unit permintaan tulis yang direplikasi (rWCU) akan konsisten dengan cara rWCU disediakan.

Sebagai contoh, jika Anda melakukan 10 aktivitas tulis ke tabel lokal yang direplikasi di dua Wilayah tambahan, Anda akan menggunakan 60 unit permintaan tulis ($10 + 10 + 10 = 30$; $30 \times 2 = 60$). 60 unit permintaan tulis yang dikonsumsi termasuk penulisan tambahan yang dikonsumsi oleh tabel global Versi 2017.11.29 (Lama) untuk memperbarui atribut `aws:rep:deleting`, `aws:rep:updatetime`, dan `aws:rep:updateregion`.

Mengelola kapasitas secara manual

Jika memutuskan untuk tidak menggunakan penskalaan otomatis DynamoDB, Anda harus secara manual menetapkan pengaturan kapasitas baca dan kapasitas tulis pada setiap tabel replika dan indeks sekunder.

Unit kapasitas tulis yang direplikasi (rWCU) yang disediakan pada setiap tabel replika harus diatur ke jumlah total rWCU yang diperlukan untuk penulisan aplikasi di seluruh Wilayah dikalikan dua. Hal ini mengakomodasi aktivitas tulis aplikasi yang terjadi di Wilayah lokal dan aktivitas tulis aplikasi yang direplikasi yang berasal dari Wilayah lain Misalnya, anggaplah Anda mengharapkan 5 aktivitas tulis per detik untuk tabel replika Anda di Ohio dan 5 aktivitas tulis per detik untuk tabel replika Anda di Virginia Utara. Dalam hal ini, Anda harus menyediakan 20 RWCU untuk setiap tabel replika ($5 + 5 = 10$; $10 \times 2 = 20$).

Untuk memperbarui pengaturan kapasitas tulis untuk tabel global Anda, sebaiknya Anda menggunakan konsol DynamoDB atau operasi API `UpdateGlobalTableSettings`. `UpdateGlobalTableSettings` menerapkan perubahan pada pengaturan kapasitas tulis ke semua tabel replika dan indeks sekunder yang cocok dalam tabel global secara otomatis. Jika menggunakan operasi `UpdateTable`, `RegisterScalableTarget`, atau `PutScalingPolicy`, Anda harus menerapkan perubahan pada setiap tabel replika dan indeks sekunder yang cocok satu per satu. Untuk informasi selengkapnya, lihat [Referensi API Amazon DynamoDB](#).

Note

Untuk memperbarui pengaturan (`UpdateGlobalTableSettings`) tabel global di DynamoDB, Anda harus memiliki izin `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy`, dan `application-autoscaling:DeregisterScalableTarget`. Lihat informasi yang lebih lengkap di [Menggunakan IAM dengan tabel global](#).

Membuat tabel global

Important

Dokumentasi ini ditujukan untuk versi 2017.11.29 (Lama) tabel global, yang tidak boleh digunakan untuk tabel global baru. Pelanggan harus menggunakan [Tabel Global versi 2019.11.21 \(Saat Ini\)](#) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).

Untuk menentukan versi mana yang sedang Anda gunakan, lihat [Menentukan versi tabel global yang Anda gunakan](#). Untuk memperbarui tabel global yang ada dari versi 2017.11.29 (Lama) ke versi 2019.11.21 (Terbaru), lihat [Meningkatkan tabel global](#).

Bagian ini menjelaskan cara membuat tabel menggunakan konsol Amazon DynamoDB atau AWS Command Line Interface (AWS CLI).

Topik

- [Membuat tabel global \(konsol\)](#)
- [Membuat tabel global \(AWS CLI\)](#)

Membuat tabel global (konsol)

Ikuti langkah-langkah ini untuk membuat tabel global menggunakan konsol. Contoh berikut membuat tabel global dengan tabel replika di Amerika Serikat dan Eropa.

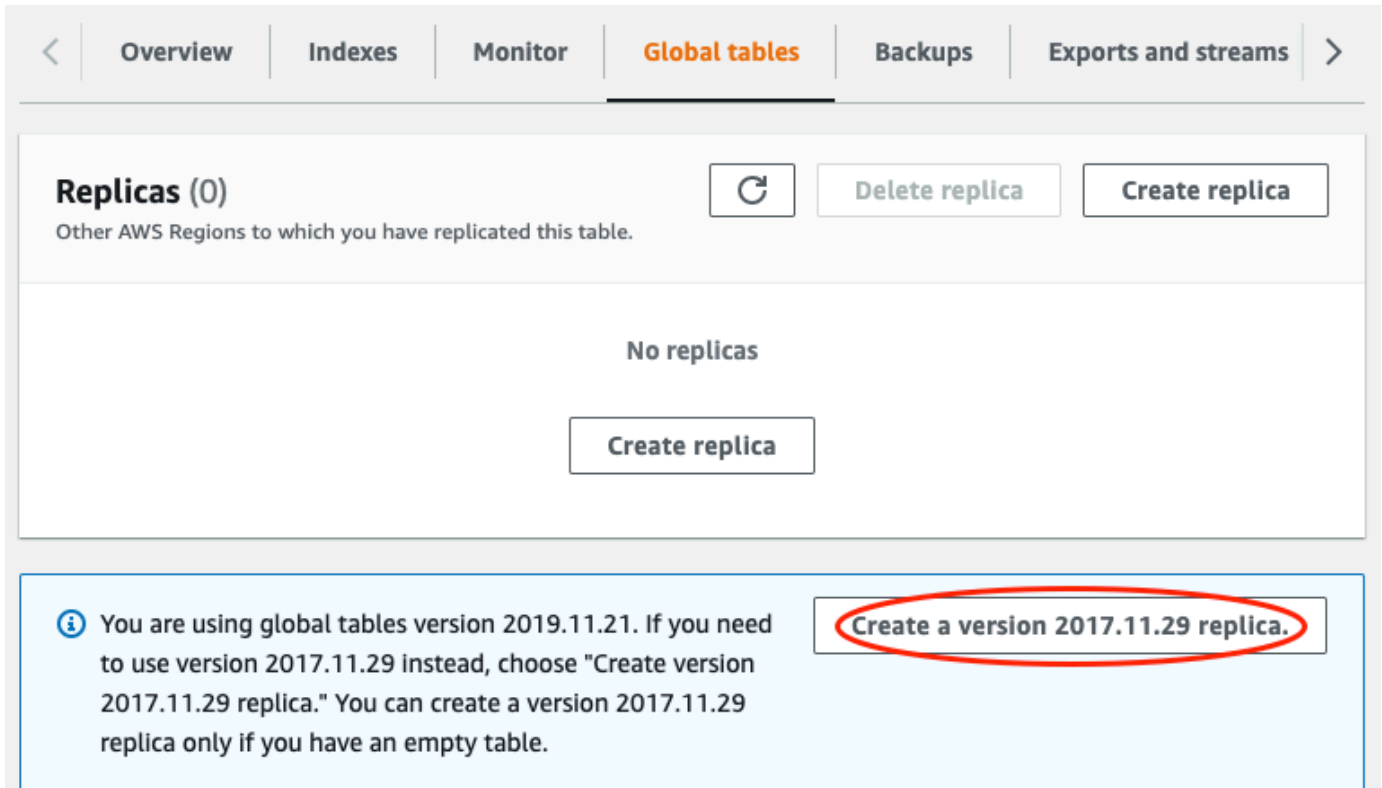
1. Buka konsol DynamoDB di <https://console.aws.amazon.com/dynamodb/home>. Untuk contoh ini, pilih Wilayah us-east-2 (AS Timur Ohio) Region.
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih Buat Tabel.

Untuk Nama tabel, masukkan **Music**.

Untuk Kunci primer, masukkan **Artist**. Pilih Tambahkan kunci urutan, dan masukkan **SongTitle**. (**Artist** dan **SongTitle** harus berupa string.)

Untuk membuat tabel, pilih Buat. Tabel ini berfungsi sebagai tabel replika pertama dalam tabel global baru. Ini adalah prototipe untuk tabel replika lain yang Anda tambahkan nantinya.

- Pilih tab Tabel Global, lalu pilih Buat replika Versi 2017.11.29 (Lama).



- Dari dropdown Wilayah replikasi yang Tersedia, pilih AS Barat (Oregon).

Konsol memeriksa untuk memastikan bahwa tidak ada tabel dengan nama yang sama di Wilayah yang dipilih. Jika ada tabel dengan nama yang sama, Anda harus menghapus tabel yang ada sebelum dapat membuat tabel replika baru di Wilayah tersebut.

- Pilih Buat Replika. Langkah ini memulai proses pembuatan tabel di AS Barat (Oregon);.

Tab Tabel Global untuk tabel yang dipilih (dan untuk setiap tabel replika lainnya) menunjukkan bahwa tabel telah direplikasi di beberapa Wilayah.

- Sekarang tambahkan Wilayah lain sehingga tabel global Anda direplikasi dan disinkronkan di seluruh Amerika Serikat dan Eropa. Untuk melakukannya, ulangi langkah 5, tetapi kali ini tentukan Eropa (Frankfurt), bukan AS Barat (Oregon).
- Anda masih harus menggunakan AWS Management Console di Wilayah AS Timur (Ohio). Pilih Item di menu navigasi kiri, pilih tabel Musik, lalu pilih Buat Item.
 - Untuk Artist, masukkan **item_1**.
 - Untuk SongTitle, masukkan **Song Value 1**.
 - Untuk menulis item, pilih Buat item.

- Setelah beberapa saat, item direplikasi di ketiga Wilayah tabel global Anda. Untuk memverifikasi ini, di pilih Wilayah di sudut kanan atas di konsol, pilih Eropa (Frankfurt). Tabel Music di Eropa (Frankfurt) akan berisi item baru.
- Ulangi langkah 9 dan pilih AS Barat (Oregon) untuk memverifikasi replikasi di wilayah itu.

Membuat tabel global (AWS CLI)

Ikuti langkah-langkah ini untuk membuat tabel global Music menggunakan AWS CLI. Contoh berikut membuat tabel global, dengan tabel replika di Amerika Serikat dan Eropa.

- Buat tabel baru (Music) di AS Timur (Ohio), dengan DynamoDB Streams diaktifkan (NEW_AND_OLD_IMAGES).


```
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
```

- Buat tabel Music identik di AS Timur (Virginia Utara).

```
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. Buat tabel global (Music) yang terdiri dari tabel replika di Wilayah us-east-2 dan us-east-1.

```
aws dynamodb create-global-table \  
  --global-table-name Music \  
  --replication-group RegionName=us-east-2 RegionName=us-east-1 \  
  --region us-east-2
```

 Note

Nama tabel global (Music) harus cocok dengan nama setiap tabel replika (Music). Untuk informasi selengkapnya, lihat [Praktik terbaik dan persyaratan untuk mengelola tabel global](#).

4. Buat tabel lain di Eropa (Irlandia), dengan pengaturan yang sama seperti yang Anda buat pada langkah 1 dan langkah 2.

```
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
```

Setelah melakukan langkah ini, tambahkan tabel baru untuk tabel global Music.

```
aws dynamodb update-global-table \  
  --global-table-name Music \  
  --replica-updates 'Create={RegionName=eu-west-1}' \  
  --region us-east-2
```

5. Untuk memverifikasi bahwa replikasi berfungsi, tambahkan item baru ke tabel Music di AS Timur (Ohio).

```
aws dynamodb put-item \  
  --region us-east-2
```

```
--table-name Music \  
--item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \  
--region us-east-2
```

6. Tunggu beberapa detik, lalu periksa untuk melihat apakah item tersebut telah berhasil direplikasi ke AS Timur (Virginia Utara) dan Eropa (Irlandia).

```
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
```

Memantau tabel global

Important

Dokumentasi ini ditujukan untuk versi 2017.11.29 (Lama) tabel global, yang tidak boleh digunakan untuk tabel global baru. Pelanggan harus menggunakan [Tabel Global versi 2019.11.21 \(Saat Ini\)](#) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).

Untuk menentukan versi mana yang sedang Anda gunakan, lihat [Menentukan versi tabel global yang Anda gunakan](#). Untuk memperbarui tabel global yang ada dari versi 2017.11.29 (Lama) ke versi 2019.11.21 (Terbaru), lihat [Meningkatkan tabel global](#).

Anda dapat menggunakan Amazon CloudWatch untuk memantau perilaku dan kinerja tabel global. Amazon DynamoDB menerbitkan metrik `ReplicationLatency` dan `PendingReplicationCount` untuk setiap replika dalam tabel global.

- **ReplicationLatency**—Waktu yang berlalu antara saat item yang diperbarui muncul dalam aliran DynamoDB untuk satu tabel replika, dan saat item tersebut muncul di replika lain dalam

tabel global. `ReplicationLatency` dinyatakan dalam milidetik dan dipancarkan untuk setiap pasangan Wilayah sumber dan tujuan.

Selama operasi normal, `ReplicationLatency` akan cukup konstan. Nilai `ReplicationLatency` yang tinggi dapat menunjukkan bahwa pembaruan dari satu replika tidak disebarkan ke tabel replika lain pada waktu yang tepat. Seiring waktu, hal ini dapat mengakibatkan tabel replika lainnya tertinggal karena tidak lagi menerima pembaruan secara konsisten. Dalam kasus ini, Anda harus memverifikasi bahwa unit kapasitas baca (RCU) dan unit kapasitas tulis (WCU) identik untuk masing-masing tabel replika. Selain itu, saat memilih pengaturan WCU, ikuti rekomendasi di [Versi tabel global](#).

`ReplicationLatency` dapat meningkat jika Wilayah AWS terdegradasi dan Anda memiliki tabel replika di wilayah tersebut. Dalam kasus ini, Anda dapat mengalihkan aktivitas baca dan tulis aplikasi Anda ke Wilayah AWS lain untuk sementara waktu.

- **PendingReplicationCount**—Jumlah pembaruan item yang ditulis ke satu tabel replika, tetapi belum ditulis ke replika lain dalam tabel global. `PendingReplicationCount` dinyatakan dalam jumlah item dan dipancarkan untuk setiap pasangan Wilayah sumber dan tujuan.

Selama operasi normal, `PendingReplicationCount` akan sangat rendah. Jika `PendingReplicationCount` meningkat untuk jangka waktu yang lama, selidiki apakah pengaturan kapasitas tulis yang disediakan tabel replika Anda mencukupi untuk beban kerja saat ini.

`PendingReplicationCount` dapat meningkat jika Wilayah AWS terdegradasi dan Anda memiliki tabel replika di wilayah tersebut. Dalam kasus ini, Anda dapat mengalihkan aktivitas baca dan tulis aplikasi Anda ke Wilayah AWS lain untuk sementara waktu.

Lihat informasi yang lebih lengkap di [Dimensi dan Metrik DynamoDB](#).

Menggunakan IAM dengan tabel global

Important

Dokumentasi ini ditujukan untuk versi 2017.11.29 (Lama) tabel global, yang tidak boleh digunakan untuk tabel global baru. Pelanggan harus menggunakan [Tabel Global versi 2019.11.21 \(Saat Ini\)](#) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).

Untuk menentukan versi mana yang sedang Anda gunakan, lihat [Menentukan versi tabel global yang Anda gunakan](#). Untuk memperbarui tabel global yang ada dari versi 2017.11.29 (Lama) ke versi 2019.11.21 (Terbaru), lihat [Meningkatkan tabel global](#).

Ketika Anda membuat tabel global untuk pertama kalinya, Amazon DynamoDB otomatis membuat peran tertaut layanan AWS Identity and Access Management (IAM) untuk Anda. Peran ini bernama [AWSServiceRoleForDynamoDBReplication](#), dan memungkinkan DynamoDB untuk mengelola replikasi lintas-Wilayah untuk tabel global atas nama Anda. Jangan hapus peran tertaut layanan ini. Jika Anda melakukannya, semua tabel global Anda tidak akan berfungsi lagi.

Untuk informasi selengkapnya tentang peran tertaut layanan, lihat [Menggunakan peran tertaut layanan](#) di Panduan Pengguna IAM.

Untuk membuat dan memelihara tabel global di DynamoDB, Anda harus memiliki izin `dynamodb:CreateGlobalTable` untuk mengakses berikut ini:

- Tabel replika yang ingin Anda tambahkan.
- Setiap replika yang ada yang sudah menjadi bagian dari tabel global.
- Tabel global itu sendiri.

Untuk memperbarui pengaturan (`UpdateGlobalTableSettings`) tabel global di DynamoDB, Anda harus memiliki izin `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy`, dan `application-autoscaling:DeregisterScalableTarget`.

Izin `application-autoscaling:DeleteScalingPolicy` dan `application-autoscaling:DeregisterScalableTarget` diperlukan saat memperbarui kebijakan penskalaan yang ada. Hal ini agar layanan tabel global dapat menghapus kebijakan penskalaan lama sebelum melampirkan kebijakan baru ke tabel atau indeks sekunder.

Jika menggunakan kebijakan IAM untuk mengelola akses ke satu tabel replika, Anda harus menerapkan kebijakan yang identik pada semua replika lain dalam tabel global. Praktik ini membantu Anda mempertahankan model izin yang konsisten di semua tabel replika.

Dengan menggunakan kebijakan IAM yang identik pada semua replika dalam tabel global, Anda juga dapat menghindari pemberian akses baca dan tulis yang tidak diinginkan ke data tabel global Anda.

Misalnya, pertimbangkan pengguna yang hanya memiliki akses ke satu replika dalam tabel global. Jika pengguna tersebut dapat menulis ke replika ini, DynamoDB menyebarkan aktivitas tulis ini ke semua tabel replika lainnya. Akibatnya, pengguna tersebut (secara tidak langsung) dapat menulis ke semua replika lain dalam tabel global. Skenario ini dapat dihindari dengan menggunakan kebijakan IAM yang konsisten pada semua tabel replika.

Contoh: Izinkan CreateGlobalTable tindakan

Sebelum dapat menambahkan replika ke tabel global, Anda harus memiliki izin `dynamodb:CreateGlobalTable` untuk tabel global dan untuk masing-masing tabel replikanya.

Kebijakan IAM berikut memberikan izin untuk memungkinkan tindakan `CreateGlobalTable` pada semua tabel.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["dynamodb:CreateGlobalTable"],
      "Resource": "*"
    }
  ]
}
```

Contoh: Izinkan UpdateGlobalTable,, application-autoscaling: DescribeLimits, dan application-autoscaling: actions DeleteScalingPolicy DeregisterScalableTarget

Untuk memperbarui pengaturan (`UpdateGlobalTableSettings`) tabel global di DynamoDB, Anda harus memiliki izin `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy`, dan `application-autoscaling:DeregisterScalableTarget`.

Kebijakan IAM berikut memberikan izin untuk memungkinkan tindakan `UpdateGlobalTableSettings` pada semua tabel.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
    "Effect": "Allow",
    "Action": [
        "dynamodb:UpdateGlobalTable",
        "dynamodb:DescribeLimits",
        "application-autoscaling:DeleteScalingPolicy",
        "application-autoscaling:DeregisterScalableTarget"
    ],
    "Resource": "*"
}
]
```

Contoh: Izinkan `CreateGlobalTable` tindakan untuk nama tabel global tertentu dengan replika yang diizinkan di wilayah tertentu saja

Kebijakan IAM berikut memberikan izin untuk memungkinkan tindakan `CreateGlobalTable` membuat tabel global bernama `Customers` dengan replika di dua Wilayah.

```
{
  "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"
      ]
    }
  ]
}
```

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.